

# 國立交通大學

資訊工程系

博士論文

網格化簡與混合式顯像技術及其在即時顯像的應用

Mesh Simplification and Hybrid Rendering Techniques for  
Real-Time Rendering



研究生：陳治君

指導教授：莊榮宏 教授

中華民國九十五年十月

網格化簡與混合式顯像技術及其在即時顯像的應用  
Mesh Simplification and Hybrid Rendering Techniques for Real-Time Rendering

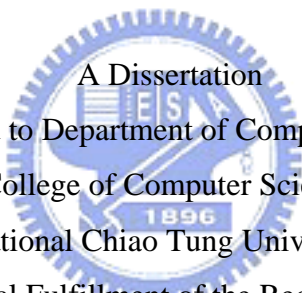
研究生：陳治君

Student : Chih-Chun Chen

指導教授：莊榮宏

Advisor : Jung-Hong Chuang

國立交通大學  
資訊工程系  
博士論文

A Dissertation  
Submitted to Department of Computer Science  
College of Computer Science  
National Chiao Tung University  
in partial Fulfillment of the Requirements  
for the Degree of  
Doctor of Philosophy  
in  
Computer Science

October 2006

Hsinchu, Taiwan, Republic of China

中華民國九十五年十月

# 網格化簡與混合式顯像技術及其在即時顯像的應用

學生：陳治君

指導教授：莊榮宏 教授

國立交通大學資訊工程學系博士班

## 摘 要

即時顯像是互動應用，如：遊戲、虛擬實境以及虛擬模擬中核心的技術。即時顯像技術包含了幾何為基礎、影像為基礎以及混合式顯像技術。這本論文中我們首先提出兩個全新的幾何顯像式技術，而後提出一個混和式顯像系統架構。在幾個以幾何為基礎的即時顯像之關鍵技術中，多層次模型建構技術為其中極其重要而且相當廣為流傳的即時顯像技術。我們發展出一個全新、簡單且有效的貼圖適應性調整機制來消除在漸進式網格模型配合貼圖時產生的貼圖扭曲現象。以此機制為基礎，我們提出一個新的幾何化簡之曲面與曲面間誤差估測方式，能與現有的方法相競爭甚至更好。混合式顯像技術在遊戲及虛擬實境中已被證實為非常有用的幾何式顯像的輔助技術。我們提出一個混合式顯像機制，利用多層次模型與貼圖並探索可見性的區域連貫性，以額外的儲存空間及預先讀取來做影像品質與顯像效率的取捨。

為了支援漸進式網格模型的貼圖，我們常讓整個漸進式網格序列共享一個共有的貼圖。這個共有貼圖能用適當的幾何參數化方法考慮幾何與貼圖壓縮值，甚至考慮幾何邊折疊時的貼圖誤差得來。我們發現即使用很好的參數化貼圖，漸進式網格模型貼上貼圖時很容易產生明顯的貼圖扭曲現象，這主要是因幾何的變化與貼圖硬體的線性內差所致。在這論文中，我們提出一個全新、簡單且很有效率的方法針對每個幾何邊折疊來調整貼圖內容，以消除貼圖扭曲現象。貼圖之適應性調整機制及其反運算都是區域性及累進式運算且能以現有的圖學加速卡直

接支援。我們另提出索引貼圖的機制來降低當調整貼圖時可能造成取樣不足進而產生的影像糊化現象。實驗結果顯示，在測試的例子中，貼圖扭曲現象幾乎都能被貼圖之適應性調整機制所消除。基於貼圖之適應性調整機制，我們提出一個新的以映像為基礎的誤差估測法，在不好的參數貼圖下，能夠比 APS 或 QEM 提供更精確的幾何誤差量測。從實驗結果得知，我們提出的方法比 APS 好上許多，而幾乎接近 QEM 的結果。

在我們所提出的混合式顯像架構下，場景先被分割成一些蜂巢式的空間，再對蜂巢內的物體用一般的顯像技術來顯像，而蜂巢外的物體則用多層次貼圖幾何配合投影式貼圖法來顯像。多層次貼圖幾何為一以物體為基礎，由原始幾何依據所得到的深度影像化簡而得來，而深度影像則是在蜂巢與其鄰近蜂巢中心點顯像取得。利用這個多層次貼圖幾何，許多在混合式顯像技術中常被發現的問題，如：因物體與物體間遮擋而產生的洞的問題，以及因解析度不一致所產生的裂縫問題都能被消除。而因為物體自身遮擋產生的洞的問題，也能被限制在使用者指定的範圍內。在我們的顯像架構下，數個由上百萬個三角片所組成的複雜場景，都可達到高於每秒鐘 600 個畫面的平均顯像速率，而且只有些微的影像失真。



MESH SIMPLIFICATION AND HYBRID RENDERING TECHNIQUES  
FOR REAL-TIME RENDERING

A Dissertation

Submitted to Department of Computer Science

College of Computer Science

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

in

Computer Science

Student: Chih-Chun Chen

Adviser: Prof. Jung-Hong Chuang

October 2006

Hsinchu, Taiwan, Republic of China



I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

---

Prof. Jung-Hong Chuang    Principal Adviser

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.



---

Prof. Zen-Chung Shih

I certify that I have read this dissertation and that, in my opinion, it is fully adequate in scope and quality as a dissertation for the degree of Doctor of Philosophy.

---

Assistant Prof. I-Chen Lin

Approved for the University Committee on Graduate Studies.

# Acknowledgements

I would like to thank my wife and my family. Without their supports I may not be able to get the Ph.D. degree. I deeply appreciate my adviser Prof. J. H. Chuang not only for his creative suggestions and discussions on my studies, but also his insightful advisements for the philosophy of life. Finally, I also like to thank all the members of Computer Graphics & Geometry Modeling Lab. for coloring my life during the studies.



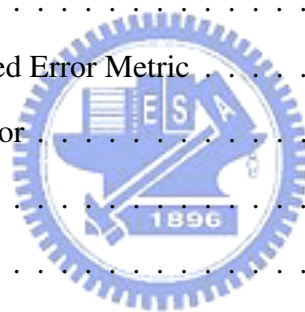
*This dissertation is dedicated to my wife Mei-Ying and  
my child who will be born around Dec. 2006.*



# Contents

<b>Acknowledgements</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Main Contributions . . . . .	3
1.2 Dissertation Organization . . . . .	4
<b>2 Related Work</b>	<b>5</b>
2.1 Real-Time Rendering . . . . .	5
2.2 Mesh Simplification . . . . .	6
2.2.1 Level-of-detail modeling . . . . .	6
2.2.2 Progressive meshes . . . . .	7
2.2.3 Error metric . . . . .	10
2.2.4 Incremental vs. total error metric . . . . .	14
2.2.5 Vertex-to-plane vs. surface-to-surface error . . . . .	15
2.2.6 Texture distortion . . . . .	15
2.3 Visibility Culling . . . . .	17
2.4 Hybrid Rendering . . . . .	17
2.4.1 Region-based sprite . . . . .	18
2.4.2 Environment-based mesh . . . . .	18
2.4.3 Object-based Textured LOD mesh . . . . .	19
<b>3 Texture Adaptation for Progressive Meshes</b>	<b>20</b>

3.1	Introduction . . . . .	20
3.2	Texture Adaptation . . . . .	23
3.2.1	Overview . . . . .	23
3.2.2	Cell correspondence between two consecutive levels . . . . .	24
3.2.3	Texture adaptation . . . . .	27
3.2.4	Indexing map . . . . .	27
3.3	Experimental Results . . . . .	30
3.3.1	Preprocessing time . . . . .	32
3.3.2	Reversibility . . . . .	32
3.3.3	Multiple charts . . . . .	37
<b>4</b>	<b>A New Mapping-Based Error Metric</b>	<b>38</b>
4.1	Introduction . . . . .	38
4.2	A New Mapping-Based Error Metric . . . . .	39
4.2.1	Maximum error . . . . .	40
4.2.2	Average error . . . . .	41
4.3	Experimental Results . . . . .	42
<b>5</b>	<b>Hybrid Rendering Based on Viewcell Dependent Textured LOD</b>	<b>49</b>
5.1	Introduction . . . . .	49
5.1.1	View-cell dependent Textured LOD Modeling . . . . .	50
5.1.2	System overview . . . . .	51
5.2	Hybrid Rendering Scheme . . . . .	52
5.2.1	Preprocessing phase . . . . .	54
5.2.2	Run-time phase . . . . .	64
5.3	Experimental Results . . . . .	70
5.3.1	Setup . . . . .	70
5.3.2	Image quality measurement . . . . .	72
5.3.3	Mesh simplification . . . . .	73
5.3.4	Run-time performance . . . . .	75



5.3.5 Discussions . . . . .	79
<b>6 Conclusion</b>	<b>84</b>
<b>Bibliography</b>	<b>86</b>



# List of Tables

3.1	Speedup by packing texture adaptations. . . . .	31
3.2	The entire run-time cost of simplifying PM w/ and w/o texture adaptation. . . . .	31
3.3	The preprocessing time for constructing the entire PM sequence of the test models. . . . .	34
4.1	The preprocessing time for constructing the entire PM sequence w/o texture adaptation under different error metrics. . . . .	47
4.2	The preprocessing time for constructing the entire PM sequence w/ texture adaptation under different error metrics. . . . .	48
5.1	Maximum ratio of side faces seen from a point inside the cell under different FOVs. . . . .	55
5.2	Object and scene statistics. . . . .	71
5.3	Simplification performance under different self-occluding-error tolerance $T_s$ . . . . .	73
5.4	Simplification performance under different projected edge-length tolerance $T_l$ . . . . .	75
5.5	4M scene under different cell sizes 50 and 100. . . . .	75
5.6	Performance of the three configurations on a 8M-scene. . . . .	76
5.7	Performance of configuration <b>C</b> under different scene complexities. . . . .	76
5.8	Preprocessing time for different scene complexities. . . . .	79
5.9	Storage and loading time under different scene complexities ( $T_s = 5, T_l = 4.5$ ). . . . .	82

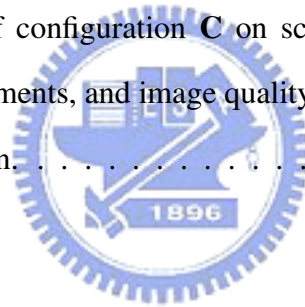
# List of Figures

2.1	Illustration of the full edge collapse $\overline{UV} \xrightarrow{ecol} \mathbf{V}'$ . . . . .	8
2.2	Half-edge collapsing of $\overline{UV} \xrightarrow{ecol} \mathbf{V}$ . . . . .	8
2.3	Half-edge collapsing of $\overline{UV} \xrightarrow{ecol} \mathbf{U}$ . . . . .	9
2.4	The accuracy vs. mesh size plot results from a mesh simplification [35]. . . . .	10
2.5	Incremental error is propagated onto per-face bounding box for the total error evaluation [14, 15]. . . . .	14
3.1	Texture distortion introduced by geometry simplification. . . . .	21
3.2	Texture distortion introduced by edge collapses. . . . .	22
3.3	Texture mapping progressive meshes with texture adaptation. . . . .	22
3.4	Mesh parameterizations before and after an edge collapse. . . . .	25
3.5	The nearest pair of points $\mathbf{A}_i$ and $\mathbf{A}_{i-1}$ of two edges, one edge $\overline{UN}$ is from level $i$ and another is from level $i-1$ . . . . .	26
3.6	Edges overlay in the one-ring neighborhood of $u$ (left), and the edges partition the neighborhood into 9 cells (right). . . . .	26
3.7	Cell correspondence between partitions of $T^i$ (left) and $T^{i-1}$ (right). . . . .	27
3.8	Accelerate texture adaptation by triangle-fan and triangle strip setups. . . . .	28
3.9	Blurred artifacts introduced by texture adaptation (left) and minimized by <i>indexing mapping</i> (right). . . . .	28
3.10	Each texel on indexing map is the texture coordinate referring to the texel on original map. . . . .	29
3.11	Indexing mapping. . . . .	29

3.12	Use one single indexing map to access to all the original maps. . . . .	30
3.13	Performance difference of indexing map in 8-bit (center) and 16-bit (left) precision. . . . .	30
3.14	Textured images of parasaur head model. Top row of (b),(c),(d): simplified by QEM of 5D without texture adaptation, middle row: by APS without texture adaptation, bottom row: by QEM of 5D with texture adaptation. . . . .	33
3.15	Normal mapped parasaur head. left: 7685 polygons, center: 499 polygons, and right: 499 polygons with texture adaptation. . . . .	34
3.16	Textured images of horse model. Left: original model of 8160 polygons with parameterized texture map, center: simplified model of 800 polygons without texture adaptation, right: simplified model of 800 polygons with texture adaptation. . . . .	34
3.17	Left: original model of 17483 polygons, center: simplified model of 500 polygons without texture adaptation, right: simplified model of 500 polygons with texture adaptation. . . . .	35
3.18	From left to right, mesh with swirled texture coordinates and its color texture map, original mesh (7688 polygons), simplified mesh (399 polygons), and the same simplified mesh with texture adaptation. . . . .	35
3.19	(a) The top image is the original mesh, and the bottom image is the mesh that has been applied three iterations of both fine-to-coarse and coarse-to-fine transformation with the texture adaptation. (b) The zoomed images, on the left is the original mesh and on the right is the resulting mesh. . . . .	36
4.1	The correspondence of $\mathbf{x}_i$ and $\mathbf{x}_{i-1}$ is established via the cell correspondence. . . . .	40
4.2	$\mathbf{x}_n$ is adapted to $\mathbf{x}_i$ by a sequence of texture adaptations $I^n \rightarrow I^{n-1} \rightarrow \dots \rightarrow I^i$ . . . . .	42
4.3	The geometric error of simplified models without texture adaptation (measured by <i>Metro</i> ). . . . .	44
4.4	The simplified parasaur head model without texture adaptation (polygon count: 7685 $\rightarrow$ 500) by QEM, APS and the new error metric. . . . .	44

4.5	The simplified bunny head model without texture adaptation (polygon count: 17483 → 500) by QEM, APS and the new error metric. . . . .	45
4.6	The simplified zebra model without texture adaptation (polygon count: 8160 → 800) by QEM, APS and the new error metric. . . . .	45
4.7	The geometric error of the simplified models with texture adaptation (measured by <i>Metro</i> ). . . . .	46
4.8	The error distribution (in percentage) of the simplified parasaur head with texture adaptation, QEM: dark blue, APS: pink, and the new error metric: yellow. . . . .	47
4.9	The simplified parasaur head model with texture adaptation (polygon count: 7685 → 500) by QEM, APS and the new error metric. . . . .	47
4.10	The simplified bunny head model with texture adaptation (polygon count: 17483 → 500) by QEM, APS and the new error metric. . . . .	48
4.11	The simplified zebra model with texture adaptation (polygon count: 8160 → 800) by QEM, APS and the new error metric. . . . .	48
5.1	Preprocessing. . . . .	54
5.2	The maximum self-occluding error occurs at the position $V'$ . . . . .	56
5.3	The derivations of SVMesh (a) and MVMesh (b). . . . .	58
5.4	(a) is the original mesh (65,491 polygons) of a bunny viewed at one cell away (cell size 50), and (b-d) are SVMeshes for the bunny at 7 (259 polygons), 8 (254), and 9 (239) cells away. The upper-right bunnies are the projected images. . . . .	59
5.5	Testing depth variation. . . . .	59
5.6	(a) is the original mesh (65,491 polygons) of a bunny viewed at one cell away (cell size 50), (b-g) are MVMeshes of the bunny at 1 (1,605 polygons), 2 (945), 3 (554), 4 (392), 5 (330), 6 (306) cells away. The upper-right indicates actual projected images. . . . .	62
5.7	Regional back-face culling. . . . .	63
5.8	Repeat clustering. . . . .	64
5.9	Run-time phase. . . . .	65

5.10	Re-projection from source to destination image, $T_1$ and $T_2$ are the camera matrix of source image $I_1$ and destination image $I_2$ , respectively. . . . .	68
5.11	The cached image is a part of a source image. . . . .	68
5.12	The smaller angle is, the more accurate result is. . . . .	70
5.13	Popping effects occur during the transition between view cells. . . . .	70
5.14	Bird's eye view of the 8M-scene. . . . .	72
5.15	Distribution of SVMesh and MVMesh for the scenes 4M-50- $T_s$ -4.5. . . . .	74
5.16	MVMeshes of bunny for different $T_l$ . . . . .	74
5.17	Rendered images by configuration <b>B</b> and <b>C</b> . . . . .	77
5.18	Rendered images by configuration <b>B</b> and <b>C</b> at another view. . . . .	78
5.19	Run-time statistics of configuration <b>C</b> on scene 8M-50-5-4.5: The frame rates with prefetching under a cold cache and a warm cache, and without prefetching. . . . .	80
5.20	Run-time statistics of configuration <b>C</b> on scene 8M-50-5-4.5: The polygon count, texture requirements, and image quality. . . . .	81
5.21	Cases of cell transition. . . . .	83





## Abstract

Real time rendering has been a kernel technology for interactive applications such as game, virtual reality (VR), and visual simulation. Real time rendering technologies can be geometry-based, image-based, or hybrid. In this thesis, we first present two novel techniques in geometry-based rendering and then propose a hybrid rendering framework. Among several key technologies in geometry-based real time rendering, level-of-detail (LOD) modeling has been a vital representation and a very popular technique in real-time applications. We develop a novel, simple, and effective texture adaptation scheme to eliminate the texture distortion commonly observed in mapping textures to progressive meshes. Based on this scheme, we propose a new surface-to-surface error metric, aiming to offer a simplification error measurement that is competitive to or even better than the existing methods. Hybrid rendering has been proven to be a very useful supplement to geometry-based technologies used in game and VR. We present a hybrid rendering scheme that explores the locality of visibility at the cost of extra storage and prefetching, and makes a tradeoff between image quality and rendering efficiency by using textured LOD meshes.

To support texture mapping progressive meshes (PM), we usually allow the whole PM sequence to share a common texture map. Such a common texture map can be derived by using appropriate mesh parameterizations that may consider the minimization of geometry stretch, texture stretch, or even the texture deviation introduced by edge collapses. We have found that even with a well parameterized texture map, the texture mapped PM still reveals apparent texture distortion due to geometry changes and the nature of linear interpolation used by texture mapping hardware. In this dissertation, we propose a novel, simple, and efficient approach that adapts texture content for each edge collapse, aiming to eliminate texture distortion. A texture adaptation and its reverse operation are local and incremental operations that can be fully supported by graphics hardware. We also propose the mechanism of indexing mapping to reduce blurred artifacts due to the under-sampling that might be introduced by the texture adaptation.

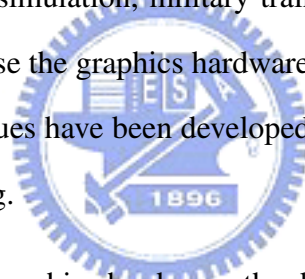
Experimental results have demonstrated that texture distortion is almost eliminated in tested examples. Based on the texture adaptation scheme, we propose a new mapping-based error metric that is able to provide much more accurate measurement of simplification error than APS or even QEM in the presence of badly parameterized texture maps. From the experimental results, we have observed that the proposed error metric outperforms the well-known error metric APS, and is better than or almost similar to QEM, depending on either maximum, mean, or RMS error used in measuring the approximation error.

In the proposed hybrid rendering framework, the object space is first subdivided into cells. For each cell, inside objects are rendered as normal while outside objects are rendered as textured LOD meshes using projective texture mapping. The textured LOD meshes is object based and derived from the original mesh based on the captured depth images viewed at the centers of the cell and its adjacent cells. With this textured LOD mesh, problems commonly found in hybrid rendering, such as hole problems due to occlusion among objects and the gap problems due to resolution mismatch, can be avoided. Moreover, the size of holes due to self-occlusion is constrained to be within a user-specified tolerance. Several scenes with millions of polygons have been tested and higher than 600 FPS has been achieved with a little loss of image quality.

# Chapter 1

## Introduction

For last decades, computer graphics has become more widely used in many applications, including visualization, surgical simulation, military training, virtual shopping, movie creation, and 3D games. Not only because the graphics hardware becomes more and more powerful and cheaper, but also many techniques have been developed to eliminate the difficulty in achieving realistic and real-time rendering.



Despite tremendous strides in graphics hardware, the demand of realism seems to grow faster than the advance of graphics hardware. To achieve realism, very complex 3D scenes may be necessary. Such complex scenes, however, may be too complex to be rendered in realtime with any available graphics hardware to date. On the other hand, from a laser scans of real objects to procedurally generated geometries, the input models can be too complex to be rendered in realtime. In fact, many polygons in a model are redundant since these polygons are projected to a portion of a screen pixel and do not help much in improving rendering quality. One possible answer to above question is the level-of-detail modeling, known as LOD in short, which aims to provide satisfactory rendering quality with less number of polygons. Although LOD modeling has been studied for about 30 years (the most remarkable paper can be tracked back to the work of James Clark [10]), LOD modeling still is an important research topic in computer graphics.

Texture mapping is a simple and efficient method for recovering surface details of a simplified

model. Applying texture maps onto simplified meshes, however, often encounters some serious texture distortion. Many approaches have been proposed to minimize such a distortion, but all fail to solve it successfully. In this dissertation, we address the texture distortion problem and investigate what are the sources of the texture distortion, and propose a novel and simple texture adaptation scheme, to eliminate the texture distortion.

In LOD modeling, meshes are simplified by a sequence of simplification operations. Usually, the sequence is ordered by the cost of the simplification operation, which measures the error between the meshes before and after simplification. The use of an accurate error metric can in general lead to a simplification that performs better in preserving geometric features and shape. While often producing satisfactory simplification results, the quadric error metric (QEM) proposed by Garland and Heckbert [29] is not geometrically accurate. On the other hand, the appearance preserving simplification (APS) proposed by Cohen et al. [15] is more geometrically accurate, but its accuracy often suffers from the incorrect mapping introduced by badly parameterized texture maps. A more accurate error metric that is efficient and performs better than current well known error metrics is desirable. Based on the texture adaptation scheme, we introduce a new mapping-based error metric which is a variant of APS but much more accurate than APS.

Although LOD modeling techniques reduce the complexity of polygonal meshes, it may still not be able to render an extreme complex scene with acceptable quality at a real-time frame rate. Other approaches, including visibility culling and image-based rendering, can be integrated to further speed up the rendering. While some important visual features such as silhouettes can only be known at run-time and better preserved by view-dependent LOD techniques, a pre-computed view-cell dependent LOD might be a compromise between the view independent and view dependent LOD. We develop a view-cell dependent textured LOD modeling for all objects outside a view cell. The construction of the viewcell dependent textured LOD modeling takes into account hybrid rendering, silhouette preserving, back-face culling, and even occlusion culling. We introduce a hybrid rendering system that is based on view-cell dependent

textured LOD modeling and space subdivision, and is able to exploit the spatial and temporary coherence. The proposed system successfully is capable of rendering complex scenes in realtime with only little image-quality loss.

## 1.1 Main Contributions

The main contributions of this dissertation research are the following:

- Propose a novel, simple, and effective texture adaptation scheme to eliminate the texture distortion often observed in texture mapped simplified meshes. It provides a very useful and automatic tool for model artists to design models and textures without concerning the texture distortion in mapping textures to simplified meshes. Moreover, the proposed indexing map successfully reduces the blurred artifacts introduced by the texture adaptation.
- Propose a new mapping-based error metric that is able to accurately measure the simplification error even in the presence of badly parameterized texture maps. The mapping of two points on two consecutive levels of mesh is established via the operation of texture adaptation. Such a mapping can be easily extended to the mapping between the original mesh and the simplified mesh by using the indexing map. Experiments have shown that the proposed total/incremental maximum error metric performs better than APS, and better than or almost similar to QEM, depending on how the approximation error is measured.
- Propose a hybrid rendering scheme that successfully adapts advantages of both image- and geometry-based rendering techniques. It is able to render extremely complex scenes at a real-time frame rate while preserving silhouettes and surface details of objects. An object-based textured LOD mesh representation is proposed to represent closer objects as well as distant scene in a unified representation. Compared to the environment-based mesh approach that is designed for distant scene. our approach has less popping effects

introduced in the transition between different representations. The proposed representation not only successfully removes the folding problems, gap problems, and rubber-sheet artifacts, but also limits the hole problem due to self occlusion. Moreover, the proposed representation together with scene subdivision is easily integrated with visibility culling techniques to exploit spatial coherence.

## 1.2 Dissertation Organization

We give an overview and related works of LOD modeling and hybrid rendering in Chapter 2. The texture distortion and texture adaptation are described in detail in Chapter 3. In Chapter 4 a new mapping-based error metric for mesh simplification is presented and comparison to some well-known error metrics is discussed. The proposed hybrid rendering scheme is described in Chapter 5. Finally, a conclusion and possible future work are stated in Chapter 6.



# Chapter 2

## Related Work

### 2.1 Real-Time Rendering

There are extensive researches in the field of real-time rendering, ranging from LOD modeling, visibility culling, image-based rendering, and hybrid rendering.

LOD modeling simplifies geometric meshes of unnecessary complexity into appropriate simplified meshes that are suitable for some viewing conditions for the efficiency of rendering and processing. However, it is difficult to attain visual fidelity in general.

Visibility culling techniques cull out polygons that are invisible from the current view before entering the graphics rendering pipeline. The task includes back-face culling that culls out back-facing polygons, view-frustum culling that culls out polygons or objects outside the view frustum, and occluding culling that culls out polygons occluded by other objects.

The image-based rendering approaches use the current view image under some constraint to generate new view image when a viewer changes his or her viewpoint. It has the advantages of that its rendering cost is independent on the scene complexity, and of that its potential to produce photo-realistic rendering images. However, it suffers from the problem of limited viewing degree of freedom, which is trivial to geometry-based rendering. On the other hand, LOD

modeling alone often reveals the problem of losing surface details. Thus, several hybrid rendering schemes have been proposed to combine traditional geometric and image-based rendering techniques to retain advantages of both rendering scheme.

## 2.2 Mesh Simplification

The geometric meshes created by geometry modeling, e.g. CAD design tools, and laser scanning systems [17] are rarely optimized for rendering efficiency. The input models may be too complex to be rendered efficiently and may be in unnecessarily high detail for some viewing conditions. These complex models require simplification process to reduce complexity for efficient rendering or further processing. Mesh simplification already has been studied intensively in the last decade. Its ultimate goal is to reduce the complexity of a geometric mesh at the cost of undetectable visual difference, and to provide a tradeoff between fidelity (image quality) and speed (rendering efficiency).

Luebke et al. [43] gave a complete and intensive review of model simplification. Additional comprehensive surveys on LOD modeling can also be found in [11, 12, 28, 34, 45]. Here, we review the works that address the reduction of texture distortion resulting from model simplification and address issues of some popular error metrics.

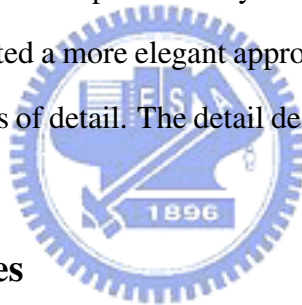
### 2.2.1 Level-of-detail modeling

The technique that generates multi-resolution representations of a model and select a representation of appropriate resolution for a particular viewing condition is called level-of-detail modeling. For example, a detailed mesh is used when the viewer is close to the object, and a coarser approximation is substituted when the viewer is distant from the object. The multi-resolution representation can be derived in preprocessing or at run time. The former is termed view independent LOD modeling and the latter is called view-dependent LOD modeling. View-independent LOD modeling usually generates a set of meshes of different levels, each of which



has resolution reduced by a factor of two from the mesh of upper level. In total, the LOD modeling for a model requires only twice as much storage space as the model.

For view-independent LOD modeling, instantaneous switching between two successive levels of detail may lead to perceptible popping effects. Popping effects can in general be minimized by blending or morphing. The blending employs alpha-blending to fade between the two successive levels of detail. Both of two successive representations are rendered, and the rendered images are alpha-blended according to the distance to the viewing position [46]. Such blending requires extra cost on rendering two successive representations of the same object, and may still have some blending artifacts if two representations differ at the silhouettes. On the other hand, the morphing gradually changes the shape between two successive levels of detail. It requires the knowledge of the vertex correspondence between two successive representations. Turk [56] proposed a method to have such correspondence by mutual tessellating the two successive levels of detail. Hoppe [35] presented a more elegant approach referred as *geomorph* that has more smooth transition between levels of detail. The detail description is given in the next subsection.



### 2.2.2 Progressive meshes

Hoppe [35] proposed a *continuous* level-of-detail representation, called *progressive mesh* or PM in short. The term of “*continuous*” emphasizes the ability to have smooth transition between two successive levels. By limiting the difference between two successive levels to a local region, it is easy to establish the vertex correspondence required for morphing. While discrete LOD modeling can provide only discrete approximation, PM is capable of providing a continuous representation that has less popping effects.

PM simplifies an initial mesh  $M = M^n$  to a coarse mesh  $M^0$  by applying a series of  $n$  edge collapses:

$$M = M^n \xrightarrow{ecol_{n-1}} \dots \xrightarrow{ecol_1} M^1 \xrightarrow{ecol_0} M^0 .$$

As shown in Figure 2.1, an edge-collapse operation collapses two adjacent vertices  $U$  and  $V$

into a single vertex  $\mathbf{V}'$ . Vertices  $\mathbf{U}$  and  $\mathbf{V}$ , edge  $\overline{\mathbf{UV}}$ , and adjacent faces  $\triangle\mathbf{UVL}$  and  $\triangle\mathbf{VUR}$  vanish after the collapse. When the new vertex  $\mathbf{V}'$  differs from  $\mathbf{U}$  and  $\mathbf{V}$ , this operation, denoted

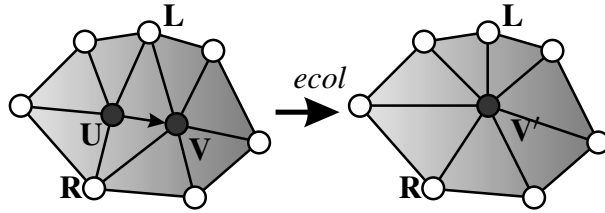


Figure 2.1: Illustration of the full edge collapse  $\overline{\mathbf{UV}} \xrightarrow{ecol} \mathbf{V}'$ .

as  $\overline{\mathbf{UV}} \xrightarrow{ecol} \mathbf{V}'$ , is referred to as a full-edge collapse. Contrarily, a half-edge collapse operation collapses edge  $\overline{\mathbf{UV}}$  to either vertex  $\mathbf{V}$  or vertex  $\mathbf{U}$ , and is denoted as

$$\overline{\mathbf{UV}} \xrightarrow{ecol} \mathbf{V}$$

or

$$\overline{\mathbf{UV}} \xrightarrow{ecol} \mathbf{U},$$

respectively. The half edge collapse operation collapses  $\mathbf{U}$  to  $\mathbf{V}$ , and also removes one vertex, three edges, and two faces from a mesh (see Fig. 2.2). Although full-edge collapse can po-

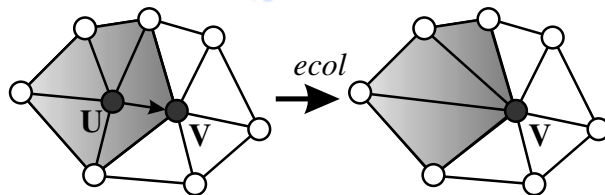


Figure 2.2: Half-edge collapsing of  $\overline{\mathbf{UV}} \xrightarrow{ecol} \mathbf{V}$ .

tentially produce a better approximation of the original model under the same polygon count, it needs to modify the vertex buffer and obstructs the vertex caching for hardware acceleration [48]. As a consequence, many descendent works use the half-edge collapse in practice because its simplicity in implementation and its efficiency in rendering.

The inverse operation of an edge collapse is the vertex split that splits a vertex into two vertices and forms a new edge. By applying a series of  $n$  vertex splits, the coarse mesh  $M^0$  can be refined

to the initial mesh  $M$ :

$$M^0 \xrightarrow{vsplit_0} M^1 \xrightarrow{vsplit_1} \dots \xrightarrow{vsplit_{n-1}} M^n = M .$$

Under a network environment, the latency can be greatly reduced by transmitting the coarse version of a mesh at the very beginning and then progressively transmitting the remaining details. One can see a progressively refined approximations to the model as data is being incrementally received.

Before introducing the error metrics that evaluate the error introduced by an edge-collapse operation and guide the mesh simplification process, we briefly describe the conventional framework for mesh simplification.

To construct a PM for a polygonal mesh, the mesh is gradually simplified by a sequence of edge collapses in an order of increasing cost, evaluated by a specific error metric. This is supported by constructing a priority queue in an increasing cost for all the edges of the mesh and then the edge with the lowest cost will be popped out from the priority queue for each edge collapse operation. Since the cost of edge collapsing is directional, collapsing  $U$  to  $V$  will have different cost from collapsing  $V$  to  $U$ ; see the difference between Figure 2.2 and Figure 2.3. Moreover, since a vertex  $U$  can only be collapsed to one of its one-ring neighboring vertices, it is not necessary to put all of its one-ring neighboring edges into the priority queue. Instead, we put only the directional edge that has the smallest cost among the neighboring edges into the priority queue. In practice, an infinite cost is usually assigned to a candidate edge  $\overline{UV} \xrightarrow{ecol} V$  that violates some constraints such as face flip and parametric flip.

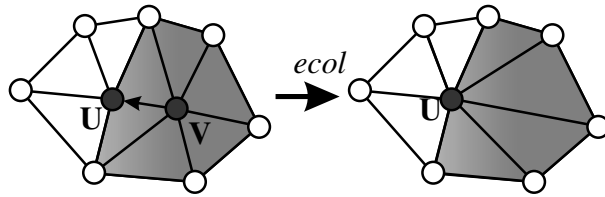


Figure 2.3: Half-edge collapsing of  $\overline{UV} \xrightarrow{ecol} U$ .

For each edge collapse operation, the edge  $\overline{UV}$  with the lowest cost (error) is popped out from

the priority queue and collapsed to  $\mathbf{V}$ . After collapsing edge  $\overline{\mathbf{UV}}$  to  $\mathbf{V}$ , the cost of  $\mathbf{V}$ 's one-ring neighbors should be re-evaluated and the priority queue should be updated accordingly. For efficiency consideration, one can postpone the evaluation until necessary. It is called lazy evaluation, proposed in [13], in which the cost of a directional edge in the priority queue is not re-evaluated until it is popped from the priority queue with a TRUE lazy flag set at the time one of its neighbors was collapsed. The lazy evaluation scheme has been reported to reduce the number of edge cost evaluation to about a factor of 3.2 [14] with only a little sacrifice in accuracy.

### 2.2.3 Error metric

The sequence of edge collapses for producing progressive meshes must be chosen carefully since it determines the quality of the approximating mesh  $M^i$ , for  $0 \leq i < n$ . Figure 2.4 shows the plot for the accuracy of simplified meshes in different polygon counts. A good simplification algorithm is expected to produce meshes that have a curve close to the optimal curve in dashed line.

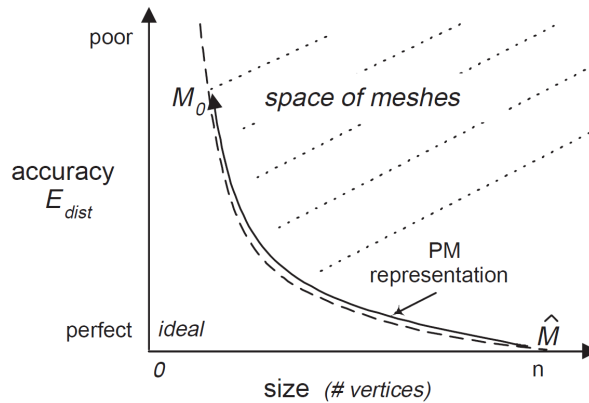


Figure 2.4: The accuracy vs. mesh size plot results from a mesh simplification [35].

However, it has been proven that finding the optimal edge collapse sequence is NP-hard for both convex polytopes [20] and polyhedral terrains [1]. Many greedy approaches have been proposed to approximate the optimal solution as much as possible by using priority queue strategy and a good error metric to guide the mesh simplification.

The simplest error metric is the vertex-to-plane distance. If the equation of a given plane  $p$  is represented by

$$p = \begin{bmatrix} a & b & c & d \end{bmatrix}^T$$

and the position of vertex  $\mathbf{V}$  is

$$\mathbf{V} = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T,$$

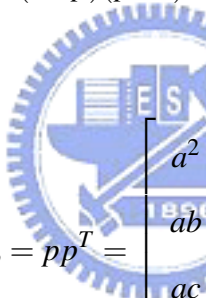
then the vertex-to-plane distance is

$$\mathbf{V}^T p = ax + by + cz + d.$$

The squared vertex-to-plane distance is then derived as

$$(\mathbf{V}^T p)^2 = (\mathbf{V}^T p)(p^T \mathbf{V}) = \mathbf{V}^T (pp^T) \mathbf{V} = \mathbf{V}^T K_p \mathbf{V},$$

where



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

The sum of squared distance from vertex  $\mathbf{V}$  to a set of planes  $P$  is then

$$\sum_{p \in P} \mathbf{V}^T K_p \mathbf{V} = \mathbf{V}^T \left( \sum_{p \in P} K_p \right) \mathbf{V}.$$

Garland and Heckbert [29] introduced the quadric error metric (QEM) for measuring vertex-to-plane distance and provided a scheme to compute the total vertex-to-plane error. For a vertex  $\mathbf{U}$ , we store the matrix  $Q_{\mathbf{U}}$  that is the sum of  $K_p$  over all one-ring neighboring planes  $P$  of  $\mathbf{U}$ , i.e.,

$$Q_{\mathbf{U}} = \sum_{p \in P} K_p,$$

where  $P$  is the set of planes neighboring to  $\mathbf{U}$ . The vertex-to-plane error for a vertex  $\mathbf{V}$  to the

set of planes  $P$  is simply defined as

$$\mathbf{V}^T Q_U \mathbf{V}.$$

When applying a full-edge collapse  $\overline{UV} \xrightarrow{ecol} \mathbf{V}'$ , the plane sets of vertices  $\mathbf{U}$  and  $\mathbf{V}$  are propagated to the collapsed vertex  $\mathbf{V}'$  and in consequence

$$Q_{\mathbf{V}'} = Q_{\mathbf{V}} + Q_{\mathbf{U}}.$$

If the set of one-ring neighboring planes of  $\mathbf{U}$  and  $\mathbf{V}$  are disjoint, the plane set propagation is equivalent to a union. Otherwise, each overlapped plane is counted multiple times. In fact, a plane is counted at most three times since it contributes only to the vertices of its defining triangle. QEM has a problem that we get different squared distance under different triangulations. In general, densely tessellated mesh gets larger squared distance than sparsely tessellated one. This may be a problem for the mesh that has different tessellation densities from region to region.



In addition to geometric error, human eye may be more sensitive on other attributes, such as colors, normals, textures, and etc. Garland and Heckbert extended their previous work to meshes with attributes by extending the quadric error metric from 3D to higher dimensions [30]. For the case of a mesh with a texture map, the dimension of  $Q$  is five, two for additional  $s$  and  $t$  coordinates. However, the Cartesian distance in geometry space may not be equivalent to the distance in texture coordinate space.

While it is easy to compute the vertex-to-plane distance, the surface-to-surface distance is more likely to be able to reveal the true error introduced in mesh simplification. Bajaj and Schikore [3] proposed a method to evaluate surface-to-surface error. Both of the one-ring neighboring edges of  $\mathbf{V}'$  (on  $M^{i-1}$ ) and the one-ring neighboring faces of  $\overline{UV}$  (on  $M^i$ ) are mapped to a projection plane using a planar projection. The projection plane is the average plane of the one-ring neighboring faces of  $\mathbf{V}$ . To have a less distorted mapping, the proportion of the projected edge length is constrained to be preserved in the mapping. It is shown that the maximum surface-to-surface in the local region is at one of the intersections of the projected one-ring neighboring edges of

$\mathbf{V}'$  and  $\overline{\mathbf{UV}}$ . Therefore, finding the maximum surface-to-surface error amounts to finding the maximum among these intersections. The problem of this approach is that a valid projection may not always exist.

Cohen et al. [13] proposed a more robust method to find a valid projection for an edge collapse. Their descendent work [15] uses a global parameterization as the bijective (1-to-1 and onto) mapping between the one-ring neighboring faces of  $\overline{\mathbf{UV}}$  and the one-ring neighboring faces of  $\mathbf{V}'$ . It is assumed that the input is a mesh with parameterized texture maps. Since the parametrization is a bijective mapping function

$$\mathcal{F}(\mathbf{X}) = \mathbf{x},$$

where  $\mathbf{X}$  is a 3D position and  $\mathbf{x}$  is its corresponding parameter in 2D, it can be used to define point correspondence before and after collapsing an edge. Note that, an edge is in general collapsed to a new vertex and, moreover, the mapping between 2D parameter and 3D surface point will be changed after an edge collapse since such a mapping is derived by the interpolation of triangle's vertices. In consequence, the bijective mapping  $\mathcal{F}$  varies while mesh  $M^i$  is simplified to  $M^{i-1}$ ,  $i = n, \dots, 1$ . For two consecutive levels of mesh,  $M^i$  and  $M^{i-1}$ , an incremental mapping-based error metric called APS is defined as the distance between two points, one on  $M^i$  and another on  $M^{i-1}$ , that are mapped to the same parameter in 2D domain. That is, APS is

$$e_{incr}(\mathbf{x}) = \|\mathcal{F}_i^{-1}(\mathbf{x}) - \mathcal{F}_{i-1}^{-1}(\mathbf{x})\|. \quad (2.1)$$

from  $\mathcal{F}_i$  to  $\mathcal{F}_{i-1}$ . The total mapping error (distance) of a parametric point  $\mathbf{x}$  can be defined as

$$e_{total}(\mathbf{x}) = \|\mathcal{F}_n^{-1}(\mathbf{x}) - \mathcal{F}_i^{-1}(\mathbf{x})\|. \quad (2.2)$$

However, finding  $\mathcal{F}_n^{-1}(\mathbf{x})$  is not a trivial task. Cohen et al. proposed a evaluation scheme for total error. Incremental error is propagated and accumulated using the per-face bounding boxes (Figure 2.5). Although the error metric APS is geometric in natural, its projected screen distance

can be viewed as a measurement for the so called texture deviation. Such an extension implies that APS takes into account not only geometric error but also texture deviation. One problem about APS is that it relies on the mapping provided by the parameterized texture map and hence its performance usually depends on how well the parameterization is.

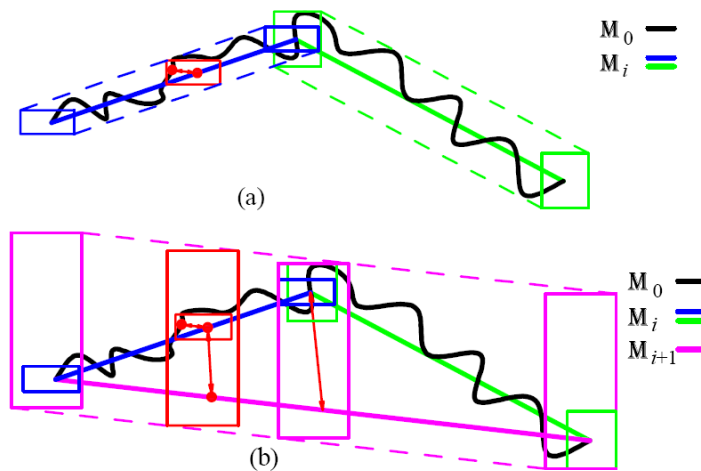


Figure 2.5: Incremental error is propagated onto per-face bounding box for the total error evaluation [14, 15].

Lindstrom and Turk [42] proposed an image-based approach to measure the error introduced in mesh simplification. The mean squared error of the difference between the images of the original model and images of the simplified model is computed for a set of views. Since images are taken into consideration, this image-driven simplification performs better than others in preserving the texture content.

## 2.2.4 Incremental vs. total error metric

Although authors of [41] claimed and demonstrated that their incremental and memory-less error metric can still output measurably good approximations, one may still need a total error metric for error bounded simplifications needed for some applications such as medical and scientific applications. Using the incremental error metric makes no guarantee on the error bound between the simplified mesh and the original mesh, and may result in serious problems in some applications.



On the other hand, the evaluation of the total error usually requires to maintain some information of the original mesh throughout the mesh simplification process and therefore is more computationally expensive than the evaluation of an incremental error.

### 2.2.5 Vertex-to-plane vs. surface-to-surface error

Again, surface-to-surface distance may reveal more truth than the vertex-to-plane distance for mesh simplification. In addition to mapping-based error metric, the Hausdorff distance can also be used to measure surface-to-surface distance. The Hausdorff distance is a distance between two point sets, said  $A$  and  $B$ , and is defined as

$$H(A, B) = \max(h(A, B), h(B, A)),$$

where

$$h(A, B) = \max_{a \in A} \min_{b \in B} \|a - b\|.$$

Note that,  $h(A, B) \neq h(B, A)$ . Since surface is a type of continuous point set, the Hausdorff distance is adopted by many error measuring tools to measure the surface-to-surface distance. For example, *Metro* [9] measures the Hausdorff distance between two surfaces by using sampling.

### 2.2.6 Texture distortion

Using texture maps is the most practical and the simplest way to recover surface details for simplified meshes. Moreover, texture mapping has a direct support from graphics hardware that lets it be a standard in real time rendering applications. However, we often observe texture distortion in mapping textures to simplified meshes. Following is a brief review of the approaches that have been proposed to minimize such texture distortion.

To associated attributes with vertices of a mesh surface requires a bijective mapping that is usually derived by the parameterization [26, 27, 37, 47, 48].

Cignoni et al. [8] presented a method to preserve surface attributes in texture maps for each level of detail. Requiring one texture map dedicated for each level of detail, however, implies the need for large storage space and long processing time. APS [15] uses a texture deviation error metric to measure the distance between two points on two consecutive levels and correspond to the same texture value. Sander et al. [48] proposed a mesh parameterization that takes into account the geometry and texture stretch as well as texture deviation introduced by edge collapses. The texture map derived from such a parameterization is then shared by the entire PM sequence. Kim and Wohn [38] proposed a method to minimize texture distortion by generating a texture map that can be mapped to all levels of detail in a pre-processing stage. A distortion metric is then used to guide the mesh simplification. Sander et al. [47] proposed a signal-specialized parameterization to minimize texture stretch of the parameterization by allocating more texture samples to areas of higher signal frequency. Khodakovsky et al. [37] proposed a globally smooth parameterization with low distortion. Xu et al. [57] proposed a texture information driven simplification in which texture image frequency distribution and texture mapping distortion energy are combined to evaluate the simplification error.

Several papers have been proposed to minimize texture distortion problem in a coarse-to-fine process. Eckstein et al. [25] proposed a coarse-to-fine optimization to generate a proper texture coordinate for the newly refined vertex to support texture mapping multi-resolution meshes. It always guarantees a solution by adding Steiner vertices. Zhou et al. [61] proposed a technique called *TextureMontage* to seamlessly map multiple texture images onto an arbitrary 3D model. One part of their work is to derive texture coordinates through optimization for all the vertices of the original mesh from the coarse texture coordinate assignments on the base mesh. Both texture stretch and texture color continuity are taken into account in this process.

However, all the existing approaches only minimize the texture distortion, they do not solve the problem from the sources. In addition to the geometric error introduced in mesh simplification, we found that the nature of the linear interpolation in hardware texture mapping is the major source of texture distortion. We will address the problem in depth in Chapter 3.

## 2.3 Visibility Culling

For very complex scenes, visibility culling is another commonly used technique to further reduce the number of polygons sent to the rendering pipeline. Visibility culling, as we have been doing in the traditional rendering pipeline, falls into two categories. The first is the back face culling, aiming to perform the culling using the polygon clusters or its hierarchy. Kumar et al. proposed a sub-linear algorithm for hierarchical back-facing culling [39]. Zhang et al. improved the sub-linear algorithm by introducing *normal mask* which reduces the per polygon back-face test to only one logical AND operation [58].

The second category is the occlusion culling, aiming to cull away the polygons occluded by other objects. Occlusion culling can be done in processing or at run time. Shadow frusta [36], hierarchical Z-buffer [33], and hierarchical occlusion map [59] are popular run-time occlusion culling techniques. To avoid the inevitable run-time overhead, several region-based conservative occlusion culling methods aiming to derive the so called potentially visible set of polygons have been proposed. Cohen-Or et al. [16] proposed a preprocessing algorithm for regional occlusion culling, but its performance depends heavily on a single strong occluder. Durand et al. [24] proposed *extended projection* operations and Schaufler et al. [49] proposed *blocker extension* to handle occluder fusion for multiple occluders.

## 2.4 Hybrid Rendering

Geometry-based rendering based on LOD modeling and visibility culling alone usually still cannot meet interactive requirement for very complex scenes. Image-based rendering (IBR) has been a well-known alternative. IBR takes parallax into account, and renders a scene realistically by interpolating neighboring reference views [5,6,44,52]. IBR is usually efficient since its computation cost is independent of the scene complexity. It is, however, often constrained by the limited viewing degree of freedom, and the problems including folding, gap, and hole. LDI [54] is a good try to eliminate hole problems due to the visibility changes. LDI structure

is more compact in the sense that redundant information has been reduced when several neighboring reference images are composed into a single LDI. However, splatting is necessary in the rendering to reduce to the gap problem. Lumigraph [32] and light field rendering [40] have been proposed to reduce the  $7D$  *plenoptic function* to a 4D function for static scenes. However, both methods require large amount of storage for the extremely large number of images.

Hybrid rendering is basically a geometry-based rendering that integrates IBR techniques, aiming to replace the rendering of geometry for certain scene regions or objects by the rendering of pre-generated or cached images. In the following subsections, we describe four types of hybrid rendering.

#### **2.4.1 Region-based sprite**

Image caching proposed in [50, 53] combines geometry-based rendering and IBR, aiming to achieve an interactive frame rate for complex static scenes. The images of subdivided regions are cached as planar sprites and are rendered in a back-to-front order using texture mapping. The cached planar sprite needs to be updated if its parallax error exceeds a user-specified tolerance. The cached texture possesses no depth and, in turns, limits its life cycle.

#### **2.4.2 Environment-based mesh**

The life cycle of the cached sprite can be longer if the so called depth mesh is used instead of images as proposed in [18, 19, 55]. The depth mesh of a region or a background is derived from the captured depth images by first triangulating the image's pixel space using depth and then re-projecting the 2D triangulation into the object space to represent the background or distant scene. Such depth meshes are then rendered via pipeline. In such approaches, folding problems and gaps resulting from the resolution changes can be eliminated; however, the hole problems due to occlusion among objects and self-occluding still remain. Moreover, disjointed objects

might be rendered as a connected object. Geometric accuracy might be a problem when re-projecting the 2D triangulation back to 3D space since depth meshes derived from the depth images are in pixel resolution. While objects near to depth meshes are normally represented as original geometry or LOD models, image popping due to the representation transition may occur.

In [23], Decoret et al. proposed multi-layered impostors to constrain visibility artifacts between objects to a given size, and a dynamic update scheme to improve the gap due to the resolution mismatch. However, it still encounters hole problems due to self occlusion, and requires a special hardware architecture for an efficient dynamic updated. In [2], an interactive massive model rendering system using geometric and image-based acceleration is proposed, in which distant objects are represented by textured depth meshes and near objects by LOD models. The method proposed in [60] integrates LOD and visibility computation and is suitable for scenes with high-depth complexity and very dynamic scene.

### **2.4.3 Object-based Textured LOD mesh**

Shade et al. [54] described a paradigm in which objects could be represented by environment map, planar sprite, sprite with depth, layered depth image (LDI), and polygonal mesh, depending on their distances to the viewer. The objects are rendered differently, but with different image qualities. In consequence, the transition between different representations may produce noticeable popping effects, especially for the transition between an image-based primitive and a geometry-based primitive.

# Chapter 3

## Texture Adaptation for Progressive Meshes

### 3.1 Introduction



Mesh simplification has been an active area of research in real-time graphics. The ultimate goal of mesh simplification is to generate a simplified mesh of low polygon count that preserves the fidelity of the original mesh. Texture mapping has been very useful in enhancing shaded images with more surface or color details. For a given mesh and its associated texture map, there are several possibilities of applying the texture map to the simplified meshes. One way is to have a texture map for each simplified mesh, which requires more artist work on texture design and more storage, especially for progressive meshes (PM). A more practical way is to have the entire PM sequence share a common texture map; however, when applying texture mapping to PMs, serious texture distortions are often observed. To reduce texture distortion, several schemes have been proposed. One is to consider texture deviation as an error metric, implying that edge collapses with higher texture deviation are more likely to be retained. This, however, cannot prevent texture distortion introduced by edge collapses that have been performed. In addition to using the metric of texture deviation, the texture map can be derived by using appropriate

parameterizations that take into account the minimization of geometry and texture stretch, as well as the texture deviation introduced by edge collapses. It is observed that even with a very well parameterized texture map, the texture mapped PM still reveals significant texture distortion due to geometry changes and the nature of linear interpolation employed by texture mapping hardware.

Let's consider the 2D case shown in Figure 3.1, where edges  $\overline{AC}$  and  $\overline{BC}$  are simplified to  $\overline{AB}$ . At the bottom,  $T$  is the texture map on which  $A$  maps to 0,  $B$  maps to 1, and  $C$  maps to 0.3. On the textured image of  $\overline{AB}$ , we see the blue color shares a portion of  $\overline{AB}$  that is much smaller than expected. Figure 3.2 shows a planar polygonal mesh  $M = M^2$  and the texture map  $T$  associated with it. The textured images of simplified meshes  $M^1$  and  $M^0$  reveal serious texture distortion, as shown in the bottom row of the figure. In this example, geometry remains the same, but the nature of linear interpolation for texture mapping affects the textured image. Essentially, the texture coordinate within a triangle is piecewise linear but is no longer linear when crossing edges of the triangle. Such texture distortion has been routinely observed for texture mapping PMs using any existing technique.

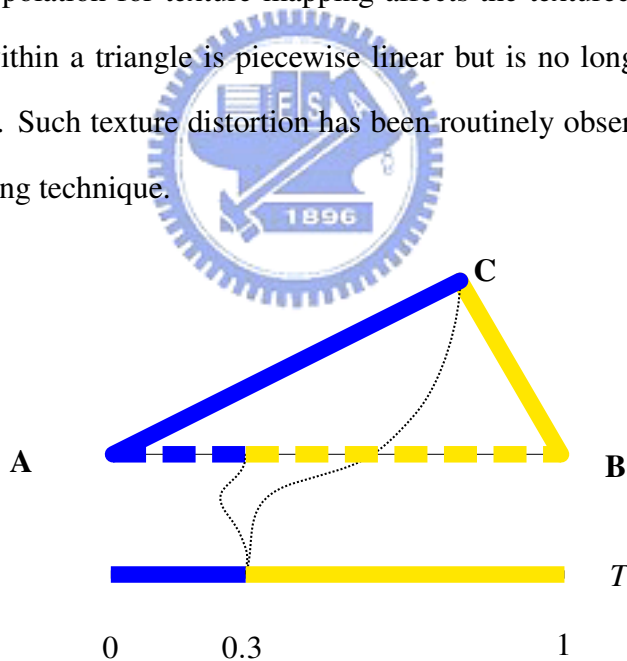


Figure 3.1: Texture distortion introduced by geometry simplification.

We present a novel, simple, and efficient approach that adapts texture content for each edge collapse, aiming to effectively eliminate the texture distortion introduced by geometry changes and the nature of linear interpolation employed by texture mapping hardware. The texture adaptation applied for each edge collapse is local to the region affected by the edge collapse and is applied to the adapted texture resulting from the previous edge collapse. The texture

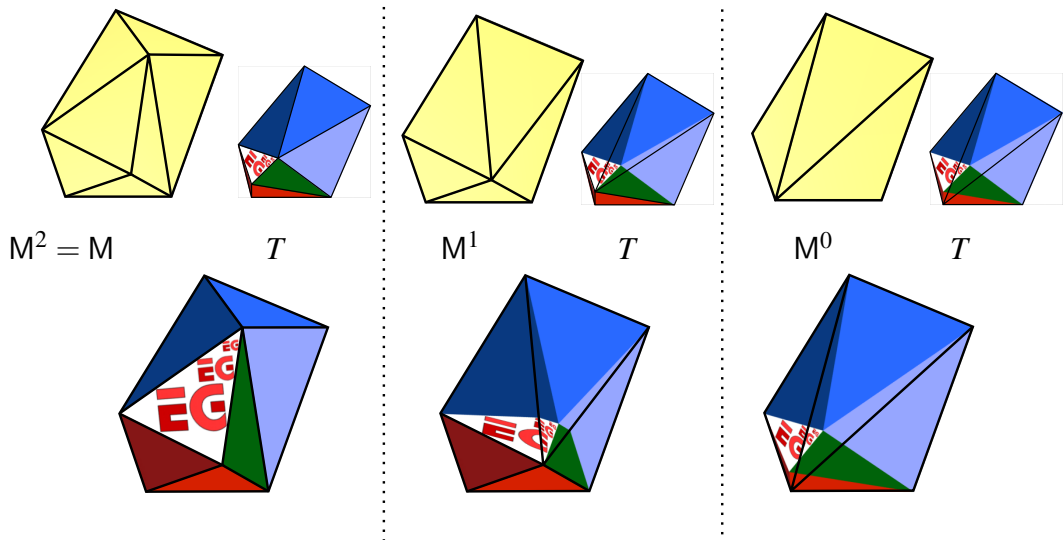


Figure 3.2: Texture distortion introduced by edge collapses.

adaptation is invertible, that is, the backward texture adaptation can be performed for a vertex split. Both texture adaptation and backward texture adaptation can be fully supported by texture mapping hardware. Once the necessary correspondence in the partition of texture space is built during the course of PM construction, the texture adaptation or its inverse can be applied on the fly before rendering the simplified or refined model with texture map. We have observed that the proposed texture adaptation is capable of eliminating texture distortion in a time that is almost negligible. Figure 3.3 depicts that the textured images of  $M^1$  and  $M^0$  with texture adaptation are indistinguishable from the textured image of the original mesh.

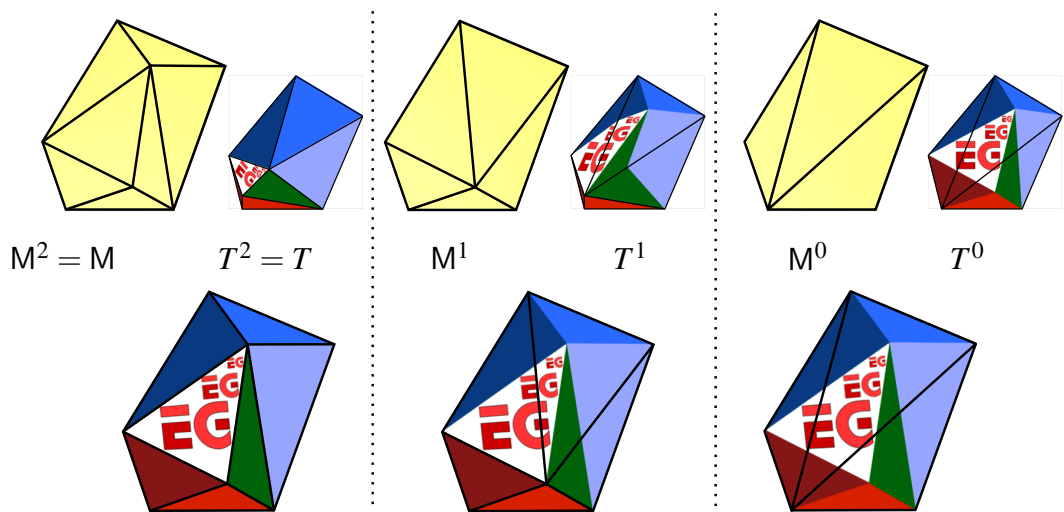


Figure 3.3: Texture mapping progressive meshes with texture adaptation.



## 3.2 Texture Adaptation

### 3.2.1 Overview

For a given polygon model  $M$  with a texture map  $T$ , the texture adaptation locally and incrementally adapts the texture map in the course of edge collapses that construct a PM of  $M$ . The goal is to eliminate the texture distortion introduced by the edge collapses. For a PM sequence, we then have a texture map  $T^i$  associated with each reduced model  $M^i$ , that is,

$$\begin{array}{ccc} M = M^n & \xrightarrow{ecol_{n-1}} \dots \xrightarrow{ecol_1} & M^1 & \xrightarrow{ecol_0} & M^0 \\ T = T^n & & T^1 & & T^0 \end{array} .$$

The texture adaptation operation from  $T^i$  to  $T^{i-1}$  is invertible, that is, for a  $T^{i-1}$  associated with  $M^{i-1}$ ,  $T^i$  of  $M^i$  can be derived by doing the inverse of the texture adaptation that brings  $T^{i-1}$  to  $T^i$ , which we call *backward texture adaptation*. Consequently, for the sequence of vertex splits from  $M^0$ , we have

$$\begin{array}{ccc} M^0 & \xrightarrow{vsplit_0} & M^1 & \xrightarrow{vsplit_1} \dots \xrightarrow{vsplit_{n-1}} & M^n = M \\ T^0 & & T^1 & & T^n = T \end{array} .$$

We will see that the texture adaptation and its inverse, backward texture adaptation, involve the same operations, which can be done very efficiently by graphics hardware. And the texture adaptation or its inverse is applied to texture map and is performed on the fly while the model is simplified or refined, respectively.

For an edge collapse  $ecol_{i-1}$  that reduces  $M^i$  to  $M^{i-1}$ , the texture adaptation is local to the region  $\mathcal{R}_{i-1}$  that is the neighborhood of the collapsed edge in texture space and is performed incrementally from  $T^i$ . Since the boundary of  $\mathcal{R}_{i-1}$  remains fixed, what the texture adaptation does is basically find an appropriate texel of  $T^i$  for each texel of  $T^{i-1}$ , all in the region  $\mathcal{R}_{i-1}$ . We will see in the next subsection that the region  $\mathcal{R}_{i-1}$  can be respectively partitioned into the same number of cells for  $T^i$  and  $T^{i-1}$ , and within each of these cells, the texture coordinates are piecewise linear. The correspondence between texels of  $T^i$  and  $T^{i-1}$  is then simplified to

the correspondence between cells of  $T^i$  and  $T^{i-1}$ . Once all pairs of corresponding cells are found, the texture adaptation can be performed by hardware texture mapping the cell of  $T^i$  to the corresponding cell in  $T^{i-1}$ . Since cell correspondence is identical for each texture adaptation and its inverse, backward texture adaptation texture maps the cell of  $T^{i-1}$  to the corresponding cell in  $T^i$ .

Essentially, texture adaptation is a re-sampling process that might pose problems of under sampling and introduce blurred artifacts. *Indexing mapping* is proposed to minimize these blurred artifacts. Indexing mapping involves an additional texture map, called the *indexing map*, that stores texture coordinates to the original texture map. During edge collapsing, texture adaptation is applied to this indexing map, leaving the original texture map alone. Only during rendering, the original texture map is mapped to the model via texture coordinates derived from the indexing map.

The cell partition and cell correspondence are derived while constructing the progressive meshes. The cell partition and cell correspondence are stored along with the information for each edge collapse and vertex split. In rendering the progressive meshes, the texture adaptation or backward texture adaptation is performed on the fly while the model is simplified or refined, respectively.

In the following sections, half-edge collapse is used in the illustration of the proposed method. Extension to the full-edge collapse is straightforward.

### 3.2.2 Cell correspondence between two consecutive levels

For a given mesh  $M$ , its texture map  $T$  is obtained by parameterizing  $M$  onto the texture plane. The parameterization of  $M$  is a one-to-one mapping  $F$  that maps each vertex  $\mathbf{V}$  of  $M$  to a point  $\mathbf{v}$  on the texture plane. For a PM using half-edge collapse, the parameterization of  $M^i$  is a subset of that for  $M$ . Figure 3.4 depicts the mapping of the neighborhood of an edge  $\overline{UV}$  on the texture plane, before and after  $\overline{UV}$  is collapsed.

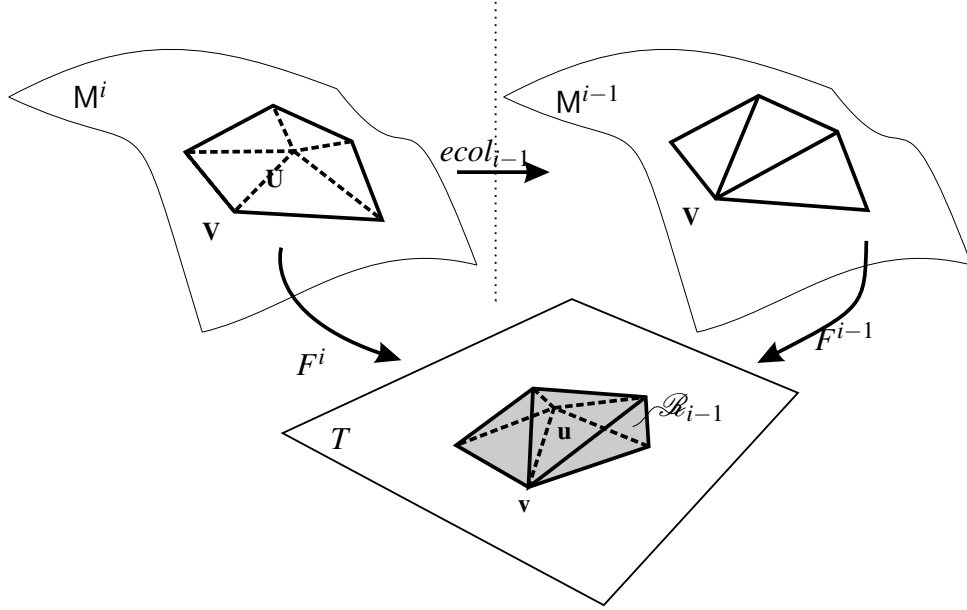


Figure 3.4: Mesh parameterizations before and after an edge collapse.

When we overlay the edges in the region  $\mathcal{R}_{i-1}$  before and after edge collapse  $ecol_{i-1}$  that reduces  $M^i$  to  $M^{i-1}$ , edges on the overlay may intersect each other and hence partition  $\mathcal{R}_{i-1}$  into cells, as shown in Figure 3.6, where  $\overline{un}$  intersects  $\overline{vw}$  at  $\mathbf{a}$  and another pair of edges intersect at  $\mathbf{b}$  and  $\mathcal{R}_{i-1}$  is partitioned into 9 cells. However, the two intersecting edges on the texture plane are generally not coplanar in 3D space. Nevertheless, for each pair of intersecting edges we can compute the pair of nearest points, one on an edge and one on another edge. Taking the mesh in Figure 3.6 as an example, we compute the pair of the nearest points  $\mathbf{A}_i$  and  $\mathbf{A}_{i-1}$ , see Figure 3.5, where  $\mathbf{A}_i$  is on  $\overline{UN}$  of  $M^i$  and  $\mathbf{A}_{i-1}$  on  $\overline{VW}$  of  $M^{i-1}$ . The points  $\mathbf{A}_i$  and  $\mathbf{A}_{i-1}$  can be derived by minimizing the distance of points on the two edges, which amounts to solving the linear system

$$\begin{bmatrix} \vec{u} \cdot \vec{u} & -\vec{v} \cdot \vec{u} \\ \vec{u} \cdot \vec{v} & -\vec{v} \cdot \vec{v} \end{bmatrix} \begin{bmatrix} \alpha_i \\ \alpha_{i-1} \end{bmatrix} = \begin{bmatrix} (\mathbf{V} - \mathbf{U}) \cdot \vec{u} \\ (\mathbf{V} - \mathbf{U}) \cdot \vec{v} \end{bmatrix},$$

where

$$\vec{u} = \mathbf{N} - \mathbf{U},$$

$$\vec{v} = \mathbf{W} - \mathbf{V}.$$

The solution of the above system is the pair of parameters  $\alpha_i$  and  $\alpha_{i-1}$ ,  $0 < \alpha_i, \alpha_{i-1} < 1$ . The texture coordinate of  $\mathbf{A}_i$ , denoted as  $\mathbf{a}_i$ , is derived by interpolating  $\mathbf{u}$  and  $\mathbf{n}$ , while the texture

coordinate of  $\mathbf{A}_{i-1}$ , denoted as  $\mathbf{a}_{i-1}$ , is derived by interpolating  $\mathbf{v}$  and  $\mathbf{w}$ , as follows:

$$\begin{aligned}\mathbf{a}_i &= \mathbf{u} + \alpha_i(\mathbf{n} - \mathbf{u}), \\ \mathbf{a}_{i-1} &= \mathbf{v} + \alpha_{i-1}(\mathbf{w} - \mathbf{v}).\end{aligned}\tag{3.1}$$

It is apparent that  $\mathbf{a}_i$  is generally not equal to  $\mathbf{a}_{i-1}$ . Note that  $\mathbf{v}_j$  represents the texture coordinate of vertex  $\mathbf{V}$  on  $M^j$ , for  $j = n, \dots, 1, 0$ , and since every vertex  $\mathbf{V}$  of  $M$  remains fixed in the course of half-edge collapsing,  $\mathbf{v}_j$  remains the same for all  $\mathbf{V}$  of  $M$ . Since  $\mathbf{U}$  is collapsed to  $\mathbf{V}$ , we need to find its texture coordinate  $\mathbf{u}_{i-1}$  on  $T^{i-1}$ . First, we find the point  $\mathbf{U}_{i-1}$  on  $M_{i-1}$  that is nearest to  $\mathbf{U}$  and identify the triangle containing  $\mathbf{U}_{i-1}$ . Then we compute the barycentric coordinates of  $\mathbf{U}_{i-1}$  with respect to the triangle and finally derive the texture coordinate  $\mathbf{u}_{i-1}$  from the texture coordinates of triangle vertices using the barycentric coordinates.

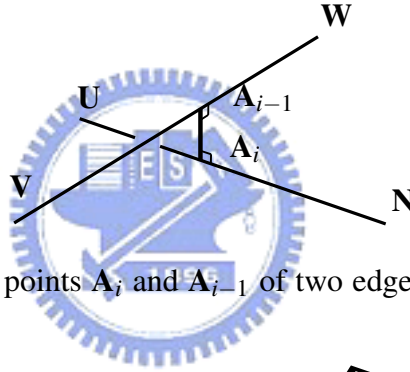


Figure 3.5: The nearest pair of points  $\mathbf{A}_i$  and  $\mathbf{A}_{i-1}$  of two edges, one edge  $\overline{\mathbf{UN}}$  is from level  $i$  and another is from level  $i-1$ .

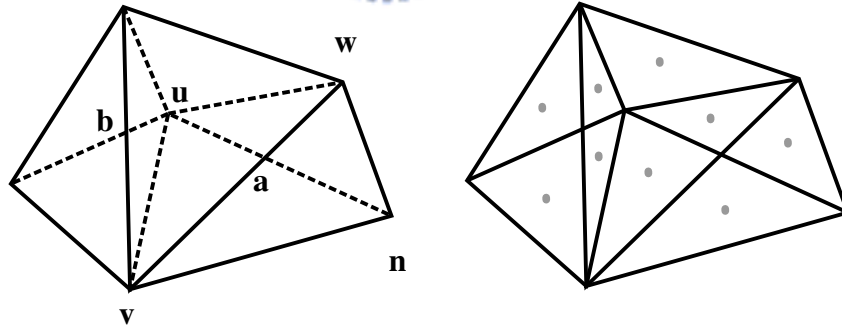


Figure 3.6: Edges overlay in the one-ring neighborhood of  $u$  (left), and the edges partition the neighborhood into 9 cells (right).

After deriving  $\mathbf{a}_i$ ,  $\mathbf{a}_{i-1}$ ,  $\mathbf{b}_i$ ,  $\mathbf{b}_{i-1}$ , and  $\mathbf{u}_{i-1}$ , we move  $\mathbf{a}$  to  $\mathbf{a}_i$  and  $\mathbf{b}$  to  $\mathbf{b}_i$  to form the partition for  $M^i$  and, similarly, move  $\mathbf{a}$  to  $\mathbf{a}_{i-1}$ ,  $\mathbf{b}$  to  $\mathbf{b}_{i-1}$ , and  $\mathbf{u}$  to  $\mathbf{u}_{i-1}$  to form the partition for  $M^{i-1}$ . See Figure 3.7. Two cells in these two region partitions are said to be in correspondence if they correspond to the same cell before moving the points  $\mathbf{a}$ ,  $\mathbf{b}$ , and  $\mathbf{u}$  to designated texture coordinates. For example,  $\triangle \mathbf{a}_i \mathbf{v} \mathbf{n}$  and  $\triangle \mathbf{a}_{i-1} \mathbf{v} \mathbf{n}$  are in correspondence.

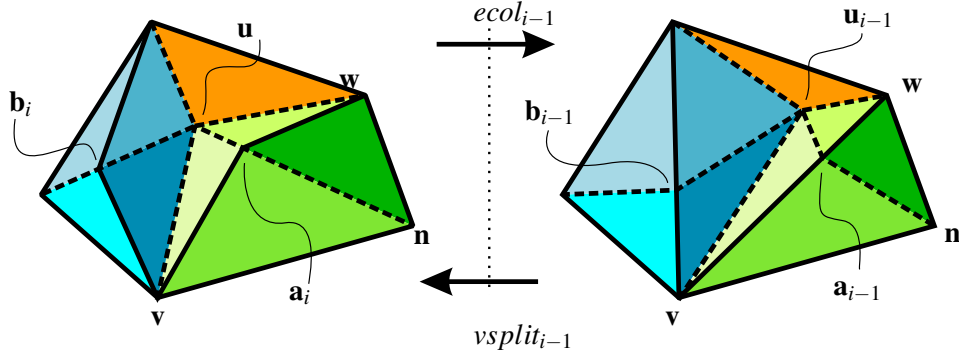


Figure 3.7: Cell correspondence between partitions of  $T^i$  (left) and  $T^{i-1}$  (right).

### 3.2.3 Texture adaptation

After cell correspondences for all cells in  $\mathcal{R}_{i-1}$  are found, the texture adaptation from  $T^i$  to  $T^{i-1}$  is performed for each pair of corresponding cells. To adapt the texture from cell  $\Delta \mathbf{a}_i \mathbf{b}_i \mathbf{c}_i$  of  $T^i$  to cell  $\Delta \mathbf{a}_{i-1} \mathbf{b}_{i-1} \mathbf{c}_{i-1}$  of  $T^{i-1}$ , we let the former be the source texture, the latter be the target polygon, and then apply hardware texture mapping. Similarly, for the backward texture adaptation from  $T^{i-1}$  to  $T^i$ ,  $\Delta \mathbf{a}_{i-1} \mathbf{b}_{i-1} \mathbf{c}_{i-1}$  is the source texture, and  $\Delta \mathbf{a}_i \mathbf{b}_i \mathbf{c}_i$  is the target polygon. Care must be taken to prevent the so called *parametric folding* in the process of texture adaptation by ensuring that areas of both cells  $\Delta \mathbf{a}_i \mathbf{b}_i \mathbf{c}_i$  and  $\Delta \mathbf{a}_{i-1} \mathbf{b}_{i-1} \mathbf{c}_{i-1}$  are positive. Note that, since we maintain all  $T_i$ ,  $i = n, \dots, 1, 0$ , in a single physical texture map, we need a temporary map to support the texture adaptation.

It is worth mentioning that the texture adaptation for all pairs of corresponding cells can be accelerated by using triangle-fan and triangle strip setups, as shown in Figure 3.8. Its cost can be further minimized by packing triangle fans and strips into arrays and using draw array commands, such as `glDrawArrays()` or `glDrawElements()`. Moreover, it can be run in parallel with the edge collapse operation.

### 3.2.4 Indexing map

Texture adaptation is basically a re-sampling process of a texture map. When a texture area of higher frequency gets adapted to a smaller texture area, blurred artifacts due to under-sampling

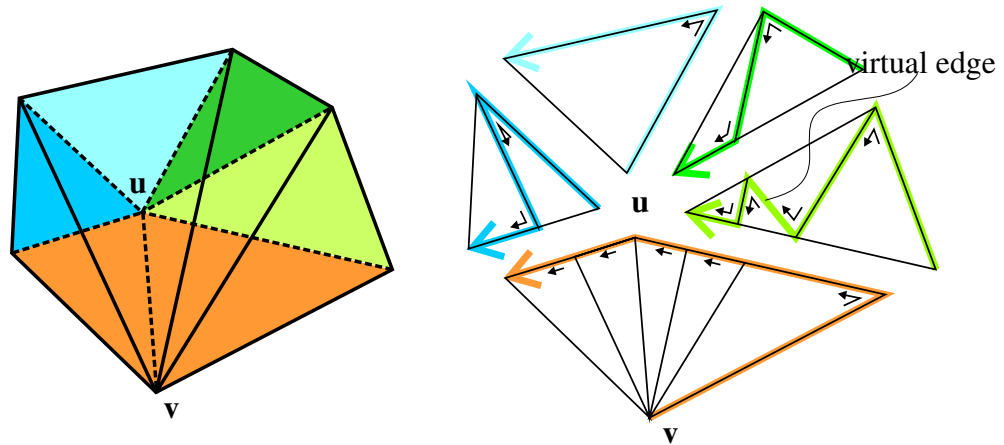


Figure 3.8: Accelerate texture adaptation by triangle-fan and triangle strip setups.

of texture samples might appear, as shown in Figure 3.9. To prevent such problems, we propose the mechanism of *indexing mapping* that uses an indexing map  $I$  to store in each texel the texture coordinate referring to the original texture map  $T$ . All the texture adaptation operations are applied to the indexing map, leaving the original texture map alone. Initially, the indexing map  $I^n$  for  $M^n$  stores in each texel the coordinate itself, that is,  $I^n(x, y) = (\frac{x+0.5}{w}, \frac{y+0.5}{h})$ , where  $w \times h$  is the resolution of the indexing map. For texture adaptation, we derive  $I^{i-1}$  from  $I^i$  in the same way as we do for deriving  $T^{i-1}$  from  $T^i$ , for  $i = n, \dots, 2, 1$ . Figure 3.10 shows the indexing map with each texel value  $s$  and  $t$  coordinates, which is visualized as red and green color respectively.

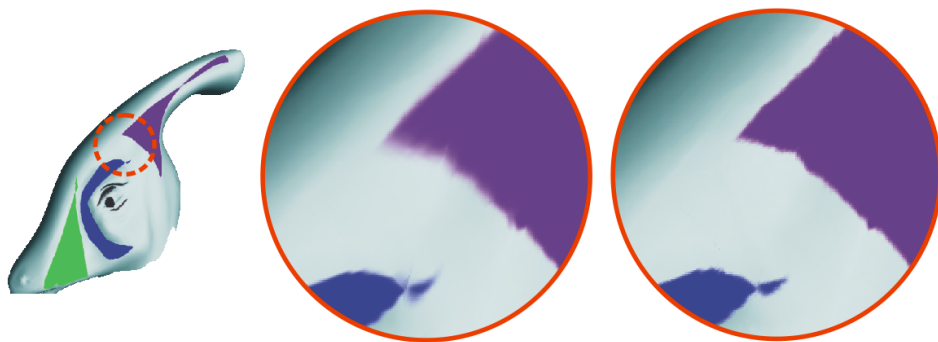


Figure 3.9: Blurred artifacts introduced by texture adaptation (left) and minimized by *indexing mapping* (right).

Figure 3.11 illustrates an example of indexing mapping. The surface  $M$  is texture mapped with  $T$ . When  $M$  is simplified to  $M'$ ,  $T$  is adapted to  $T'$  and, in this example, the texels on  $T$  are re-sampled to two texels on  $T'$ . As a result, one gets a blurred textured image while

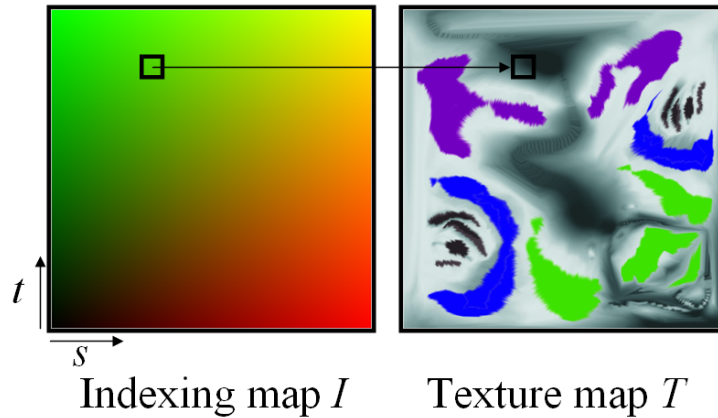


Figure 3.10: Each texel on indexing map is the texture coordinate referring to the texel on original map.

texture mapping  $T'$  onto  $M'$ . With the proposed indexing mapping, the texel values of  $I'$  are the coordinates of  $T$ , and are used to access to the original texels on  $T$ . Therefore, blurred artifacts are reduced.

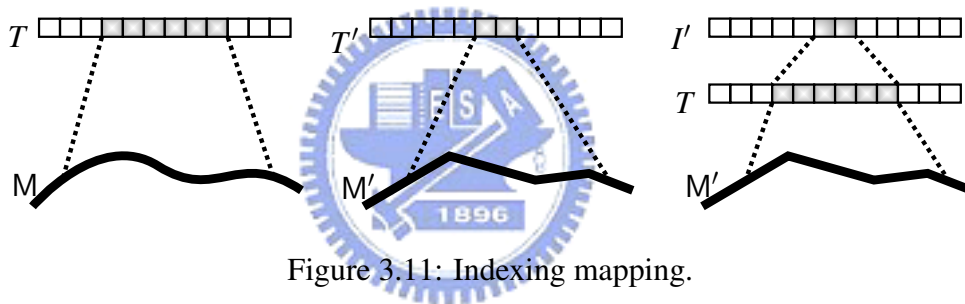


Figure 3.11: Indexing mapping.

The use of indexing map is also advantageous when more than one texture map, such as a combination of a color map, a normal map, and a bump map, see Figure 3.12, are associated with the model since texture adaptation is applied to the indexing map only, leaving all the maps untouched.

Indirect accessing of a texture map is not supported by graphics APIs such as OpenGL. Fortunately, it can be implemented using the fragment shader. Another issue is the precision of the indexing map. In our experience, a texture map of 16-bit precision is sufficient to deliver acceptable image quality and performance for texture adaptation. See the example shown in Figure 3.13.

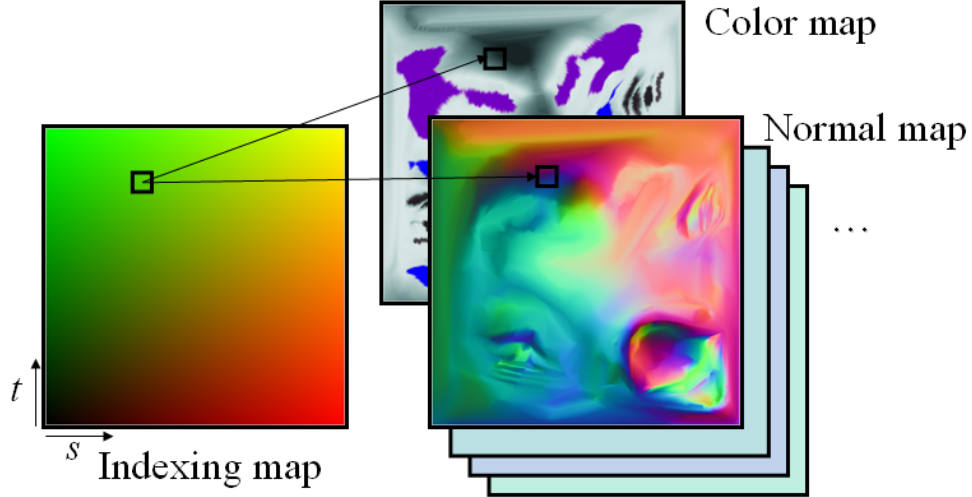


Figure 3.12: Use one single indexing map to access to all the original maps.

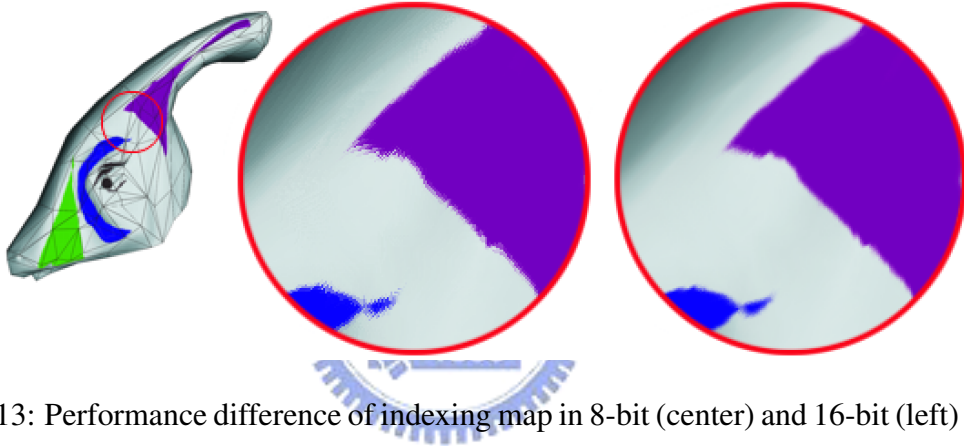


Figure 3.13: Performance difference of indexing map in 8-bit (center) and 16-bit (left) precision.

### 3.3 Experimental Results

In our experiments, we used a Pentium4 3.0Ghz platform with an *n*VIDIA GeForce 6800GT graphics card. All test models are using a  $1024 \times 1024$  16-bit floating indexing map.

We use the render-to-texture feature to avoid transferring texture maps between graphics hardware and the host CPU. In our experiments, we observed that the render-to-texture setup is the most expensive operation in texture adaptation even with the help of framebuffer objects (FBOs). To minimize the setup cost, we simply collect texture adaptations that are applied to disjoint regions and perform adaptations together. When we simplify the model from  $M^j$  to  $M^i$ , where  $j > i$ , we start with the set COLLECTION containing only the texture adaptation from  $T^j$  to  $T^{j-1}$  and the set  $R = \{R_j\}$ . We then check, in the sequence of  $k = j - 1, \dots, i + 1$ , to see if



$R_k$  overlays with any element of  $R$ . If not,  $R_k$  is inserted into  $R$  and the texture adaptation from  $T^k$  to  $T^{k-1}$  is put into COLLECTION. After doing this cycle, we perform all the texture adaptations collected in COLLECTION with one common setup. We repeatedly perform the packing of texture adaptations until all the texture adaptations are done. A similar approach is applied to the refinement process. We have observed that 13.73 and 20.20 texture adaptations on average are packed to share a common setup for the parasaur head and bunny head, respectively, and such a simple approach achieves about a  $13.66\times$  speedup factor for the parasaur head model and  $17.51\times$  for bunny head model, as depicted in Table 3.1, and the entire run-time cost is also listed in Table 3.2. Moreover, the setup can be further shared among objects in an environment with multiple objects.

Table 3.1: Speedup by packing texture adaptations.

model	original model (polygons)	simplified model (polygons)	w/o packing		w/ packing			
			# of texture adaptations	avg. time (ms)	# of packs	# of texture adaptations per pack	avg. time (ms)	speedup factor
parasaur head	7,835	499	3,611	0.7581	263	13.73	0.0555	$13.66\times$
bunny head	17,483	500	8,545	0.7616	423	20.20	0.0435	$17.51\times$

Table 3.2: The entire run-time cost of simplifying PM w/ and w/o texture adaptation.

model	original model (polygons)	simplified model (polygons)	run-time cost (ms)	
			w/o texture adaptation	w/ texture adaptation
parasaur head	7,835	499	52.99	253.42
bunny head	17,483	500	137.16	508.87

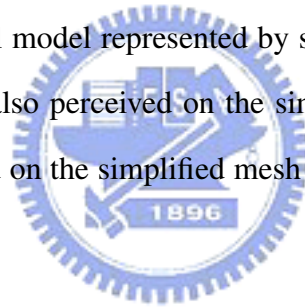
Experimental tests have been done on several models. For PM construction, a quadric error metric (QEM) of five dimensions [30] is applied to all tested models. Figure 3.14 demonstrates the power of texture adaptation on the parasaur head at bottom row. A comparison with QEM of 5D without texture adaptation and APS [15] without texture adaptation is shown at top and middle row, respectively. The texture map for the parasaur head is derived using signal-specialized parameterization [47]. Texture distortion becomes noticeable when the mesh is simplified to 2000 polygons and becomes obvious for meshes of 1000 and 499 polygons. We have observed that texture distortion is almost eliminated by texture adaptation even in the textured image of the

simplified model with 499 polygons. A normal mapped parasaur head is also greatly improved by texture adaptation in Figure 3.15.

The horse model shown in Figures 3.16 is parameterized by geometric-stretch-minimizing parameterization [48]. Texture distortion is perceivable in the simplified meshes and is almost eliminated by using texture adaptation.

Figure 3.17 depicts the performance of texture adaptation on a bunny head model with color map parameterized by geometric-stretch-minimizing parameterization. The bottom row shows the parameterization and texture map. The texture distortion is visualized by texture mapping the check board onto the model. For the textured image without texture adaptation, texture distortion is apparent inside the enlarged image. On the other hand, texture distortion is not noticeable in the textured image with texture adaptation.

Our last tested model is a swirl model represented by swirled texture coordinates on a curved surface. Texture distortion is also perceived on the simplified swirl model. See Figure 3.18. The swirl lines get straightened on the simplified mesh but are well preserved by using texture adaptation.



### 3.3.1 Preprocessing time

Table 3.3 depicts the preprocessing time for constructing the entire PM sequence for the test models w/o and w/ texture adaptation. The PM construction w/ texture adaptation requires additional time for computing the correspondences and performing the texture adaptation operations, and it can be done in a few seconds.

### 3.3.2 Reversibility

Although the texture adaptation is theoretically an invertible operation, it is, in practice, applied to the texture map of discrete texels. Repeatedly applying texture adaptation and its inverse

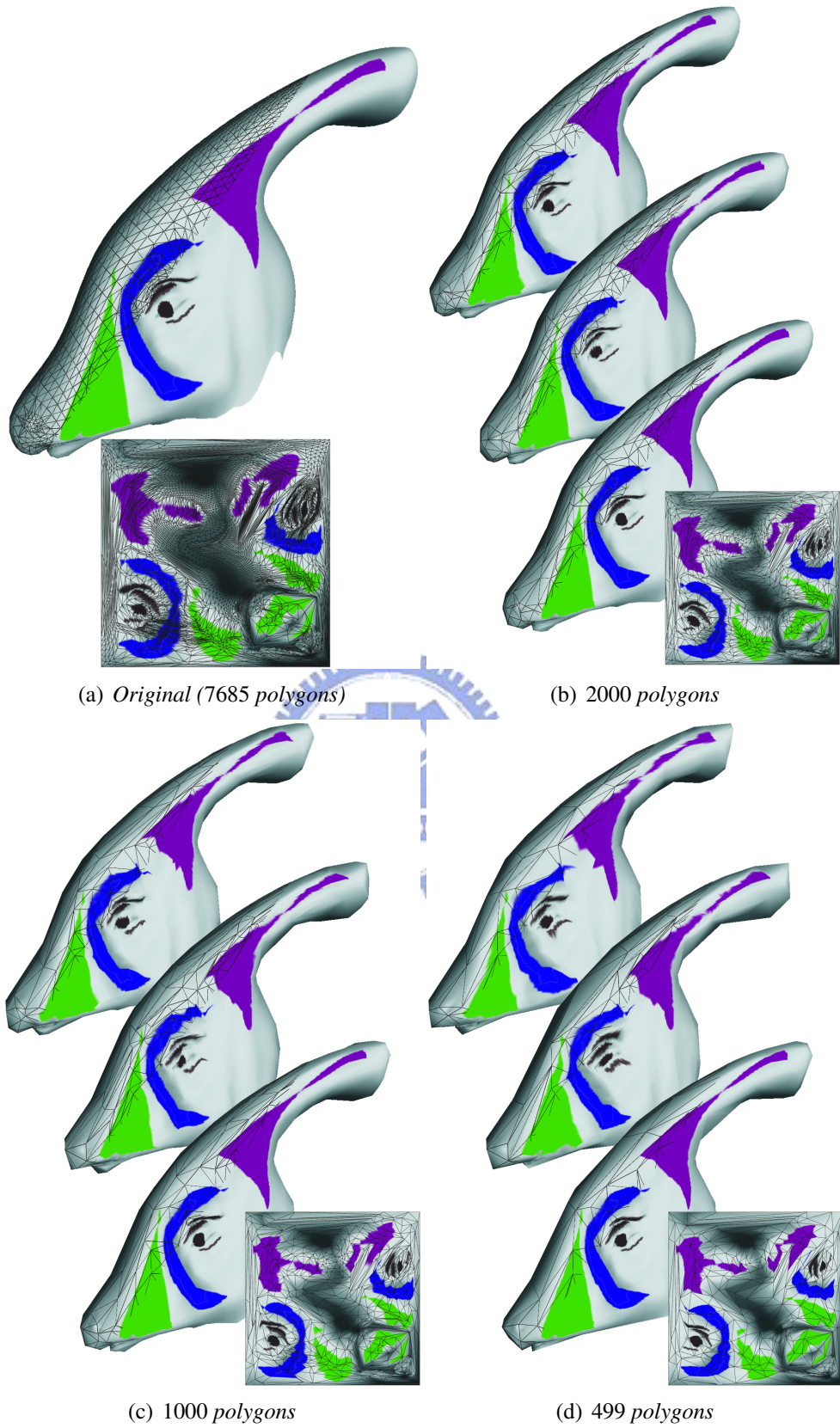


Figure 3.14: Textured images of parasaur head model. Top row of (b),(c),(d): simplified by QEM of 5D without texture adaptation, middle row: by APS without texture adaptation, bottom row: by QEM of 5D with texture adaptation.

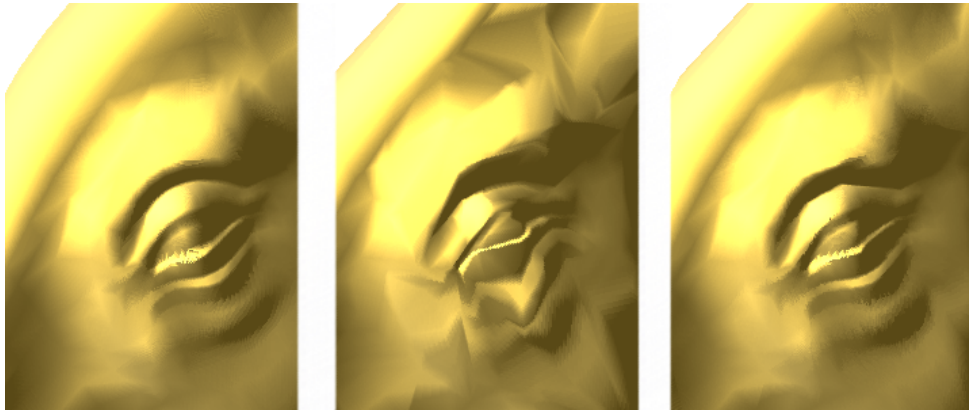


Figure 3.15: Normal mapped parasaur head. left: 7685 polygons, center: 499 polygons, and right: 499 polygons with texture adaptation.



Figure 3.16: Textured images of horse model. Left: original model of 8160 polygons with parameterized texture map, center: simplified model of 800 polygons without texture adaptation, right: simplified model of 800 polygons with texture adaptation.

Table 3.3: The preprocessing time for constructing the entire PM sequence of the test models.

model	original model (polygons)	simplified model (polygons)	preprocessing time (ms)	
			w/o texture adaptation	w/ texture adaptation
parasaur head	7,835	499	655.23	1,263.73
zebra	8,160	800	679.19	1,189.21
bunny head	17,483	500	1,629.11	4,120.62
swirled mesh	7,688	399	690.96	8,066.98

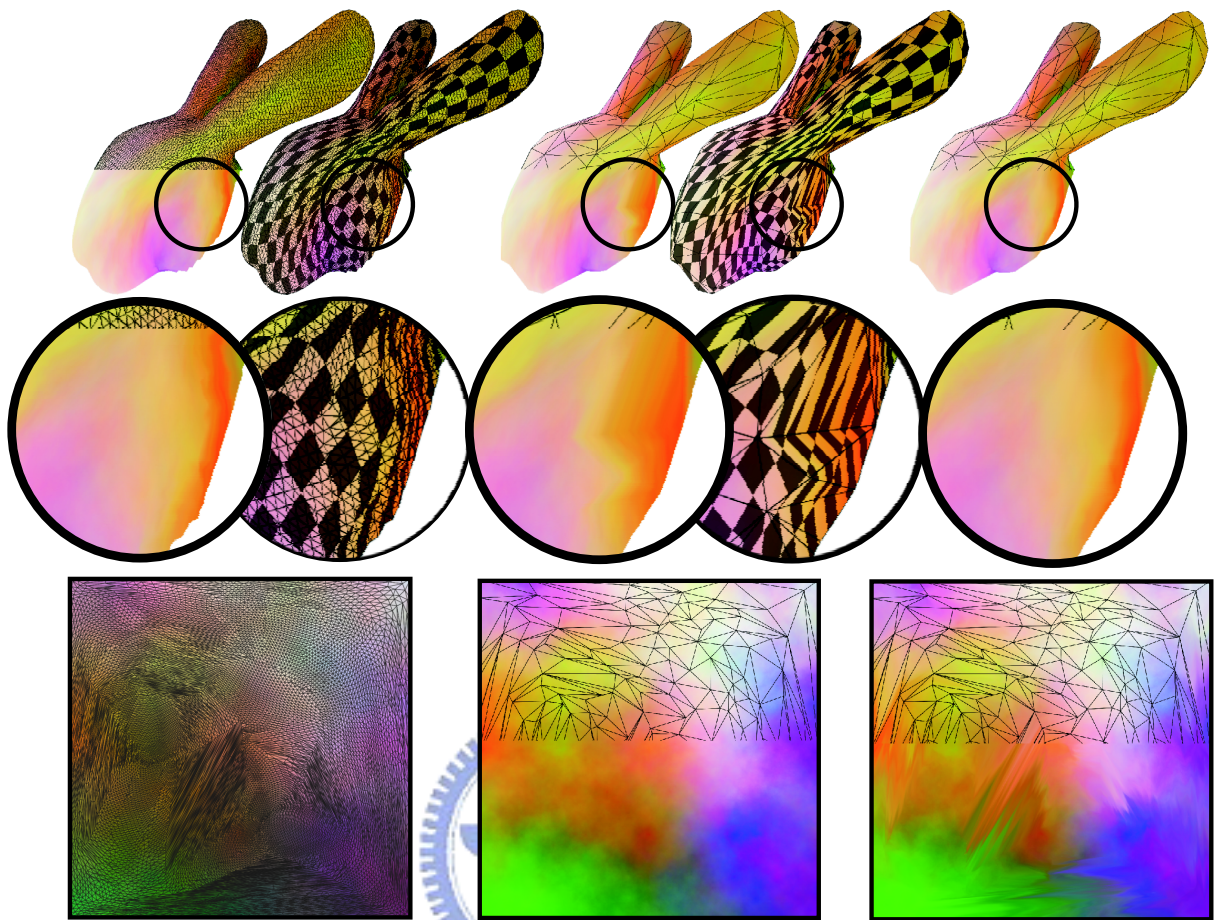


Figure 3.17: Left: original model of 17483 polygons, center: simplified model of 500 polygons without texture adaptation, right: simplified model of 500 polygons with texture adaptation.

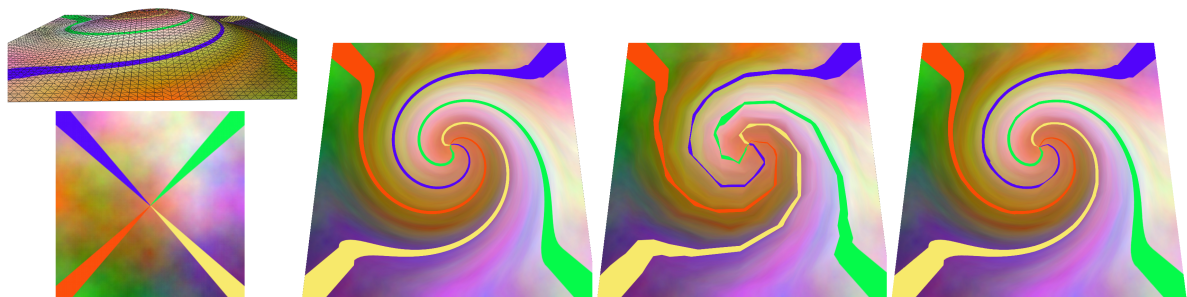


Figure 3.18: From left to right, mesh with swirled texture coordinates and its color texture map, original mesh (7688 polygons), simplified mesh (399 polygons), and the same simplified mesh with texture adaptation.

many times may introduce artifacts.

In our experiment, we have found that only very small and unnoticeable artifacts exist in images resulting from several runs of fine-to-coarse simplification and coarse-to-fine refinement. Figure 3.19 shows the original mesh on top, and in the bottom the mesh results from three iterations of both fine-to-coarse simplification and coarse-to-fine refinement.

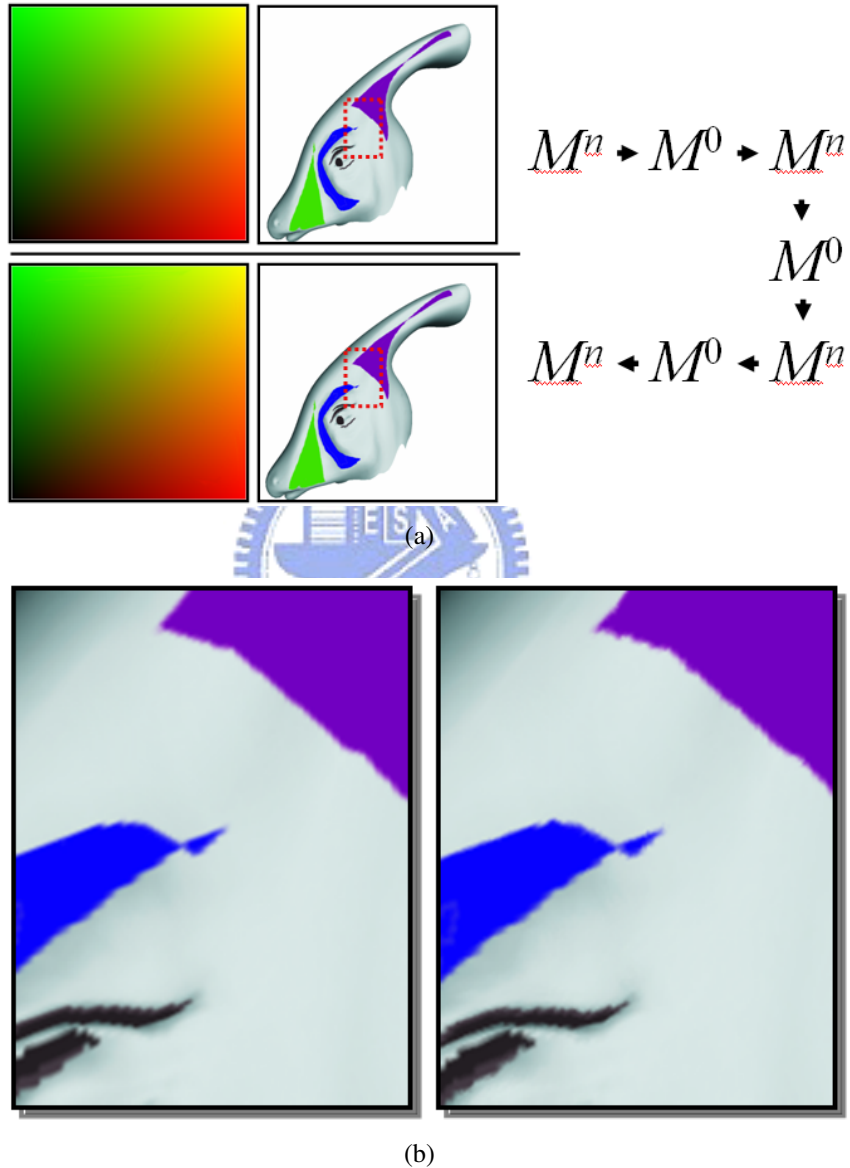


Figure 3.19: (a) The top image is the original mesh, and the bottom image is the mesh that has been applied three iterations of both fine-to-coarse and coarse-to-fine transformation with the texture adaptation. (b) The zoomed images, on the left is the original mesh and on the right is the resulting mesh.

### 3.3.3 Multiple charts

It is usually impossible to parameterize a whole model to 2D domain using available parameterization technique. Meshes are often subdivided into multiple charts in order to derive disk-like patches or to have patches with smaller parameterization stretch. The proposed texture adaptation is a local operation within the same texture map. In consequence, current implementation limits texture adaptation within a texture map. Moreover, a vertex on the chart boundary is constrained to be merged to other vertices on the same chart boundary. Otherwise, there will be some artifacts of texture distortion around the boundary. One possible approach to reduce such texture distortion around chart boundaries is to straighten the chart boundaries, as done by the method used in [48].

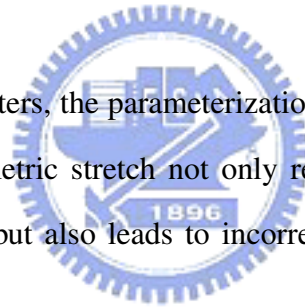


# Chapter 4

## A New Mapping-Based Error Metric

### 4.1 Introduction

As mentioned in previous chapters, the parameterization of most meshes inevitably introduces geometric stretch. Such geometric stretch not only results in texture distortion in mapping textures to simplified meshes but also leads to incorrect evaluation for mapping-based error metric such as APS [15].



APS defines the texture deviation through the mapping defined by the texture map, which is essentially a result of mesh parameterization. For an edge collapse, each point on the neighborhood of the collapsed edge is mapped to a point on the simplified mesh through texture map. That is, for each texture parameter  $\mathbf{x}$ , as in the Equation 2.1, the texture deviation of  $\mathbf{x}$  is defined as the distance between such two points that are in correspondence through  $\mathbf{x}$ . While APS successfully measures the deviation of texture mapping, it leads to texture distortion in mapping textures to the simplified meshes and, moreover, fails to provide an accurate measurement for geometry error introduced in the edge collapse. One way to reduce the distortion of mesh parameterization is to dissect the given mesh into multiple charts and parameterize each chart with less distortion. However, the charting strategy imposes several constraints in mesh simplification. For example, vertices on the chart boundary can only be collapsed to the vertices on



the boundary and the chart corners must be preserved.

In the last chapter, we have proposed texture adaptation approach to successfully eliminate the texture distortion in mapping textures to simplified meshes. The next meaningful and important goal would be the search of a better geometric error metric for simplifying textured meshes. Our motivation is to reduce the impact of the geometric stretch introduced in mesh parameterization and provide a meaningful geometric error metric which is far more accurate than APS, even QEM. In this chapter, we describe a mapping-based error metric, which is a variant of APS and based on the texture adaptation approach proposed in the previous chapter. Methods for both maximum error and average error, and incremental error and total error will be described.

## 4.2 A New Mapping-Based Error Metric

Recall that, in the framework of texture distortion, we partition the texture domain corresponding to the 1-ring neighborhood of the collapsed vertex into cells. The cell partition for domains before and after the edge collapse forms an one-to-one correspondence between cells. Texture adaptation is then performed for each pair of corresponding cells.

Let's consider the half edge collapse  $\mathbf{U} \xrightarrow{ecol_{i-1}} \mathbf{V}$  that simplifies the mesh from  $M^i$  to  $M^{i-1}$ . Let  $\mathcal{R}_i$  and  $\mathcal{R}_{i-1}$  denote the 1-ring neighborhoods of the collapsed vertex  $\mathbf{U}$ , and  $\mathcal{F}_i^{-1}$  and  $\mathcal{F}_{i-1}^{-1}$  represent the mapping from texture domain to 3D space, for the mesh before and after edge collapse, respectively. Note that for a texture coordinate  $\mathbf{x}$  in  $\mathcal{R}'_i$ , the texture domain corresponding to  $\mathcal{R}_i$ , APS defines the incremental texture deviation as

$$\text{texture\_deviation}_{incr}(\mathbf{x}) = \|\mathcal{F}_i^{-1}(\mathbf{x}) - \mathcal{F}_{i-1}^{-1}(\mathbf{x})\|. \quad (4.1)$$

Based on the texture adaptation scheme, a texture coordinate  $\mathbf{x}_i$  in the texture domain  $\mathcal{R}'_i$  actually corresponds to the texture coordinate  $\mathbf{x}_{i-1}$  in the texture domain  $\mathcal{R}'_{i-1}$ ; see Figure 4.1. Such a correspondence is established via the cell correspondence, in which a cell in  $\mathcal{R}'_i$  maps

uniquely to a cell in  $\mathcal{R}'_{i-1}$ , and the interpolation involved in copying the texture content from the cell to the corresponding cell. Using such an indirect mapping, we define a new incremental error metric as follows:

$$e_{incr}(\mathbf{x}_i) = \|\mathcal{F}_i^{-1}(\mathbf{x}_i) - \mathcal{F}_{i-1}^{-1}(\mathbf{x}_{i-1})\|, \quad (4.2)$$

where  $\mathbf{x}_i$  is the point with Barycentric coordinate  $(\alpha_i, \beta_i)$  in cell $_i$  and  $\mathbf{x}_{i-1}$  is the point in cell $_{i-1}$  that has the same Barycentric coordinate  $(\alpha_i, \beta_i)$  as  $\mathbf{x}_i$ , and cell $_i$  and cell $_{i-1}$  are cells in correspondence.

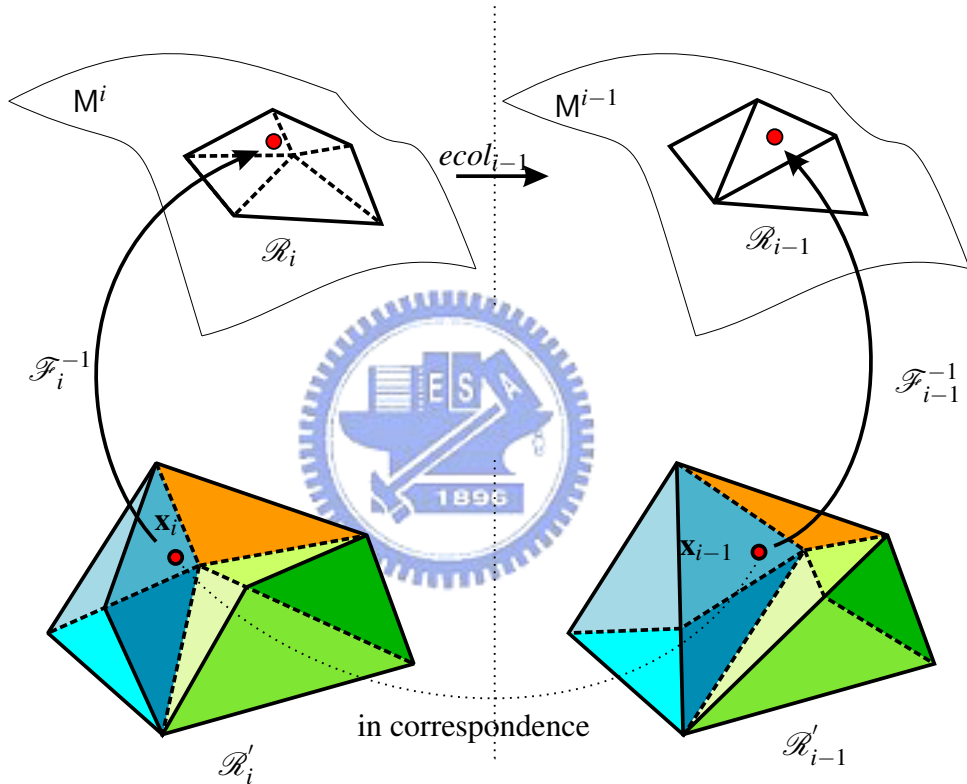


Figure 4.1: The correspondence of  $\mathbf{x}_i$  and  $\mathbf{x}_{i-1}$  is established via the cell correspondence.

### 4.2.1 Maximum error

To evaluate simplification error for an edge collapse  $\mathbf{U} \xrightarrow{e_{col_{i-1}}} \mathbf{V}$ , we have two choices: maximum error vs. average error. The maximum error is advantageous when a guaranteed error bound is expected for some applications such as medical and scientific visualization. On the other hand, the average error indicates the error across the entire meshes. The maximum error of the new

incremental error metric can be defined as

$$\max_{\mathbf{x}_i \in \mathcal{R}'_i} e_{incr}(\mathbf{x}_i) = \max_{\mathbf{x}_i \in \mathcal{R}'_i} \|\mathcal{F}_i^{-1}(\mathbf{x}_i) - \mathcal{F}_{i-1}^{-1}(\mathbf{x}_{i-1})\|. \quad (4.3)$$

Since for each pair of corresponding cells  $\text{cell}_i$  and  $\text{cell}_{i-1}$ , the value of  $e_{incr}(\mathbf{x}_i)$  varies linearly across the cell, its maximum locate on the corners of the cell. In consequence,  $\max_{\mathbf{x}_i \in \mathcal{R}'_i} e_{incr}(\mathbf{x}_i)$  will happen at the corners of all cells in  $\mathcal{R}'_i$ . Recall that the cells are derived by computing the nearest points of two edges, one from  $M^i$  and another from  $M^{i-1}$ , that intersect on the texture domain. Let  $\mathcal{S}_i$  denote the set of the collapsed vertex  $\mathbf{U}_i$  and  $\mathbf{A}_i$  for all pairs of the nearest points  $(\mathbf{A}_i, \mathbf{A}_{i-1})$  and  $\mathcal{S}'_i$  denote the set of  $\mathbf{a}_i$  and  $\mathbf{u}_i$ , where  $\mathbf{A}_i = \mathcal{F}_i^{-1}(\mathbf{a}_i)$  and  $\mathbf{U}_i = \mathcal{F}_i^{-1}(\mathbf{u}_i)$ , for  $\mathbf{a}_i$  and  $\mathbf{u}_i$  in  $\mathcal{S}_i$ . Consequently, we can rewrite  $\max_{\mathbf{x}_i \in \mathcal{R}'_i} e_{incr}(\mathbf{x}_i)$  as

$$\begin{aligned} \max_{\mathbf{x}_i \in \mathcal{R}'_i} e_{incr}(\mathbf{x}_i) &= \max_{\mathbf{a}_i \in \mathcal{S}'_i} \|\mathcal{F}_i^{-1}(\mathbf{a}_i) - \mathcal{F}_{i-1}^{-1}(\mathbf{a}_{i-1})\| \\ &= \max_{\mathbf{A}_i \in \mathcal{S}_i} \|\mathbf{A}_i - \mathbf{A}_{i-1}\|. \end{aligned} \quad (4.4)$$

The total error of the new error metric can be evaluated by propagating the incremental error using per face bounding box approach proposed in [15].

### 4.2.2 Average error

To evaluate the average error, a scheme is required to sample the texture domain  $\mathcal{R}'_i$  and average the derived error. To derive the corresponding texels  $\mathbf{x}_i$  and  $\mathbf{x}_{i-1}$  for evaluating  $e_{incr}(\mathbf{x}_i)$ , texture adaptation has to be performed and hence indexing map is required. In the following, we describe how to compute  $e_{incr}(\mathbf{x}_i)$  using indexing map. Recall that the indexing map is initially assigned the texture coordinates of the texture map, i.e.,

$$I^n(\mathbf{x}_n) = \mathbf{x}_n,$$

where  $\mathbf{x}_n$  is the texture coordinate, which implies that the indexing map inherits the 2D to 3D mapping imposed on the original texture maps. In consequence, the incremental error metric remains the same as before.

In the course of texture adaptation, only the content of the indexing map is updated, leaving the original texture maps untouched. Hence, after a sequence of texture adaptations, the value stored in each texel  $\mathbf{x}_i$  of  $I^i$  refers to the texture coordinate in the original map that represents the source of  $\mathbf{x}_i$ , i.e.,

$$I^i(\mathbf{x}_i) = \mathbf{x}_n,$$

where  $\mathbf{x}_n$  is adapted to  $\mathbf{x}_i$  by a sequence of texture adaptations  $I^n \rightarrow I^{n-1} \rightarrow \dots \rightarrow I^i$ ; as shown in Figure 4.2. This property enables an effective way for the evaluation of total average error, expressed as follows:

$$\begin{aligned} e_{total}(\mathbf{x}_i) &= \|\mathcal{F}_n^{-1}(\mathbf{x}_n) - \mathcal{F}_i^{-1}(\mathbf{x}_i)\| \\ &= \|\mathcal{F}_n^{-1}(I^i(\mathbf{x}_i)) - \mathcal{F}_i^{-1}(\mathbf{x}_i)\|, \end{aligned}$$

where  $\mathbf{x}_n$  is the corresponding point of  $\mathbf{x}_i$  in the original texture map. Note that, without indexing map, finding  $\mathbf{x}_n$  for each  $\mathbf{x}_i$  may require a complicated backtracking process.

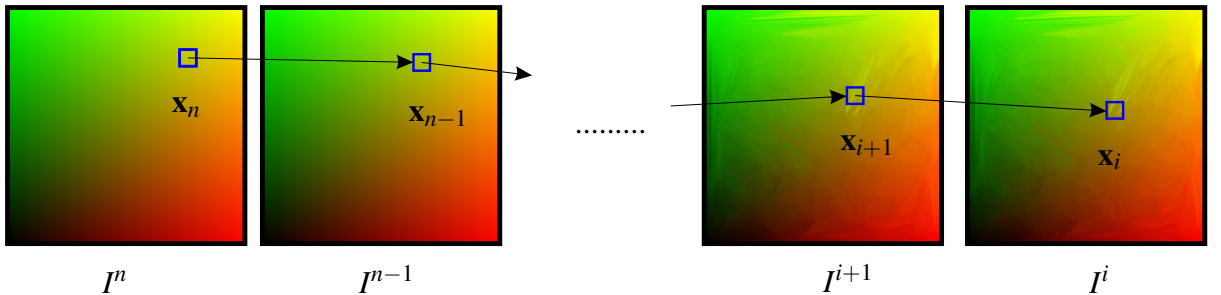


Figure 4.2:  $\mathbf{x}_n$  is adapted to  $\mathbf{x}_i$  by a sequence of texture adaptations  $I^n \rightarrow I^{n-1} \rightarrow \dots \rightarrow I^i$ .

### 4.3 Experimental Results

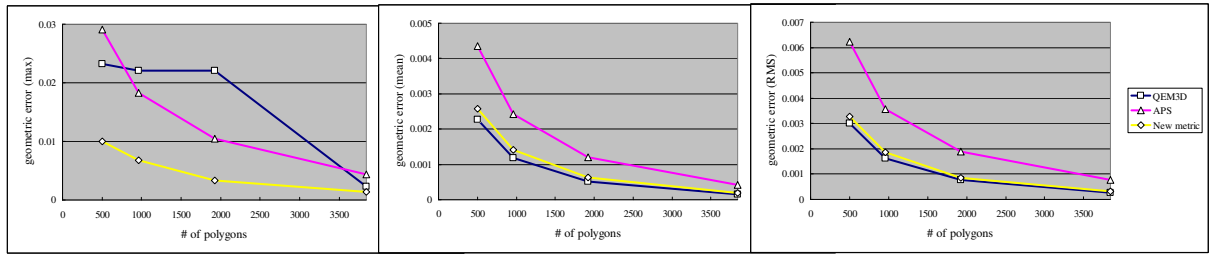
We use the proposed error metric to measure the total surface-to-surface maximum error introduced in mesh simplification in the following tests. The tested models are the parasaur head

model with 7685 polygons, the bunny head model with 17483 polygons, and the zebra model with 8160 polygons. The test platform is a Pentium4 in 3.0Ghz with 2GB main memory and a graphics card of nVIDIA GeForce 6800GT with 256MB video memory. We first simplify the models without texture adaptation by QEM, APS and the new error metric, then simplify the models with texture adaptation by QEM, APS and the new error metric.

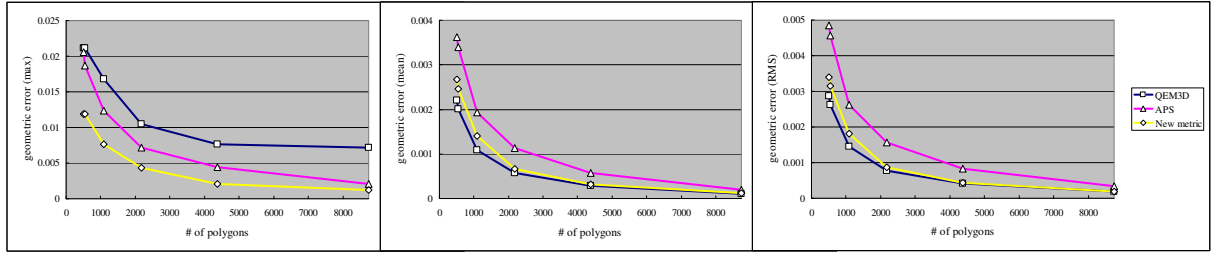
To evaluate the geometric error, all original models are normalized to fit into a unit cube and the geometric error is evaluated by the tool *Metro* [9]. The tool *Metro*, which is publicly available on the internet, not only computes and stores Hausdorff distance as per-vertex pseudo color for visualizing the mesh quality and reports histogram of error distribution, but also provides us a fair comparison to the existing error metrics, such as QEM [30] and APS [15]. The version of *Metro* we used for the following tests is V4.0.6, and the parameter for *Metro* is “-c -C 0.00 0.03 -s1 -u -n1000000”, i.e., the range for visualizing the geometric error is  $0.00 \sim 0.03$  (color blue means larger geometric error, red means in-between, and green means the error is close to zero).

We firstly compare the proposed error metric to QEM in 3D and APS under mesh simplification without texture adaptation. Figure 4.3 plots the *max*, *mean*, and *RMS* (from left to right) geometric error measured by *Metro*, for parasaur head, bunny head, and as well as zebra model. The new error metric is better than APS for *max*, *mean*, and *RMS*, and better than QEM under *max* geometric error while reveal almost similar performance as QEM for *mean* and *RMS*. The main reason is that QEM is a kind of average error metric while our new error metric measures the total surface-to-surface maximum error. The visualization of mesh quality is given in Figure 4.4, Figure 4.5, and Figure 4.6.

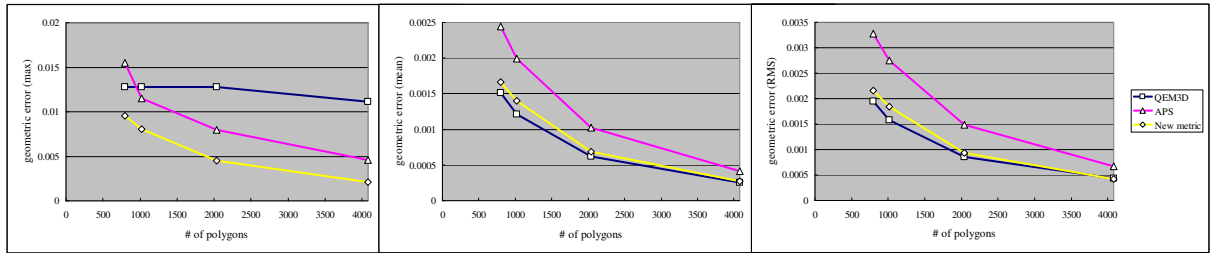
Since texture mapping is essential to the visual appearance of a model, we then compare the proposed error metric to both QEM in 5D and APS with texture adaptation applied, and the results reported by *Metro* are presented in Figure 4.7. It appears that the proposed error metric performs better than QEM and APS. Figure 4.8 depicts the error distribution of the simplified mesh of parasaur head model in 500 polygons. It shows that, for larger errors, the proposed



(a) parasaur head

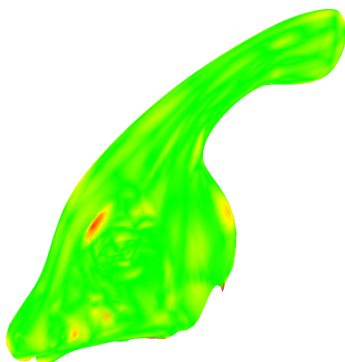


(b) bunny head

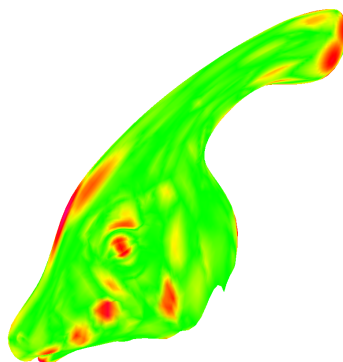


(c) zebra

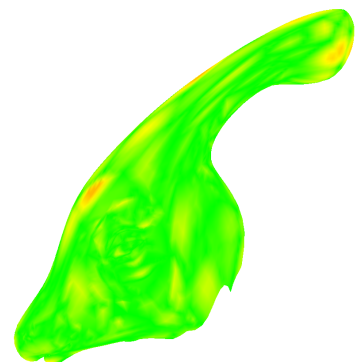
Figure 4.3: The geometric error of simplified models without texture adaptation (measured by *Metro*).



(a) QEM3D



(b) APS



(c) New error metric

Figure 4.4: The simplified parasaur head model without texture adaptation (polygon count: 7685  $\rightarrow$  500) by QEM, APS and the new error metric.

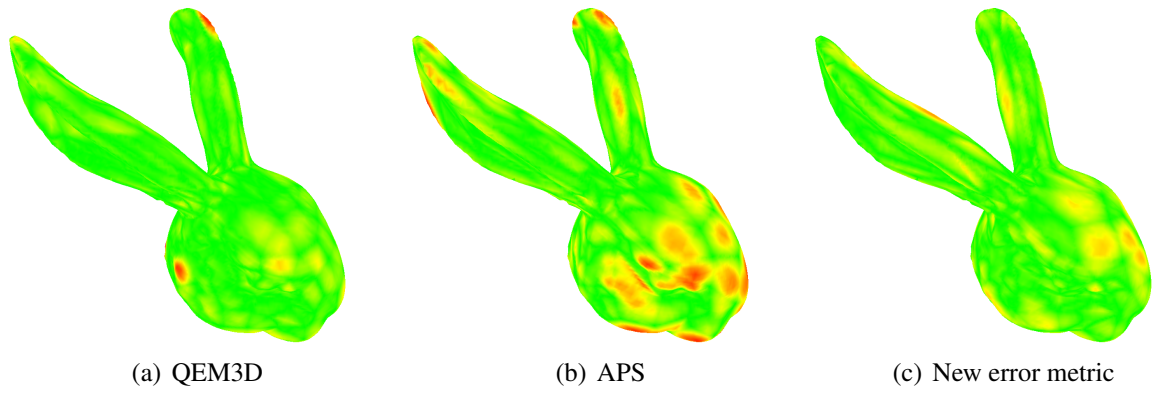


Figure 4.5: The simplified bunny head model without texture adaptation (polygon count: 17483  $\rightarrow$  500) by QEM, APS and the new error metric.

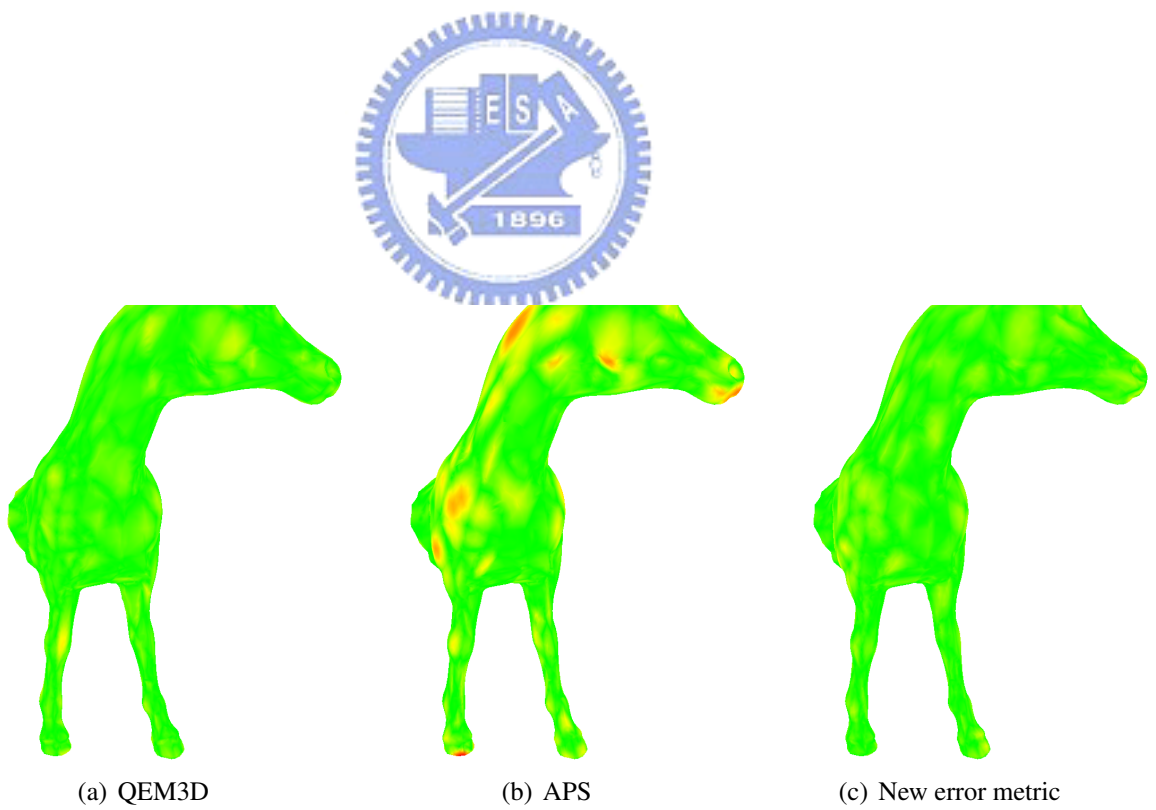
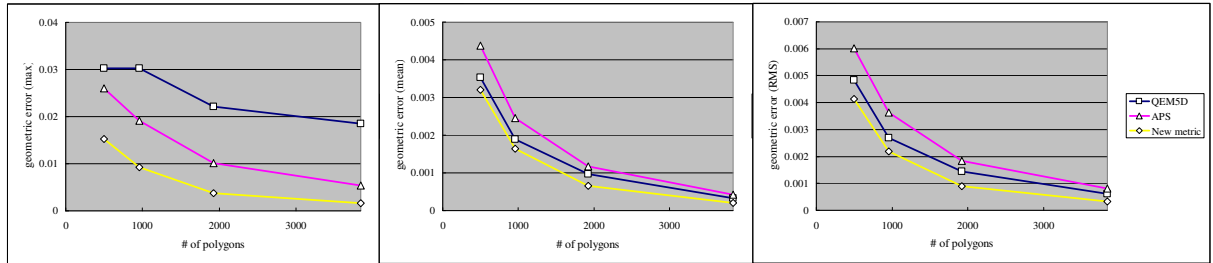
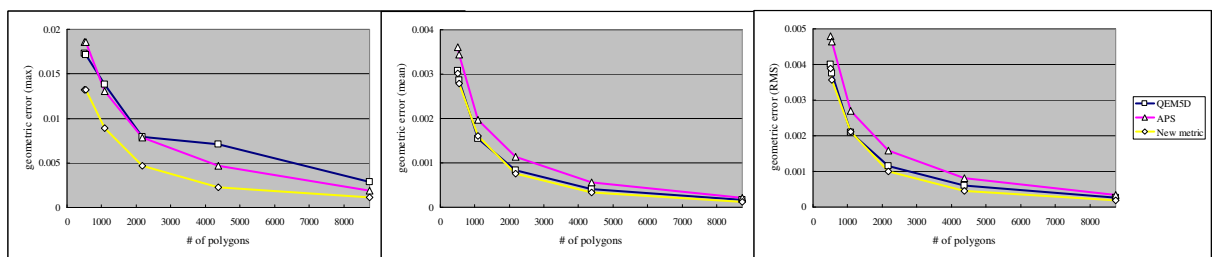


Figure 4.6: The simplified zebra model without texture adaptation (polygon count: 8160  $\rightarrow$  800) by QEM, APS and the new error metric.

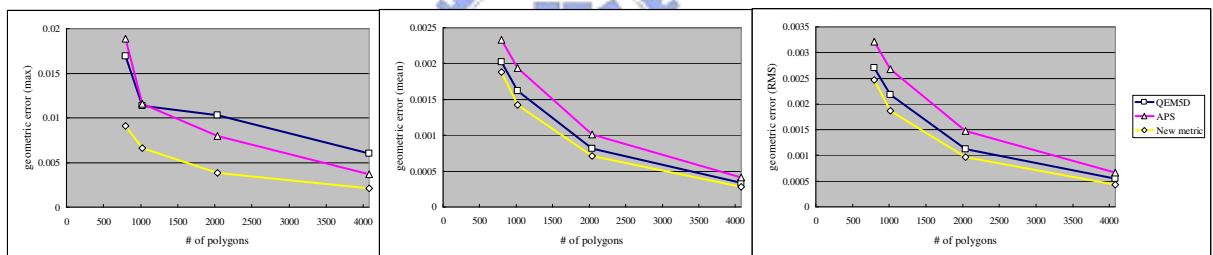
new error metric has less percentage than QEM and APS. In addition, the visualization of mesh quality for parasaur head, bunny head, and zebra model is given in Figure 4.9, Figure 4.10 and 4.11, respectively.



(a) parasaur head



(b) bunny head



(c) zebra

Figure 4.7: The geometric error of the simplified models with texture adaptation (measured by *Metro*).

The preprocessing time for constructing the PM sequence for the test models under different error metric are depicts in Table 4.1 and Table 4.2, note that the new error metric has not yet been optimized.



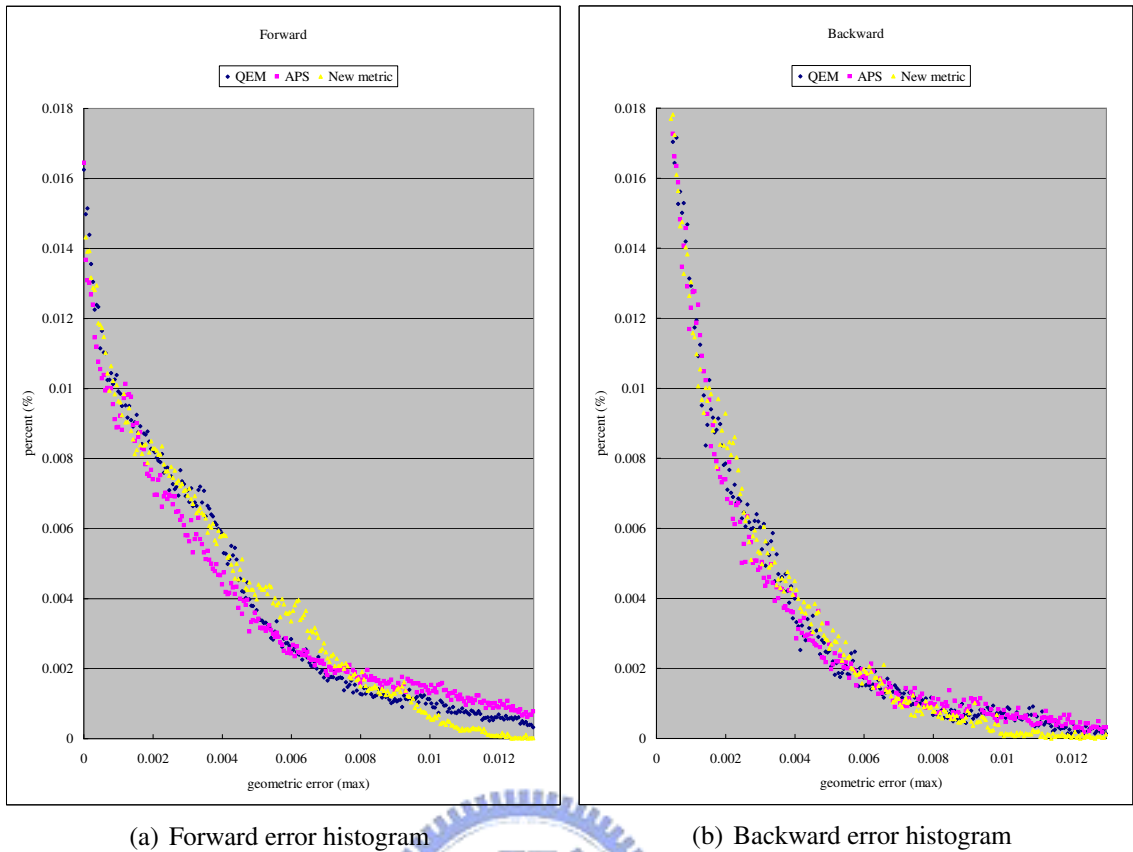


Figure 4.8: The error distribution (in percentage) of the simplified parasaur head with texture adaptation, QEM: dark blue, APS: pink, and the new error metric: yellow.

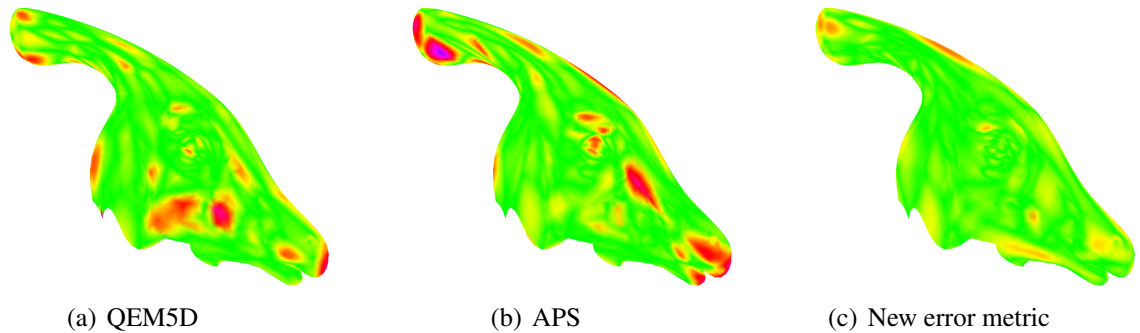


Figure 4.9: The simplified parasaur head model with texture adaptation (polygon count: 7685  $\rightarrow$  500) by QEM, APS and the new error metric.

Table 4.1: The preprocessing time for constructing the entire PM sequence w/o texture adaptation under different error metrics.

model	original model (polygons)	simplified model (polygons)	preprocessing time (ms)		
			QEM3D	APS	New error metric
parasaur head	7,835	499	679.14	1,618.94	8,023.30
zebra	8,160	800	666.49	1,597.98	2,906.21
bunny head	17,483	500	1,557.19	4,087.67	39,814.59

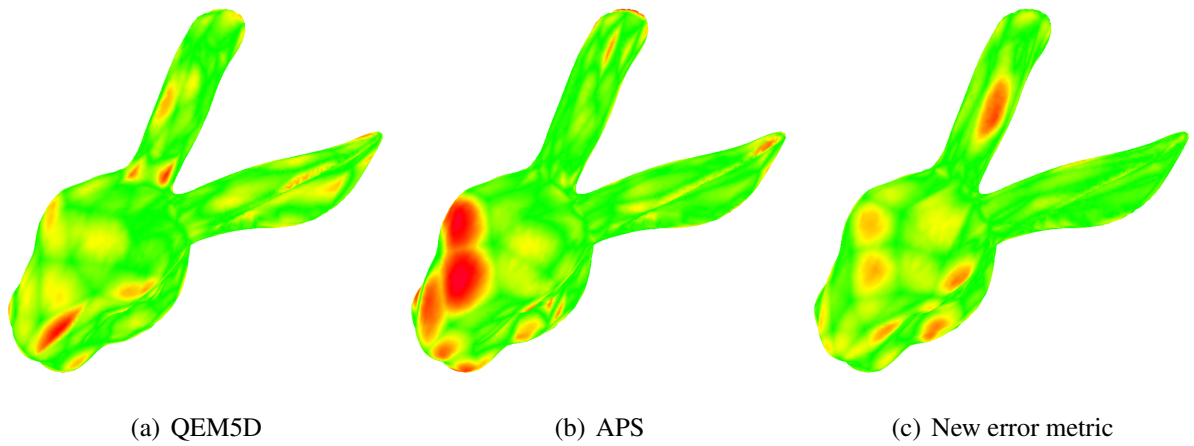


Figure 4.10: The simplified bunny head model with texture adaptation (polygon count: 17483  $\rightarrow$  500) by QEM, APS and the new error metric.

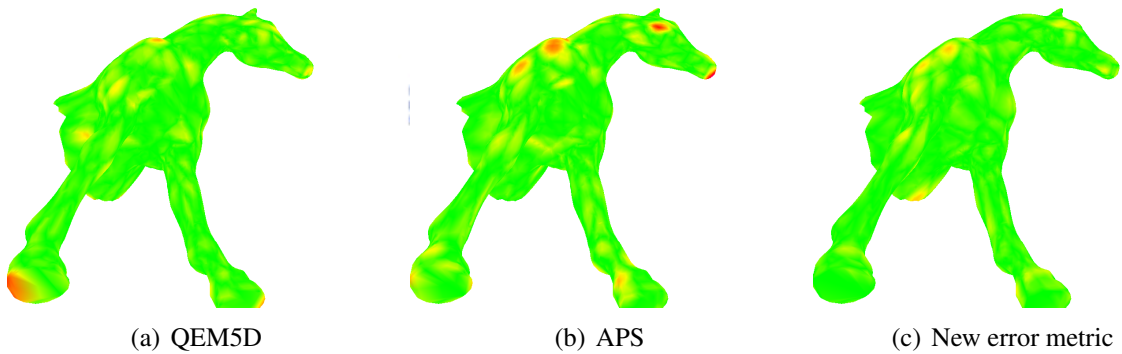


Figure 4.11: The simplified zebra model with texture adaptation (polygon count: 8160  $\rightarrow$  800) by QEM, APS and the new error metric.

Table 4.2: The preprocessing time for constructing the entire PM sequence w/ texture adaptation under different error metrics.

model	original model (polygons)	simplified model (polygons)	preprocessing time (ms)		
			QEM5D	APS	New error metric
parasauro head	7,835	499	1,267.76	2,179.67	5,296.40
zebra	8,160	800	1,190.39	2,107.92	3,011.51
bunny head	17,483	500	4,262.25	6,322.57	41,720.19

# Chapter 5

## Hybrid Rendering Based on Viewcell

### Dependent Textured LOD

#### 5.1 Introduction



In order to achieve an immersive visual effect during the VR navigation, rendering with photo-realistic scene images in high frame rate has been an ultimate goal of real-time rendering. In the traditional geometry-based rendering, very complex scenes often consist of numerous polygons that cannot be rendered at an acceptable frame rate even using a state-of-the-art hardware. Many techniques have been proposed in last decades on reducing the polygon count while preserving the visual realism of the complex scenes, including visibility culling, level-of-detail (LOD) modeling, and image-based rendering (IBR). Although IBR is capable of rendering complex scenes with photo-realistic images in the time that is independent of the scene complexity, it has been suffered from the static lighting, the limited viewing degree of freedom, and some losses of image quality due to gaps and holes. As a consequence, hybrid rendering that combines geometry- and image-based technique has become a viable alternative.

As a representation for an object or a region of the scene, several image-based or hybrid representations have been proposed. Shade et al. [54] described a paradigm in which regions

or objects could be represented by environment map, planar sprite, sprite with depth, layered depth image (LDI), and polygonal mesh, depending on their distances to the viewer. Although the scheme integrates several existing representations, each individual form has its own problems. For example, sprites in general have gap problem due to resolution mismatch, and have to be re-computed once the viewer is outside the safe-region. LDI can only be drawn using software rendering with splatting. Finally, transition between different representations may produce noticeable popping effects.

To reduce gap problems due to resolution mismatch and to improve the efficiency of pixel-based rendering, depth meshes are extracted from the sprite with depth based on depth variation. However, rubber-sheet artifacts between disjoint surfaces are often encountered, and re-projecting pixel coordinates back to 3D coordinates may result in precision problems. The depth mesh approach can be incorporated by space subdivision, in which, when navigating inside a cell, distant objects are rendered using depth meshes with textures while near objects are rendered by selected LOD models. With such approaches, the polygon count of a complex scene can be still high and, most importantly, the transition between LOD and depth mesh with texture will generally results in visually noticeable popping effects.

### **5.1.1 View-cell dependent Textured LOD Modeling**

Another more uniform representation is LOD modeling, which can be incorporated with texture mapping for recovering surface details. Although most view-independent LOD techniques claim their capability to preserve geometric shape, silhouette is explicitly preserved as part of geometry. Moreover, view-independent LOD modeling has no control over the silhouette during navigation. View-dependent LOD modeling, on the other hand, is able to preserve silhouette through proper screen error test. However, it has to deal with silhouette problems at run-time by maintaining a mesh of fine resolution along the silhouettes. Furthermore, view-dependent LOD modeling has not been popular in real-time applications because it doesn't fit well in the pipeline rendering stream. Silhouette clipping that incorporates LOD modeling and normal/texture map

needs to extract fine silhouettes at run-time, which is in general time consuming.

An alternative to view-dependent and view-independent LOD modeling is the so called *view-cell dependent LOD modeling*. View-cell dependent LOD modeling is basically a view-independent LOD modeling but certain features of view-dependent LOD modeling are taken into account by limiting the view within a view cell. For an object outside the view cell, polygons that are back-facing to the cell are culled away and the remaining polygons are simplified using information derived from the captured depth images viewed at the center of the view cell and its adjacent cells. With such a mesh simplification, in addition to sharp edges and corners, interior and exterior silhouette with respect to the cell can be well preserved. Moreover, at run time, the view-cell dependent LOD models can be easily rendered by texture mapping the captured images.

Differ to existing approaches, the representation of viewcell dependent textured LOD has the potential to render closer object with smaller artifacts and provides a unified representation for both nearby object and distant object. Nearby objects are represented by SVMesh or MVMesh dependent on the self-occluding-error test for restricting the hole problem due to self occlusion, while distant objects may be clustered together if clustered objects pass the self-occluding-error test. Such clustering performs an implicit visibility culling as well as saves storage from occluded cached images.

### **5.1.2 System overview**

A hybrid rendering scheme that aims to render complex scenes in a constant and high frame rate with only a little or an acceptable quality loss is presented in this chapter. To this end, view space is partitioned into cells to explore the locality of visibility, and for a view cell, each object outside the cell is represented by a LOD mesh together with textures that are derived with respect to the view cell. All these are done in a preprocessing. In contrast with IBR or depth mesh approach, the object-based LOD mesh derivation avoids hole problems due to occlusion among objects. In the meantime, to reduce hole problems due to self-occluding,

the LOD mesh is classified into either single-view LOD mesh (termed as SVMesh) or multi-view LOD mesh (termed as MVMesh), depending on the object's self-occluding error (w.r.t. the viewcell). The SVMesh is chosen if the object's self-occluding error is smaller than a user-specified tolerance, otherwise MVMesh is chosen. Such a condition on SVMesh ensures that the potential holes possibly found in the images viewed from any point inside the cell will have size less than the user-specified tolerance. Hence all the information necessary to guide the derivation of SVMesh and the texture associated with the SVMesh come from the captured image and captured depth image of the cell's center. On the other hand, the MVMesh presents geometry and texture necessary to avoid holes on images viewed from some points in the cell. Therefore, the derivation of MVMesh and its texture associations are based on captured images and depth images from the cell's center as well as the centers of adjacent cells. In the proposed scheme, prefetching is also implemented to preload the data necessary for the following cells such that sudden drops in the frame rate at the cell transition can be avoided .

The proposed approach explores locality of visibility at the cost of extra storage and prefetching, and makes a tradeoff between image quality and rendering efficiency by using the SVMesh and MVMesh together with textures. Our experiments have shown that for a scene of 8 million polygons we have achieved higher than 600 frames/s. with a little loss of image quality (average PSNR 37.34dB). The polygons and textures require about 1260MB hard disk storage and about 287MB run-time memory on average. With such high frame rates, the overhead of prefetching is hardly noticeable.

## 5.2 Hybrid Rendering Scheme

The proposed hybrid scheme consists of a preprocessing phase and a run-time phase. In the preprocessing phase, the  $x$ - $y$  plane of the given 3D scene is first partitioned into equal-sized hexagonal cells. Then for each cell, we derive object-based textured LOD meshes, called SVMesh or MVMesh, for each object outside the cell. Note that with object-based LOD meshes, the holes due to occlusion among objects can be avoided. Furthermore, substituting original meshes with

textured SVMeshes or MVMeshes allows us to make a tradeoff between image quality and rendering efficiency. The SVMesh is a LOD mesh associated with the object whose potential self-occluding error is within a user-specified tolerance. Such a constraint ensures that the potential holes found in the image of an SVMesh viewed from any point inside the cell will have size less than the user-specified tolerance. The MVMesh will be associated with objects who fail to pass the self-occluding-error test. Before deriving SVMesh, those objects legitimate to SVMesh are tested for a possible clustering operation. Such an operation clusters those objects whose union is still legitimate to SVMesh and possesses a reduced texture size. After SVMesh or MVMesh is derived for each object outside the cell, an optional cell-based occlusion culling can be performed to further reduce the polygon count.

Both the SVMesh and MVMesh are derived from object's original meshes, with emphasis on preserving interior and exterior silhouettes. SVMesh is derived from polygons in original mesh that are front-facing to the cell's center while MVMesh comes from polygons that are front-facing to the whole cell. Moreover, they also differ in how the vertex's weights are derived for mesh simplification using edge collapsing and how textures are associated with simplified polygons. For SVMesh, the weight associated with each polygon vertex and the texture associated with each simplified polygon are derived only from the object's depth image viewed from the cell's center. On the other hand, for MVMesh, the derivation of vertex's weight and polygon's textures also takes into account the depth images viewed from centers of nearby cells.

At run-time phase, window culling and view-frustum culling are performed for the whole scene, followed by a back-facing culling for all objects inside the current navigation cell and a run-time occlusion culling for all meshes. SVMeshes and MVMeshes with associated textures are then texture mapped by hardware-accelerated projective texture mapping and meshes inside the cell are rendered as normal. To reduce the overhead of loading data from secondary storage when navigating across the cell boundary, a prefetching mechanism is applied to amortize the loading to previous frames.

## 5.2.1 Preprocessing phase

The steps in the preprocessing phase are (see Figure 5.1):

1. Hexagonal spatial subdivision.
2. For each cell, for each object outside the cell:
  - (a) perform regional conservative back-face test;
  - (b) perform self-occluding-error test and select single-view LOD mesh (SVMesh) or multi-view LOD mesh (MVMesh);
  - (c) derive SVMesh or MVMesh and texture(s) association;
  - (d) perform regional conservative back-face culling.
3. (Optional) Perform regional conservative occlusion culling.

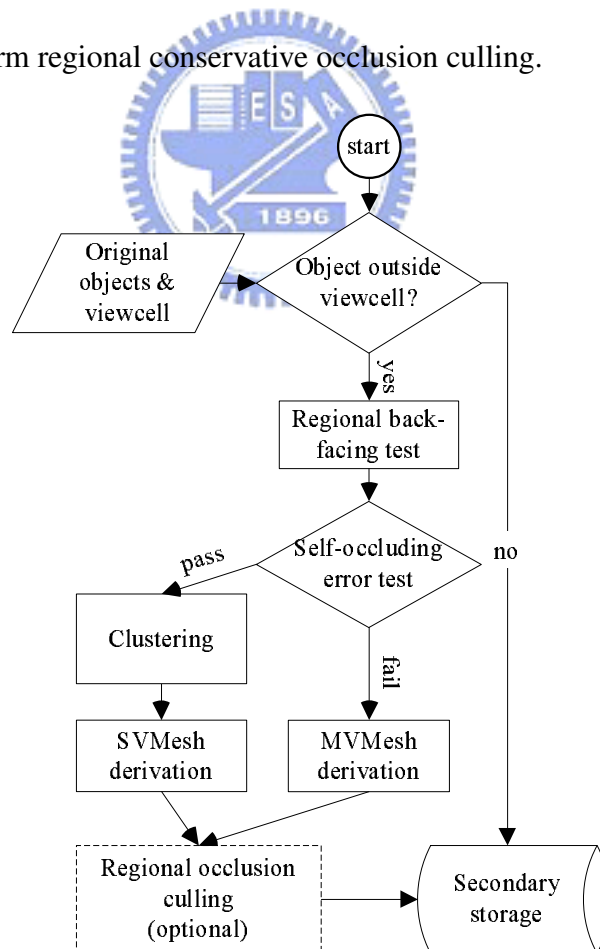


Figure 5.1: Preprocessing.



## Hexagonal spatial subdivision

In order to utilize the spatial locality of visibility, we subdivide the  $x$ - $y$  plane of the scene into  $N \times M$  hexagonal cells. With the spatial subdivision, the viewpoint can be localized to cells, and, therefore, cell-based visibility culling, back-facing and occlusion culling can be performed in the preprocessing phase. Compared to four for rectangular subdivision, hexagonal subdivision requires that data of only three adjacent cells need to be loaded when navigating across the cell boundary. Table 5.1 depicts the maximum ratio of side faces that can be seen from a point inside the hexagonal or the rectangular cell under different field of views (FOVs). We can see that hexagonal subdivision is better than rectangular one in most cases, except that they are equal for the  $45^\circ$ .

Table 5.1: Maximum ratio of side faces seen from a point inside the cell under different FOVs.

FOV( $^\circ$ )	120	90	60	45	30
Hexagonal	5/6	2/3	1/2	1/2	1/3
Rectangular	4/4	3/4	3/4	1/2	1/2

## Self-occluding-error test

Since the SVMesh of an object represents only those polygons that are front-facing to the cell's center, the images derived from SVMesh for views other than the cell's center may have holes due to the self-occlusion. Here we describe a conservative estimation of self-occluding error.

As shown in Figure 5.2, the maximum error occurs at the farthest view position  $V'$  from the cell center  $V$ . Let the cell size, i.e., the length of  $\overline{VV'}$ , be  $c$ , the distance between object and the cell center, i.e., the length of  $\overline{VO}$ , be  $d$ , and the depth of the object itself, i.e., the length of  $\overline{OP}$ , be  $l$ . The length of  $\overline{OC}$  is  $l \tan \theta$ , the angle  $\theta$  between  $\overline{VP}$  and  $\overline{V'P}$  is  $\theta = \tan^{-1} \frac{c}{d+l}$ , and  $s$ , the projected size of  $\overline{OP}$  or  $\overline{OC}$ , is

$$s = \frac{\overline{AB}}{c} \text{ImageRes.}$$

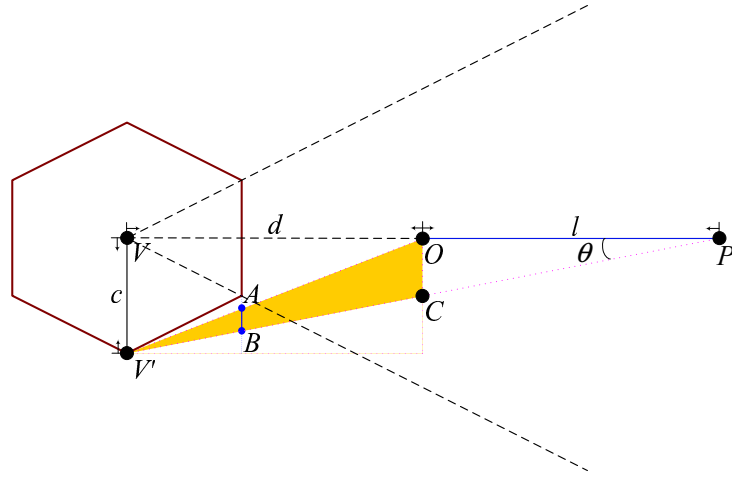
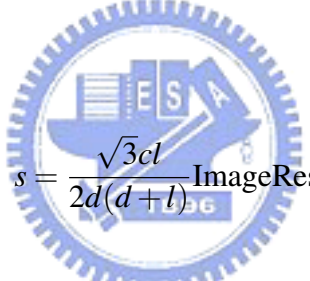


Figure 5.2: The maximum self-occluding error occurs at the position  $V'$ .

Since

$$\overline{AB} = \frac{\frac{\sqrt{3}}{2}c}{d} \overline{OC} = \frac{\frac{\sqrt{3}}{2}c}{d} \frac{cl}{d+l} = \frac{\sqrt{3}c^2l}{2d(d+l)},$$

we have,



$$s = \frac{\sqrt{3}cl}{2d(d+l)} \text{ImageRes.}$$

The self-occluding error of an object  $O$ , denoted as  $\text{self-occluding-error}(O)$ , is approximated by  $s$ , derived based on those polygons that are front-facing w.r.t. the cell. The self-occluding-error test is to check if  $s$  is smaller than a predefined tolerance  $T_s$  specified in image resolution. If it is, the object is represented by an SVMesh; otherwise by an MVMesh.

### SVMesh derivation

SVMesh intends to provide a textured LOD model for the portions of an object that is front-facing to the cell's center. The SVMesh is derived by simplifying the object using edge collapsing. The vertices are associated with weights derived from the depth variation found on the object's depth image captured at the cell's center. The cost of collapsing an edge is defined as a function of vertex's weights as well as the local geometry. The weight assignment is designed to distinguish important geometric features such as exterior silhouettes, interior silhouettes, and

sharp edges such that those features can be preserved according to their importance during the simplification.

The derivation of the SVMesh of an object  $O$  with respect to a cell  $C$  is outlined as follows.

1. Capture the image and depth image of  $O$  using cell's face as the window and cell's center as the center of projection.
2. Categorize pixels on the depth image as *exterior silhouette*, *interior silhouette*, *sharp edge*, and *interior*, and assign each category a weight.
3. Assign weights to object's vertices:
  - vertices that are back-facing with respect to the center of  $C$ : vertex weight is 0.5.
  - other vertices: vertex weight is the weight of the pixel gets projected by the vertex;
4. Perform edge collapsing in increasing order of edges' cost.

Figure 5.3(a) presents the flowchart for the derivation of SVMesh. Figure 5.4 depicts the SVMeshes of a bunny model.



### **Categorizing pixels on the depth image**

Pixels on the depth image are categorized into four categories:

- *Exterior silhouette*: a pixel on the external silhouette, which can be extracted using *contour extraction* techniques.
- *Interior silhouette* ( $C^0$ -discontinuity): a pixel  $Z$  whose value differs from adjacent pixels over a user-specified tolerance  $T_{C^0}$ ; that is,  $Z_{i+1} - Z_i > T_{C^0}$  or  $Z_{i-1} - Z_i > T_{C^0}$  (see Figure 5.5).
- *Sharp edge* ( $C^1$ -discontinuity): a pixel whose  $Z$  variation differs from  $Z$  variation of an adjacent pixel over a user-specified tolerance  $T_{C^1}$ ; that is,  $|(Z_{i-1} - Z_i) - (Z_i - Z_{i+1})| > T_{C^1}$  (see Figure 5.5).

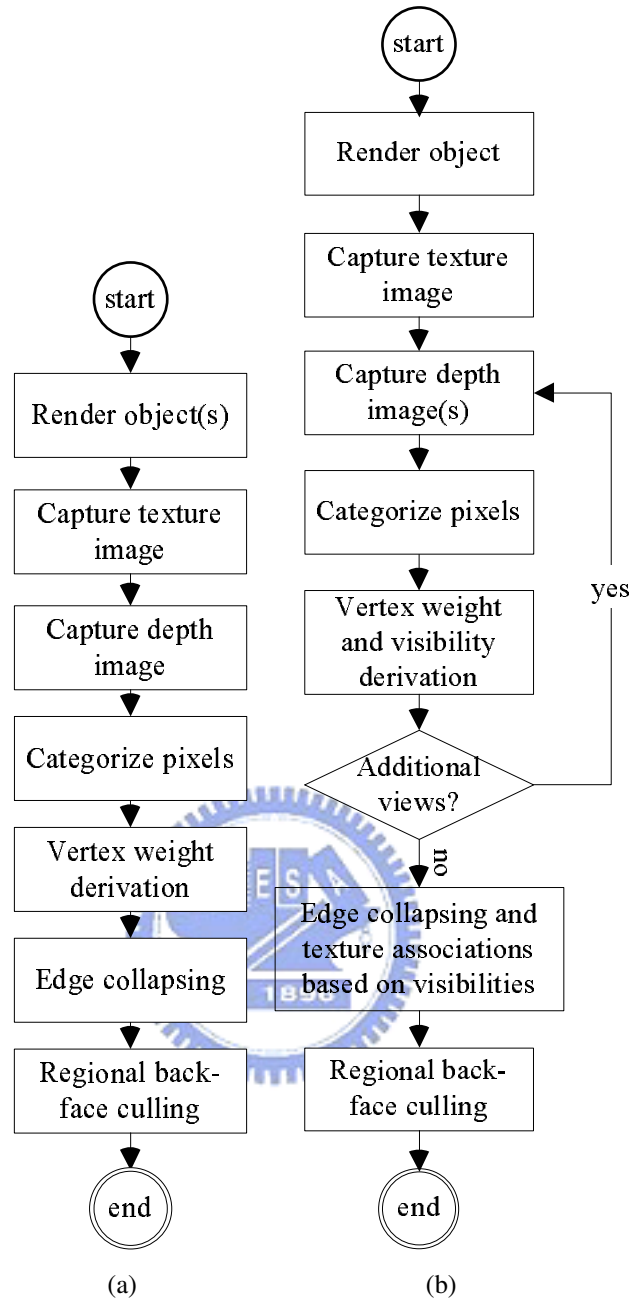


Figure 5.3: The derivations of SVMesh (a) and MVMesh (b).

- *Interior*: other pixels whose  $Z$  values are different from the background  $Z$  value.

Each category corresponds to a weight. We have derived from our experience that 0.5 is for *exterior silhouette*, 0.4 for *interior silhouette*, 0.25 for *sharp edge*, and 0.125 for *interior*.

### Assigning vertex weights

The vertex weight indicates how important the vertex is, which is usually determined by the

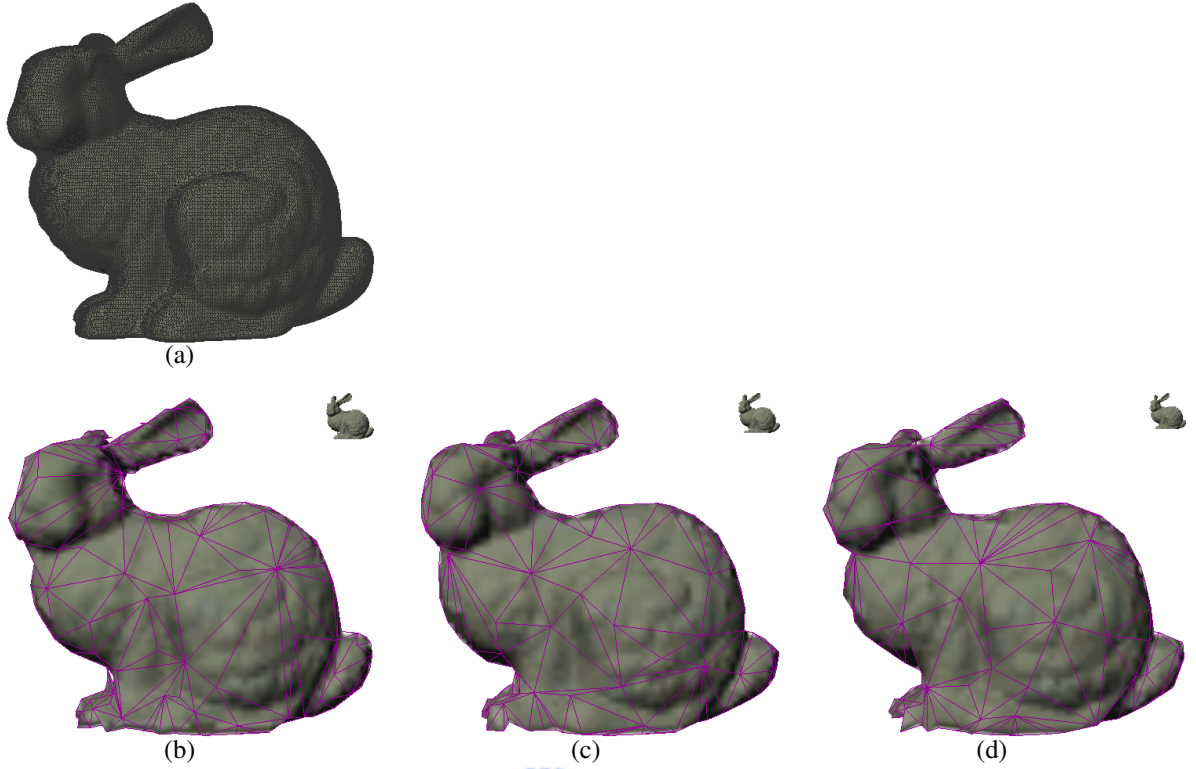


Figure 5.4: (a) is the original mesh (65,491 polygons) of a bunny viewed at one cell away (cell size 50), and (b-d) are SVMeshes for the bunny at 7 (259 polygons), 8 (254), and 9 (239) cells away. The upper-right bunnies are the projected images.

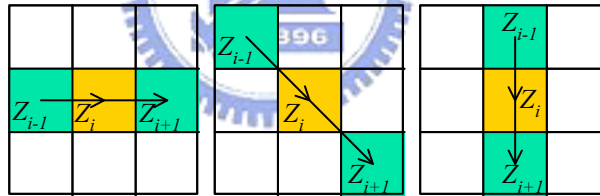


Figure 5.5: Testing depth variation.

local geometry and the viewing parameters. Here we propagate the weight derived for pixels on the depth image to corresponding vertices. We first distinguish back-facing and front-facing vertices. A vertex is back-facing (w.r.t. the cell's center) if all polygons incident to it are back-facing (w.r.t. the cell's center), otherwise it is front-facing. Each back-facing vertex is assigned with the weight 0.5 (same as that for the exterior silhouette vertex). For a front-facing vertex, we do the projection and check to see if it is visible to the cell's center by checking its Z-value against the Z-value of the pixel that gets projected. If it is, the pixel's weight is the weight of the vertex, otherwise it is invisible and assigned with the weight 0.05, which is smaller than vertices corresponding to the pixel category *interior*.

## Edge collapsing

To perform edge collapsing [35], the cost of collapsing an edge  $(v_i, v_j)$  is defined as

$$\text{cost}(v_i, v_j) = (1.5 - n_i \cdot n_j)^2 l (w_i + w_j),$$

where  $n_i$  and  $n_j$  are normals of  $v_i$  and  $v_j$ , respectively,  $l$  is the edge's projected length with respect to the cell's center, and  $w_i$  and  $w_j$  are the weights of  $v_i$  and  $v_j$ , respectively.

Edges are first maintained in an increasing order according to their costs, and stored in a heap. In each edge collapsing, the edge at top of the heap is removed and the vertex of smaller weight gets collapsed to the other. Such an collapsing order ensures that the edge with smaller cost gets collapsed first. The costs of some edges may be altered as a result of an edge collapsing, and must be updated afterwards. The edge collapsing is repeated until the edge on the top of the heap has cost higher than a user-specified value  $T_l$ , where  $T_l$  is a tolerance on the edge's projected length w.r.t. the cell's center.



## MVMesh derivation

The derivation of the MVMesh is an extension of that for SVMesh; as shown in Figure 5.3(b) [4]. For MVMesh, we consider those polygons that are front-facing with respect to the cell, rather than cell's center. Furthermore, the derivation of the vertex's weight takes into account the captured depth images viewed at the centers of the cell and its adjacent cells. For each vertex, a weight is obtained from each depth image as we do for the SVMesh and the vertex is assigned with the maximum of all those weights.

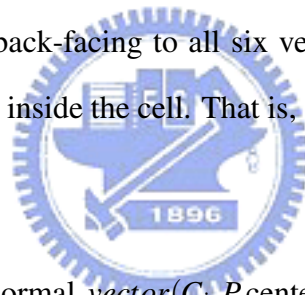
In addition to the weight, each vertex is also associated with a set of views to which the vertex is visible. The views associated with a simplified polygon is determined by the intersection of view sets associated with the polygon's vertices. Since the views associated with a vertex cannot propagate in the course of edge collapsing, we place one more condition on edge collapsing. Namely, for an edge  $\overline{uv}$ ,  $u$  can be collapsed to  $v$  if the weight of  $u$  is smaller than or equal to that

of  $v$  and both  $u$  and  $v$  are either visible to some common views or associated with empty view sets. Note that determining polygon's set of views based on that of vertices is not able to reflect the cases in which the polygon is partially occluded, but its vertices are not, by other polygons. Such exceptions should be handled carefully by considering general visibility problems.

For the cost function of an edge, we should replace  $l$ , the projected length of an edge with respect to the cell's center, by  $l'$ , which is the projected length of the edge with respect to the cell. When the object is far from the cell, we have  $l \approx l'$ . The edge's projected length for a near object, however, varies when we navigate in the cell. Figure 5.6 depicts the MVMeshes of the bunny model.

### Regional conservative back-face culling

We claim that if a polygon is back-facing to all six vertices of the cell, the polygon is back-facing with respect to any point inside the cell. That is, a polygon  $P$  is back-facing with respect to the cell  $C$  if



$$\text{dot\_product}(P.\text{normal}, \text{vector}(C_i, P.\text{center})) < 0, \text{ for } i = 0, \dots, 5,$$

where  $C_i$ 's are the corners of  $C$ . A simple proof for the 2D case is as follows: If a polygon  $P$  is back-facing with respect to both  $A$  and  $B$ ,  $P$ 's normal will be bounded in the dark green area, as shown in Figure 5.7(a). Given a point  $G$  on the line  $\overline{AB}$ , vector  $\overrightarrow{GP}$  is bounded by  $\overrightarrow{AP}$  and  $\overrightarrow{BP}$ . As a result,  $P$  is shown to be back-facing with respect to  $G$ . An interior point  $I$  of the cell  $C$  is on a line  $\overline{C_i E}$ , for some  $i$ , and  $E$  on  $\overline{C_j C_{(j+1) \bmod 6}}$  for some  $j$ . Since  $P$  is back-facing with respect to all corners,  $P$  is back-facing with respect to  $E$  and therefore  $I$ ; see Figure 5.7(b).



Figure 5.6: (a) is the original mesh (65,491 polygons) of a bunny viewed at one cell away (cell size 50), (b-g) are MVMeshes of the bunny at 1 (1,605 polygons), 2 (945), 3 (554), 4 (392), 5 (330), 6 (306) cells away. The upper-right indicates actual projected images.



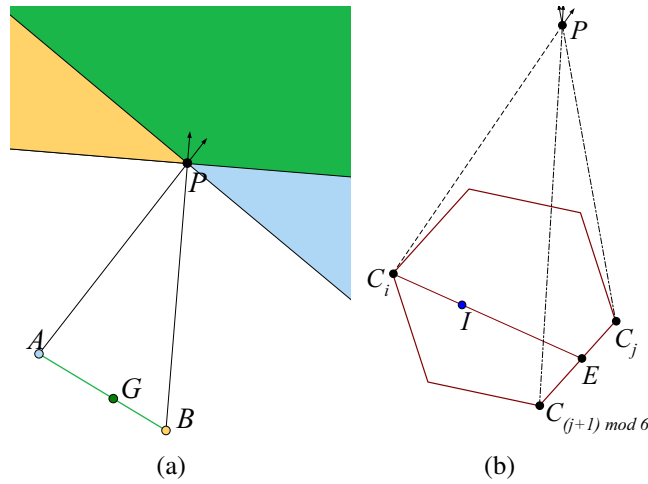


Figure 5.7: Regional back-face culling.

## Object clustering

In order to reduce the texture size associated with LOD meshes and to reduce polygon count, objects that pass the self-occluding-error test and are close to each other can be clustered together, provided that certain conditions are satisfied. The clustering operation amounts to the coloring problem, and itself is an NP-complete problem. Before getting into the details of the proposed greedy approach, several terms are first described.

- *Cluster-able*: Object or cluster  $M$  is cluster-able with cluster  $C$  if the texture size of  $M \cup C$  is less than the sum of texture sizes of  $M$  and  $C$ , and  $\text{self-occluding-error}(M \cup C) < T_s$ .
- *Overlapping size*: Overlapping size of an object  $M$  and a cluster  $C$  is the size of the intersection of projected areas of  $M$  and  $C$ .

The greedy approach proceeds as follows. Firstly, objects that pass self-occluding-error test are sorted according to the size of their projected areas. Initially no cluster is formed. Secondly, for each object  $M$  removed from the sorted list,  $M$  itself forms a new cluster if there is no cluster or no cluster found to be cluster-able with  $M$ . Otherwise,  $M$  is repeatedly clustered with all the clusters that  $M$  is cluster-able with, in the order of decreasing overlapping size. As shown in Figure 5.8(a),  $M$  is cluster-able with  $C_1$ ,  $C_2$  and  $C_3$  in the order of decreasing overlapping size.  $M$  is clustered with  $C_1$  first. The result  $M \cup C_1$  is, however, is no longer cluster-able with  $C_2$ ; but still cluster-able with  $C_3$ ; see Figure 5.8(b). Finally,  $M$  is clustered with  $C_1$  and  $C_3$ ; see

Figure 5.8(c).

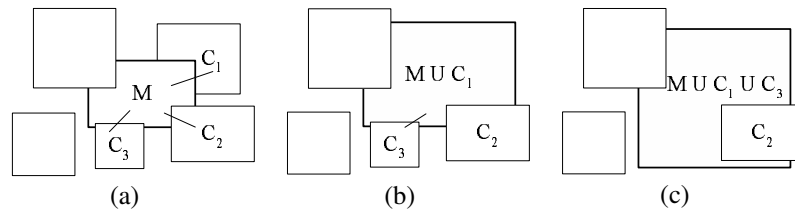


Figure 5.8: Repeat clustering.

The clustering is performed after the self-occluding-error test is applied for all objects, and before the derivation of SVMesh. The objects in the same cluster are considered as a single object that possesses an SVMesh. The SVMesh derivation can be slightly modified to construct an SVMesh for the clustered objects. In consequence, surfaces that are occluded by others in the cluster will be culled out in the simplification process. Such an SVMesh derivation for clustered objects implicitly performs occlusion culling among objects.

### Regional conservative occlusion culling

Since SVMesh or MVMesh is object based and our scheme does space subdivision for utilizing view locality, it will be advantageous to do the regional conservative occlusion culling in the preprocessing phase. Such operations will enhance the rendering efficiency, especially for densely occluded scenes. Methods proposed recently can be used. For example, the extended projection [24] can be easily modified to fit into our system. This extended projection can also handle the case of multiple occluders by using occluder fusion. The selection of occluders is based on the meshes' projected sizes. Only those meshes whose projected sizes are larger than a user-specified threshold are selected to be occluders.

## 5.2.2 Run-time phase

At the run-time phase, within the current navigation cell we first set up a lower priority thread for prefetching the geometry and image data belonging to neighboring cells, and then do the following steps when navigating inside the cell: (see also the flowchart in Figure 5.9.)

1. Ensure that the geometry and image data for the current navigation cell has been loaded into memory.
2. Perform window culling and view-frustum culling for the whole scene.
3. (Optional) Perform a run-time occlusion culling for all meshes.
4. (Optional) Perform a run-time back-face culling for the meshes inside the current cell.
5. Render the meshes outside the current cell using *projective texture mapping*, followed by rendering meshes inside the cell as normal.
6. Prefetch data for neighboring cells when the CPU load is relatively low.

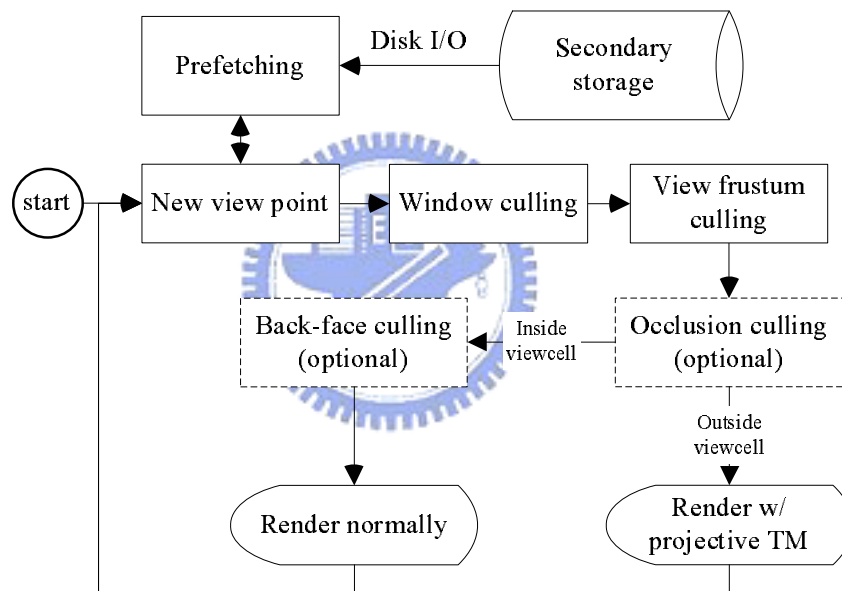


Figure 5.9: Run-time phase.

A view with an FOV sees through a fixed number of windows, which are faces of the navigating cell. Window culling can be considered as an effective pre-calculation of the view-frustum culling. As optional operations, the run-time back-face culling and occlusion culling can be applied to further reduce the polygon count. Back-facing culling is performed only for objects inside the navigation cell, while occlusion culling is applied to all meshes in the scene.

## Prefetching scheme

One of the major problems arises in our cell-based navigation is how to achieve smooth cell transition. When the view point moves across from one cell to its neighbors, the geometry and textures will be switched. The prefetching is a mechanism to preload the geometry and texture data of neighboring cells when CPU load is relative low during navigating inside the cell. It will amortize the loading time to several inside-cell frames and hence reduce the FPS gap between inside-cell frames and a cross-boundary frame.

The proposed scheme runs in a lower priority thread and prefetches data of neighboring cells that will be possibly visited in a short time. We set a *timestamp* for the navigation system. The timestamp is initially 0 and gets increased by 1 whenever the viewer have moved by a predefined distance or have turned by a predefined angle. When the timestamp gets increased by 1, we identify those cells that need to be prefetched and add them to a priority queue that maintains those cells waiting for prefetching, and then begin the prefetching. The cell is added to the priority queue according to its *t-priority*, which is the sum of the current timestamp and a priority value. The addition of timestamp in the *t-priority* allows us to distinguish the freshness of the cells in the priority queue. The priority value of a cell conceptually indicates how urgent it is for prefetching and is in the range of  $[0, 1)$ . In principle, cells that are within the view frustum get higher priorities than those outside, the cells closer to the aiming vector get higher priorities, and cells closer to the viewpoint get higher priorities. The prefetching repeatedly removes the cell of the highest *t-priority* from the priority queue and loads the cell's data from disk to main memory, and removes those cells which are out of date by checking if *t-priority* values are smaller than the current timestamp minus 2.

Disk I/O can run in parallel; however, the system bus that loads texture data from main memory to texture memory can hardly run in parallel. To this end, textures that have been loaded from disk to main memory are put into a *texture queue* and get loaded in FIFO order. The loading of texture from main memory to texture memory runs in main thread, in which an amount of texture constrained by a budget is loaded before each frame. One practical concern is that the

size of texture varies a lot. A texture that is not the first in the texture queue and is of size larger than the remaining budget is put back as the first in the texture queue.

### **Rendering of object-based LOD mesh w/ textures**

After cell data is loaded the system memory, SVMesh and MVMesh are rendered by mapping the cached images as textures using projective texture mapping [51] or the proposed *projective-alike texture mapping*. The texture shifting introduced by casting the cached image onto the simplified mesh can be reduced by blending multiple cached images that are captured at adjacent view-cell centers, and the popping effects at view-cell boundary due to the change of representation can be minimized by interpolating the rendering results from their two different simplified meshes.

SVMeshes are simply rendered by projective texture mapping while the rendering of MVMeshes involves texture blending as part of view-dependent projective texture mapping [21]. As mentioned previously in MVMesh derivation, a simplified polygon is associated with a set of views to which it is visible. If the set is empty, normal map approach [7] can be applied to that polygon. If the set contains only one view, then the polygon is rendered by standard projective texture mapping. If the set contains three or more views, two views are chosen from the set according to the vector defined from the viewer to the polygon. The textures corresponding to these two views are then mapped onto the polygon using projective texture mapping with blending.

Alternatively, we can apply the proposed *projective-alike texture mapping* to map the cached images onto SVMesh or MVMesh in such a way that the coordinate of each vertex is specified in image space and the texture coordinate of each vertex is generated automatically using graphics hardware. The re-projection from the source image coordinate to the destination image coordinate can be done by a transformation matrix

$$T_2 T_1^{-1},$$

where  $T_2$  is the destination camera matrix and  $T_1^{-1}$  is the inverse of the camera matrix of the source image. Figure 5.10) depicts the re-projection.

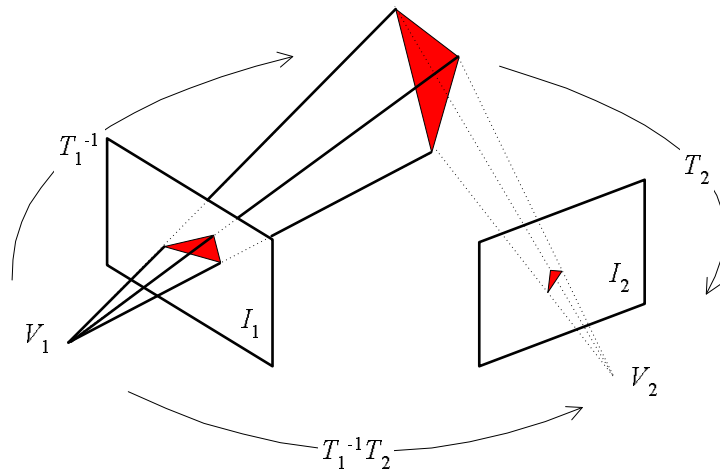


Figure 5.10: Re-projection from source to destination image,  $T_1$  and  $T_2$  are the camera matrix of source image  $I_1$  and destination image  $I_2$ , respectively.

In most cases, an object-based LOD mesh located in part of a source image (e.g. see Figure 5.11). To minimize the storage requirement for cached images, only necessary rectangular part of the source image is stored as a cached image.

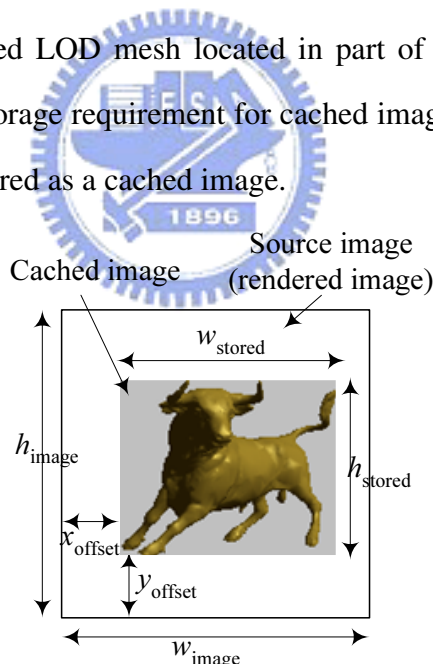


Figure 5.11: The cached image is a part of a source image.

The texture coordinates  $(s, t)$  necessary for projective-like texture mapping can be automatically generated from image coordinate  $(x, y)$  using graphics hardware (e.g. by the standard OpenGL API `glTexGen()`). The derivation is

$$(s, t) = ((x - x_{\text{offset}})/w_{\text{stored}}, (y - y_{\text{offset}})/h_{\text{stored}}),$$

where  $x_{\text{offset}}$  and  $y_{\text{offset}}$  are the offset of the cached image relative to the source image, and  $w_{\text{stored}}$  and  $h_{\text{stored}}$  are width and height of the cached image, respectively. In consequence, we don't need additional memory to store texture coordinate for each vertex, and save the bandwidth needed between CPU and graphics accelerator. Moreover, vertices of the object-based LOD mesh are allowed to be stored in the source image coordinate space in integer precision, which requires 16-bit unsigned integer for  $x$  and  $y$ , and 32-bit floating point for  $z$  (provides a tradeoff between storage requirement and precision). As a result, only 8 bytes is necessarily sufficient for each vertex, compared to 20 bytes per vertex if all  $x$ ,  $y$ , and  $z$  are stored as world coordinates as well as texture coordinates  $s$  and  $t$ . While referencing the vertex through vertex ID requires only 4 bytes per vertex but requires its original mesh presented in the memory for reference, storing vertices in image-space coordinate may save more memory space for distant objects, and, moreover, poses less precision error problems for distant objects.

Since the texture maps are mapped onto the simplified meshes, there are some texture shifting artifacts on the nearby objects. Fig. 5.12 illustrates the problem. The color of point  $p$  on the simplified mesh can be  $c_1$  or  $c_2$ , depending on which textures, casted from the reference view  $V_1$  or  $V_2$ , is used. However, the correct color of  $p$  seen from the eye position should be  $c$ . It is obvious that one can select the reference view that forms the smallest angle with the eye direction. Alternatively, blending multiple texture values results in less popping effects when switching the textures. The blending equation for 2-D cases is

$$c' = c_1 \frac{\alpha_2}{\alpha_1 + \alpha_2} + c_2 \frac{\alpha_1}{\alpha_1 + \alpha_2}.$$

For a 3-D scene, we can use the method proposed by [22] in which blending weights are assigned to three textures.

An object in general possess different simplified meshes for adjacent view cells. As a result, popping effects occur during the transition between view cells. We illustrate this effect in Fig. 5.13, where the eye position is at the cell boundary and  $M_1$  and  $M_2$  represent simplified meshes of the same object in two adjacent cells. The object is represented by  $M_1$  when the eye

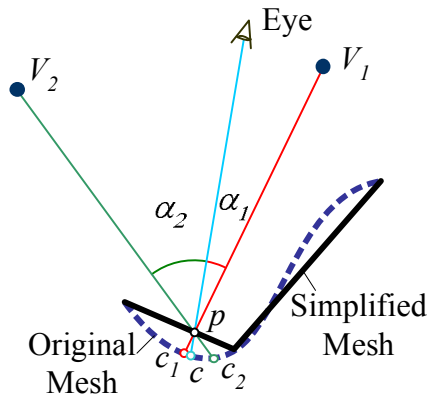


Figure 5.12: The smaller angle is, the more accurate result is.

position moves to the left, and by  $M_2$  when it moves to the right. The viewing ray intersects  $M_1$  and  $M_2$  at points  $p_1$  and  $p_2$ , respectively. Apparently, the blended texture values of  $p_1$  and  $p_2$  may be different. In fact, only objects that are very close to a reference view, such as  $V_1$  or  $V_2$ , would produce noticeable popping effects. For such an object, we may interpolate the rendering results from its simplified meshes but at about double cost on rendering.

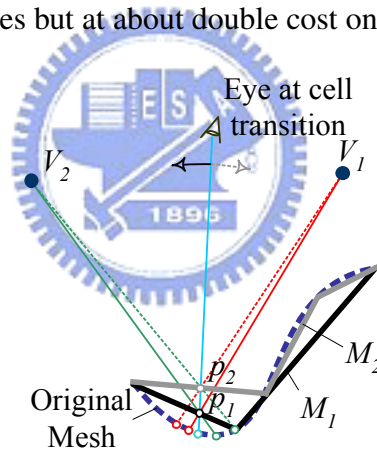


Figure 5.13: Popping effects occur during the transition between view cells.

## 5.3 Experimental Results

### 5.3.1 Setup

The test platform is a PC with Pentium4 in 3.0Ghz CPU, 2GB main memory, and an nVIDIA GeForce 6800GT with 256MB video memory graphics accelerator. The OS is Windows XP Pro SP2. The output image is in a resolution of  $1024 \times 1024 \times 32$ . S3's S3TC DXT3 is used to



compress textures (in a ratio of 1/4).

For efficiency consideration, polygons and objects are represented by vertex IDs and object IDs, respectively. The original meshes are loaded into main memory before the navigation. In prefetching objects, SVMeshes, and MVMeshes, only their object IDs and vertex IDs are loaded.

### Scene statistics

The three scenes tested are statuary parks consisting of eight kinds of object that are randomly distributed in the same area of  $1650 \times 2035$ . The three scenes are called 2M-scene, 4M-scene, and 8M-scene, and have 2017700, 4188885 and 8004863 polygons, respectively. The scenes are generated such that 2M-scene is a subset of the 4M-scene, which in turn is a subset of 8M-scene. Table 5.2 lists data statistics for the objects that compose the scenes, including polygon number, dimension, and distribution of polygon numbers for the scenes. A bird’s eye view of the 8M-scene is shown in Figure 5.14.

Table 5.2: Object and scene statistics.

Object name	Polygon no.	Dimension (w×d×h)	2M	4M	8M
dragon	202,520	$57.3 \times 25.6 \times 40.4$	4	10	18
bunny	69,451	$43.6 \times 33.8 \times 43.2$	7	12	26
statue	35,280	$11.8 \times 13.4 \times 23.4$	13	21	40
cattle	12,398	$40.0 \times 40.8 \times 30.7$	9	19	42
horse	7,257	$38.3 \times 57.2 \times 82.6$	13	29	51
easter	4,976	$12.4 \times 10.7 \times 30.8$	6	14	22
camel	3,969	$49.4 \times 16.8 \times 46.6$	4	14	26
venus	1,396	$10.2 \times 8.4 \times 21.9$	8	13	28
Total object number			64	132	253

### Settings

Performance on frame rate and image quality may vary for different settings of parameters. We set  $T_s = 3, 5, \text{ or } 7$  pixels for self-occluding-error tolerance,  $T_l = 3.0, 4.5, \text{ or } 6.0$  for edge’s project length tolerance, and 50 or 100 for cell size. The parameters  $T_{C0}$  and  $T_{C1}$  for pixel categorizing

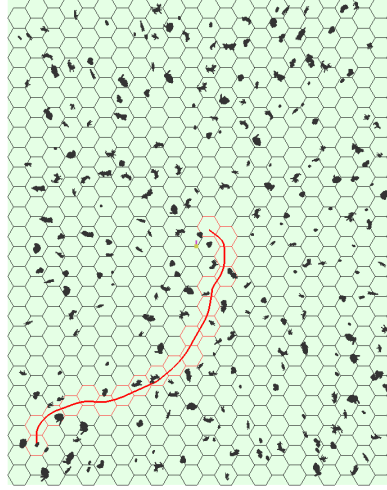
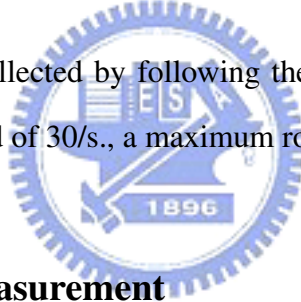


Figure 5.14: Bird's eye view of the 8M-scene.

are fixed in this experiment as  $3.4 \times 10^{-4}$  and  $1.28 \times 10^{-4}$ , respectively. For simplicity, we denote the  $k$ M-scene with cell size  $c$ , parameters  $T_s$  and  $T_l$  as  $kM-c-T_s-T_l$ ; for example, the 4M-scene with cell size 50,  $T_s = 5$ , and  $T_l = 4.5$  is denoted as 4M-50-5-4.5.

All experimental results are collected by following the navigation path shown in red in Figure 5.14 with a maximum speed of 30/s., a maximum rotation of  $45^\circ/s.$ , and an FOV of  $60^\circ$ .



### 5.3.2 Image quality measurement

To identify how much is the quality-loss, we use the peak signal-to-noise ratio PSNR(dB) defined as

$$\text{PSNR} = 10 \log_{10} \frac{255^2}{\frac{1}{HW} \sum_{x=0}^{W-1} \sum_{y=0}^{H-1} [\hat{f}(x,y) - f(x,y)]^2},$$

where  $f(x,y)$  and  $\hat{f}(x,y)$  are the pixel colors of the original image and approximated image at position  $(x,y)$ , respectively,  $W$  and  $H$  are the dimensions of the image. Before applying PSNR, the RGB color is mapped to a single luminance value  $Y$  since human eyes are more sensitive to the changes in luminance. Such a mapping [31] is

$$Y = 0.299 * R + 0.587 * G + 0.114 * B.$$

### 5.3.3 Mesh simplification

#### Self-occluding-error tolerance

The value of self-occluding-error tolerance  $T_s$  determines the distribution of SVMesh and MVMesh. For the scene 4M-50- $T_s$ -4.5, where  $T_s = 3, 5, 7$ , Table 5.3 shows the averaged percentages of objects that are represented by SVMesh and MVMesh and their averaged polygon counts over all cells. Larger  $T_s$  implies higher percentage of SVMesh, more objects are clustered, higher simplification rate, less texture size, and finally higher frame rate. Note that numbers in the parenthesis under Avg. polygon count inside view frustum are Avg. polygon count for SVMesh and MVMesh inside view frustum. Figure 5.15 depicts the distribution of SVMesh and MVMesh for the particular cell at the scene's center, on which MVMeshes are colored in blue, SVMeshes from single objects are in purple, and SVMeshes from clustered objects are in other colors.

Table 5.3: Simplification performance under different self-occluding-error tolerance  $T_s$ .

4M-50- $T_s$ -4.5	$T_s = 3$	$T_s = 5$	$T_s = 7$
Statistics for object's representations and polygon counts			
Avg. percentage of SVMeshes from clustered objects (%)	9.8	18.7	25.5
Avg. percentage of SVMeshes from a single object (%)	64.8	64.9	62.4
Avg. percentage of MVMeshes (%)	25.3	16.3	12.1
Avg. polygon no. inside a viewcell	9,308	9,308	9,308
Avg. polygon no. for SVMesh & MVMesh	40,742	39,981	39,360
Avg. polygon no. for a viewcell	50,050	49,290	48,669
Simplified : original	1 : 83.7	1 : 85.0	1 : 86.1
Performance statistics			
Avg. FPS	1114.8	1157.0	1182.4
Avg. PSNR (dB)	39.64	39.54	39.46
Avg. texture size (KB) inside view frustum	592.0	544.1	516.6
Avg. polygon count inside view frustum	13,235 (11,532)	13,102 (11,418)	13,026 (11,367)

#### Projected edge-length tolerance

Through projected edge-length tolerance  $T_l$ , the edge collapsing can be tested for termination. Figure 5.16 shows the MVMeshes of bunny derived by setting  $T_l = 3.0, 4.5, 6.0$ . Table 5.4 depicts the average polygon counts of SVMesh and MVMesh and simplification ratio for all cells. As we can see, larger  $T_l$  implies higher simplification rate, larger texture size, and finally

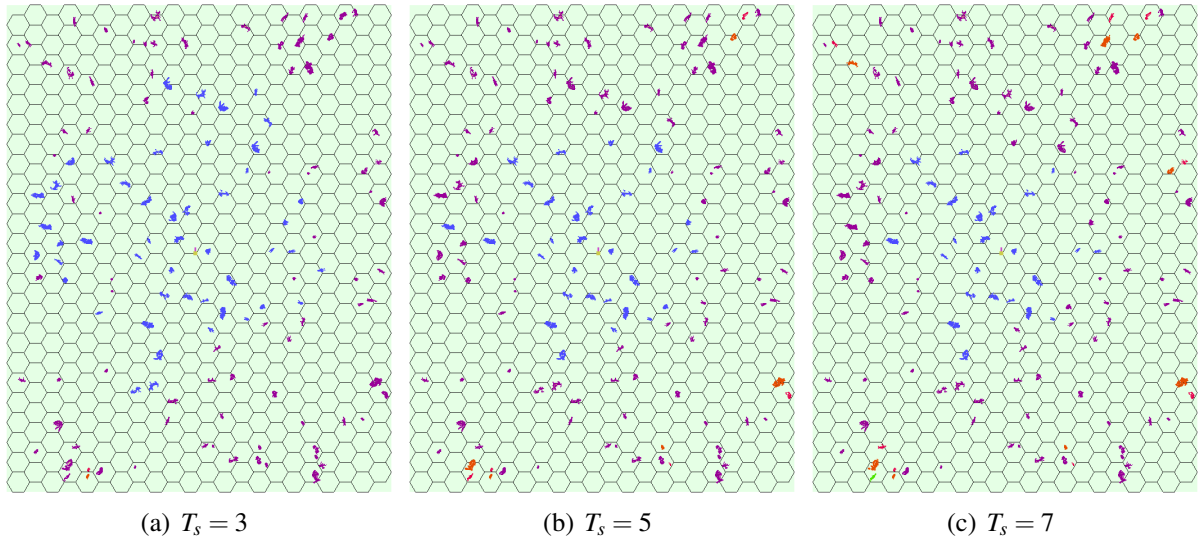


Figure 5.15: Distribution of SVMesh and MVMesh for the scenes 4M-50- $T_s$ -4.5.

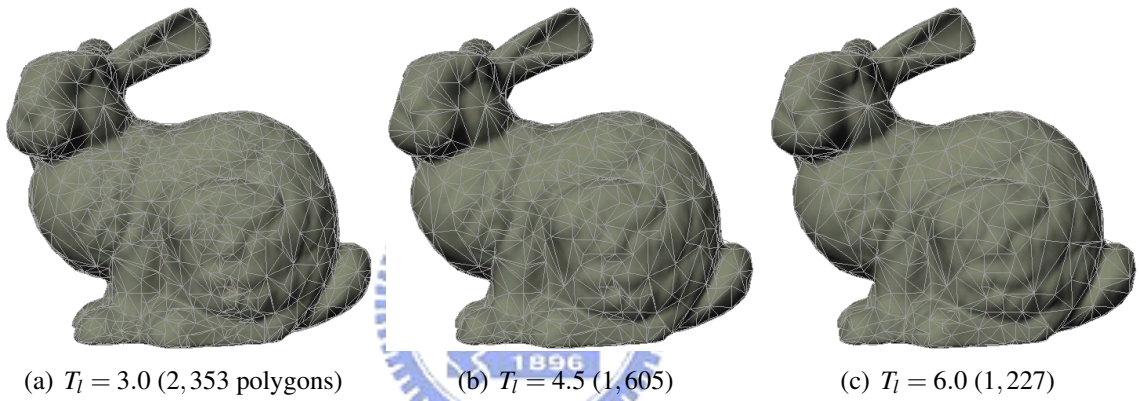


Figure 5.16: MVMeshes of bunny for different  $T_l$ .

higher frame rate.

### Cell size consideration

Setting an optimal cell size is in general difficult. To test the effect of cell size, we continue to use the same data set and set  $T_s = 5$  and  $T_l = 4.5$  for cell sizes 50 and 100. Table 5.5 depicts the average polygon counts of SVMesh and MVMesh and simplification ratio for all cells. Larger cell size in general results in smaller simplification ratio and, in turns, lower frame rate, since the number of polygons inside a cell may increase dramatically.

Table 5.4: Simplification performance under different projected edge-length tolerance  $T_l$ .

4M-50-5- $T_l$	$T_l = 3.0$	$T_l = 4.5$	$T_l = 6.0$
Statistics for polygon counts			
Avg. polygon no. inside a viewcell	9,308	9,308	9,308
Avg. polygon no. for SVMesh & MVMesh	46,249	39,981	35,226
Avg. polygon no. for a viewcell	55,557	49,290	44,535
Simplified : original	1 : 75.4	1 : 85.0	1 : 94.1
Performance statistics			
Avg. FPS	1091.4	1157.0	1223.9
Avg. PSNR (dB)	39.98	39.54	39.13
Avg. texture size (KB) inside view frustum	538.4	544.1	545.4
Avg. polygon count inside view frustum	14,741 (13,008)	13,102 (11,418)	11,701 (10,048)

Table 5.5: 4M scene under different cell sizes 50 and 100.

Cell size	50	100
Viewcells	$22 \times 24$	$11 \times 12$
Statistics for polygon counts		
Avg. polygon no. inside a viewcell	9,308	38,986
Avg. polygon no. for SVMesh & MVMesh	39,981	45,356
Avg. polygon no. for a viewcell	49,290	84,342
Simplified : original	1 : 85.0	1 : 49.7
Performance statistics		
Avg. FPS	1157.0	1003.7
Avg. PSNR (dB)	39.54	39.62
Avg. texture size (KB) inside view frustum	544.1	387.9
Avg. polygon count inside view frustum	13,102 (11,418)	17,457 (12,319)

### 5.3.4 Run-time performance

The three rendering configurations used to test the performance comparison are:

- **A: (Pure geometry)** render the original scene geometry using the traditional graphics pipeline.
- **B: (Pure geometry with view frustum culling)** same as A, but with software view frustum culling.
- **C: (Proposed hybrid scheme)** render the scene using proposed hybrid scheme, without regional occlusion culling, run-time back-face culling, and run-time occlusion culling.

The parameter setting for the following performance tests is  $T_s = 5$ ,  $T_l = 4.5$ , and cell size 50.

All simulations follow the navigation path shown in Figure 5.14.

Table 5.6 lists the run-time performance of three configurations on the scene 8M-50-5-4.5. Without regional occlusion culling, back-face culling, and run-time occlusion culling, configuration **C** achieves 321.8 gain factor over configuration **A**, and 88.8 gain factor over configuration **B**, with little quality-loss at PSNR 37.34dB.

Table 5.6: Performance of the three configurations on a 8M-scene.

	<b>A</b>	<b>B</b>	<b>C</b>
Avg. polygon count	8,004,863	2,443,969	23,580
Avg. frame time (ms)	520.8	143.7	1.618
Avg. frame rate (FPS)	1.92	6.96	617.9
Speedup	1.00	3.63	321.8

Figure 5.17 and Figure 5.18 represent the images rendered at views that are far from the cell center by configuration **B** and **C**. In Figs. 5.17(c) and 5.18(c), the MVMeshes are flat shaded with gray wireframes, SVMeshes from single objects are in purple, and SVMeshes from clustered objects in other colors.

Table 5.7 depicts the performance of configuration **C** for different scene complexities 2M-50-5-4.5, 4M-50-5-4.5, and 8M-50-5-4.5. It reveals that as the scene complexity goes up from 2M, 4M, to 8M, the FPS goes down from 1816, 1157, to 618. This is due to the fact that all objects outside a navigation cell are in the form of SVMesh or MVMesh, which have much less varied polygon counts.

Table 5.7: Performance of configuration **C** under different scene complexities.

Scene complexity	2M	4M	8M
Statistics for polygon counts			
Avg. polygon no. inside a viewcell	4,145	9,308	18,302
Avg. polygon no. for SVMesh & MVMesh	18,829	39,981	75,891
Avg. polygon no. for a viewcell	22,974	49,290	94,193
Simplified : original	1 : 87.8	1 : 85.0	1 : 85.0
Performance statistics			
Avg. FPS	1815.7	1157.0	617.9
Avg. PSNR (dB)	44.92	39.54	37.34
Avg. texture size (KB) inside view frustum	112.4	544.1	992.4
Avg. polygon count inside view frustum	4,548 (3,991)	13,102 (11,418)	23,580 (21,735)

Figure 5.19 and Figure 5.20 show the run-time statistics of running configuration **C** on the scene 8M-50-5-4.5. In Figure 5.19, FPS plots are shown for different prefetching schemes. From the



(a) Configuration **B**: 5,207,350 polygons in view frustum.



(b) Configuration **C**: 35,216 polygons, PSNR 35.33dB, and 512.5 FPS.



(c) Configuration **C**: flat shaded with wireframes.

Figure 5.17: Rendered images by configuration **B** and **C**.



(a) Configuration **B**: 4,693,355 polygons in view frustum.



(b) Configuration **C**: 29,641 polygons, PSNR 36.39dB, and 563.2 FPS.



(c) Configuration **C**: flat shaded with wireframes.

Figure 5.18: Rendered images by configuration **B** and **C** at another view.



plot for prefetching under a cold cache, we can see that the frame rate changes rapidly after a cell transition (illustrated by yellow vertical line) and becomes more stable frame rate after a while. The frame rate for the prefetching under a warm cache is quite stable except some sudden decreases appear. The suddenly decreased FPS in the plots indicates the presence of objects inside the navigation cell. Along this particular navigation path, among 7 cells that contain objects, two of them contain the massive models such as bunny and dragon, respectively; as shown also in the plot for polygon count. Polygon count, texture size, and PSNR follow the FPS plots which are shown in Figure 5.20. Note that most frames have PSNR above 37dB.

### 5.3.5 Discussions

On problems and potential of the proposed hybrid rendering scheme, we address the preprocessing time, storage requirement and run-time loading time for very complex scenes.

Table 5.8 shows the preprocessing time for 2M-, 4M-, and 8M-scene. The most cost operations in the preprocessing are the derivation of viewcell dep. LOD meshes and the capture of textures.

Table 5.8: Preprocessing time for different scene complexities.

Scene complexity	2M	4M	8M
Time (hours)	4.45	9.52	17.48

Since each polygon and each object are presented by vertex IDs and object ID, respectively, original meshes must be available in main memory during navigation and meshes are prefetched by loading their vertex IDs. Original meshes of 2M-, 4M-, and 8M-scene account for 74, 141, and 261MB main memory, respectively. Taking into account the prefetched data as well, the main memory requirement is 82, 153, and 287MB for 2M-, 4M-, and 8M-scene, respectively.

As shown in Table 5.9, storage requirement for 2M, 4M, and 8M scenes is 311MB, 659MB, and 1,260MB, respectively, which are roughly 11 times that for original geometries. For each cell, the average size of polygons and textures that needs to be in memory is 553, 1165, and 2252KB for 2M-, 4M-, and 8M-scene, respectively. Let us take as an example the hard disks having reading speeds at 35MB/s. For the 8M-scene, the loading time for each cell requires

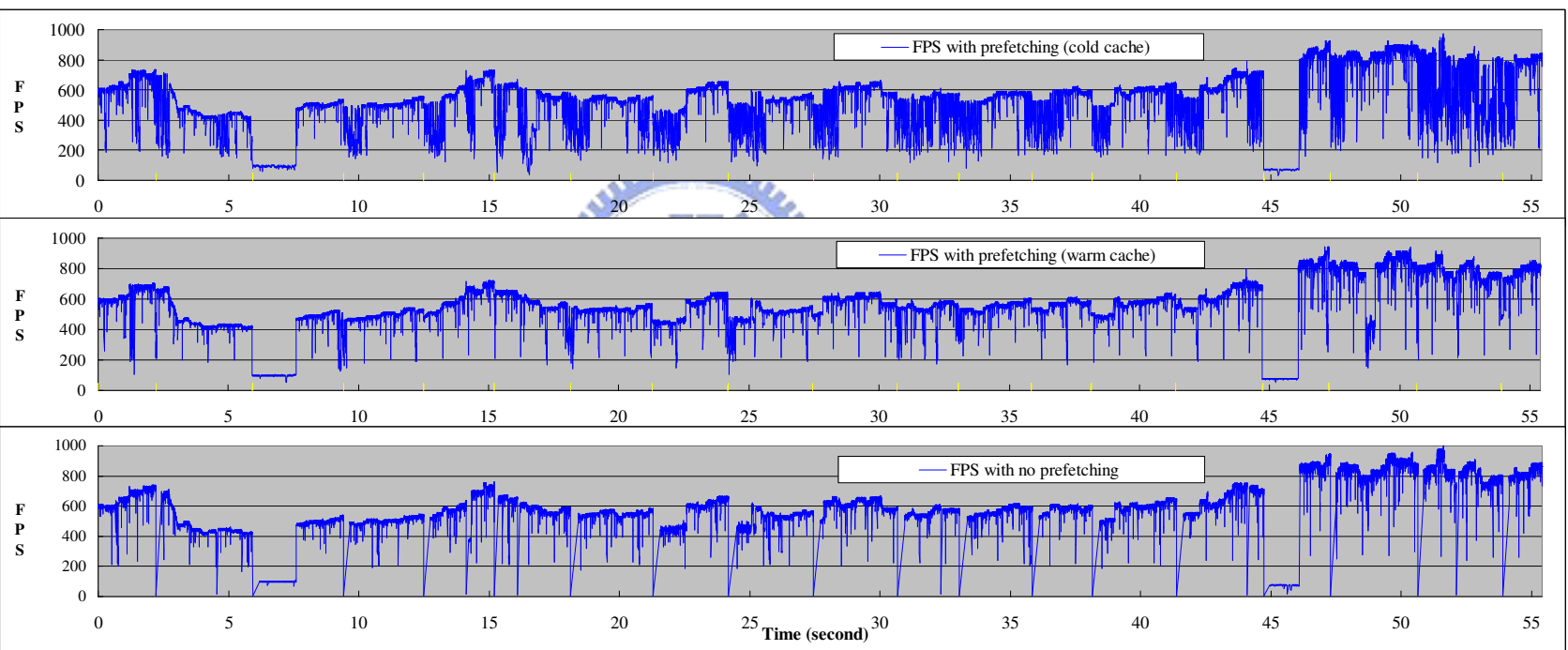


Figure 5.19: Run-time statistics of configuration C on scene 8M-50-5-4.5: The frame rates with prefetching under a cold cache and a warm cache, and without prefetching.

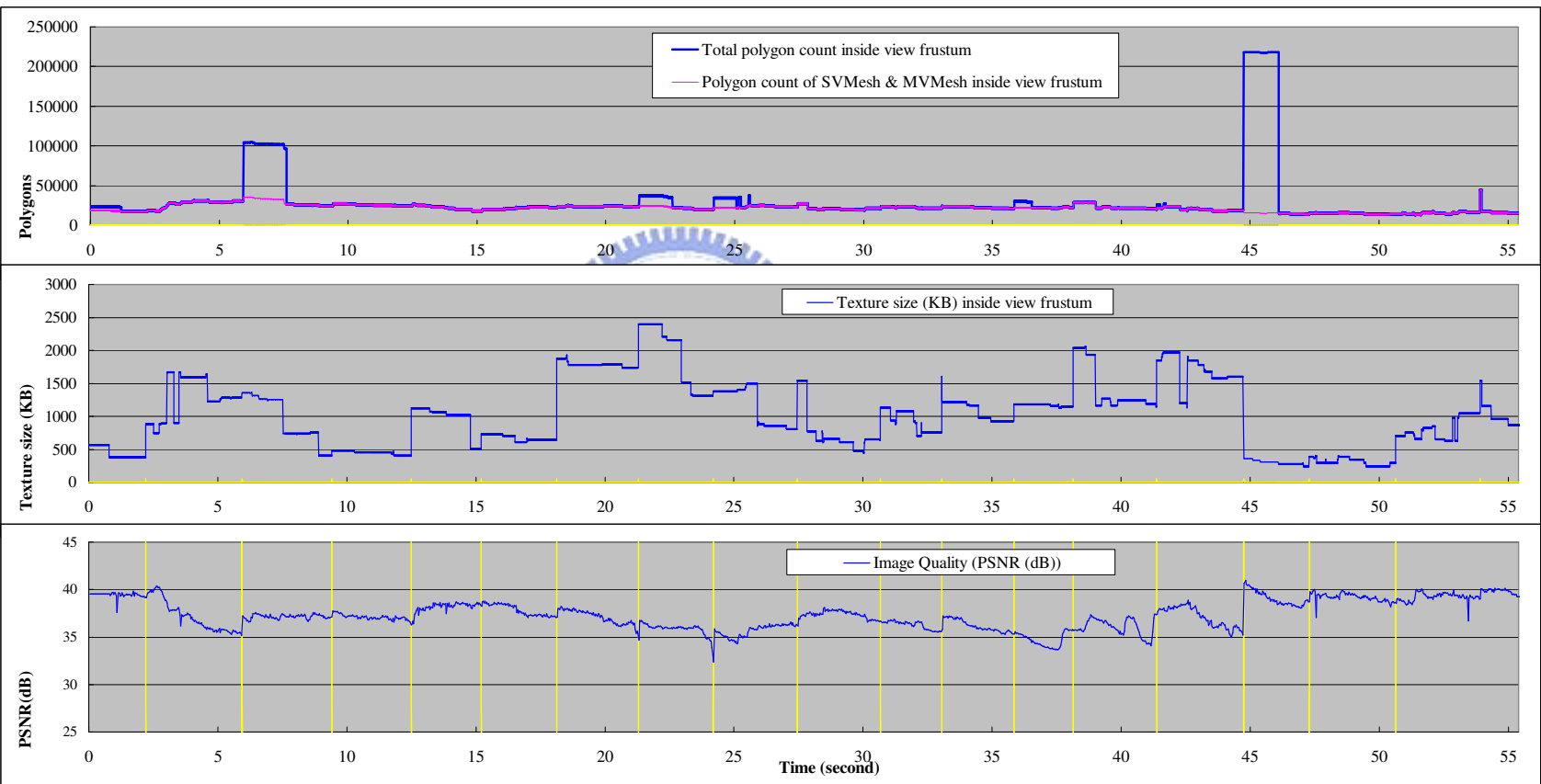


Figure 5.20: Run-time statistics of configuration C on scene 8M-50-5-4-5: The polygon count, texture requirements, and image quality.

64.4ms. Under the assumption that the maximum navigation speed is 30/s. and the cell size is 50, the time between cell transitions will be 2887ms for the path 1 shown in Figure 5.21. The ratio of the average loading time for a cell over the time between transitions is only 2.23%. On the other hand, for the case of path 2, the ratio is 4.46%. Several implementation details can be included to smooth out the loading time. First of all, the loading can be easily amortized into in-cell frames without notice since disk I/O can run in parallel. Secondly, when navigating in high speed, user perception is more sensitive to smooth frame rate than image quality. In this case, the texture can be mapped with lower resolutions, which implies smaller size for loading.

Table 5.9: Storage and loading time under different scene complexities ( $T_s = 5, T_l = 4.5$ ).

Scene complexity	2M	4M	8M
Total secondary storage requirement (MB)			
SVMesh & MVMesh	101.4	218.6	408.8
Textures	209.3	440.5	850.8
Total	310.7	659.1	1,260
Original meshes	27.8	59.6	112.6
Average run-time requirement (KB) per cell			
Potentially visible polygons	71.4	154.2	287.2
Textures	481.2	1,011	1,965
Total	552.6	1,165	2,252
Average loading time (ms) per cell			
Hard disk (55MB/s.)	10.0	21.2	41.0
Hard disk (35MB/s.)	15.8	33.3	64.4

In the experiment, we found that textures can only be loaded from the hard disk to main memory, and then to texture memory. Transferring data to texture memory, however, has to compete with the data transferring between CPU and graphics accelerator. Another problem appeared is that textures in general have size in a wide range, for example, from several bytes to hundred kilobytes. In consequence, the prefetching of textures cannot be easily amortized effectively to in-cell frames.

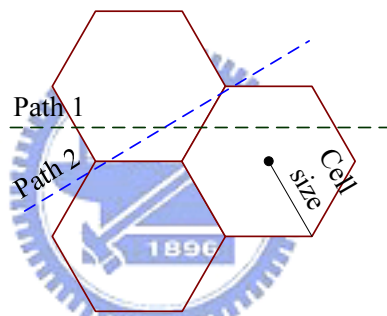


Figure 5.21: Cases of cell transition.

# Chapter 6

## Conclusion

We have addressed three important real-time rendering problems. First of all, we have presented a texture adaptation scheme for mapping textures to progressive meshes, aiming to eliminate texture distortion introduced by edge collapses. The texture adaptation applied to each edge collapse or vertex split is a local, incremental, and invertible operation, and can be effectively accelerated by graphics hardware. We have also proposed the mechanism of indexing mapping to reduce blurred artifacts due to under-sampling that might be introduced in texture adaptation. The experiment results have revealed that the texture adaptation scheme is capable of eliminating texture distortion in a very efficient manner.

Secondly, based on the texture adaptation and the indexing map, we have presented a new mapping-based error metric that is able to accurately measure the simplification error even in the presence of badly parameterized texture maps. Maximum and average error as well as incremental and total error are discussed in depth. The use of indexing map in the texture adaptation enables an efficient and simple evaluation of total average error. The proposed new error metric together with the texture adaptation scheme outperforms well-known error error metric QEM and APS in terms of the geometric and appearance preserving in mapping textures to progressive meshes. We also observed that the proposed total maximum error metric alone performs better than APS and almost the same as QEM.

Finally, we have presented a hybrid scheme for real-time rendering of complex scenes. The scheme partitions the model space into cells, thus explores the locality of visibility based on which the objects outside a cell are rendered as textured LOD meshes and inside objects are rendered as normal. Such a hybrid representation allows us to avoid problems that are commonly found in image-based rendering; such as the gap problem due to resolution mismatch and the hole problem due to occlusion among objects. The representation also constrains the hole due to self-occlusion to be within a user-specified tolerance. A prefetching mechanism has also been proposed to predict data of which neighboring cells will be needed shortly and how the loading can be amortized to frames before crossing the cell boundary. In the proposed scheme, acceleration techniques such as regional occlusion culling, back-facing culling, and run-time occlusion culling can be easily integrated. We have demonstrated our system on several scenes consisting of millions of polygons and observed very encouraging results. For a scene of 8 millions of polygons, we have achieved higher than 600 frames per second with a little loss of image quality (average PSNR 37.34dB) on the current graphics hardware. The polygons and textures require about 1260MB secondary storage space and about 294MB main memory on average.

Several problems along this research direction require further study. Although the texture adaptation operation is theoretically invertible, in practice artifact may appear after several runs of edge collapses and vertex splits. Such artifact stems from the discrete nature of texture mapping. Although current testing results revealed that only very small or invisible artifacts were found, however, we need to look into this problem and see how serious the problem could be in practice. Due to the success in eliminating texture distortion, the proposed texture adaptation scheme points out to a possible way of re-parameterizing a mesh. The proposed mapping-based error metric has demonstrated its capability to outperform APS and work about the same as QEM when it is formulated in the form of maximum error. We will study the sampling strategy that is required for evaluating the average error and expect to have an error metric that is more competent than QEM.

# Bibliography

- [1] P. K. Agarwal and S. Subhash. Surface Approximation and Geometric Partitions. In *Proceedings of 5th ACM-SIAM Symposium on Discrete Algorithms*, pages 24–33, 1994.
- [2] D. Aliaga, J. D. Cohen, A. Wilson, E. Baker, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. MMR: An Interactive Massive Model Rendering System Using Geometric and Image-Based Acceleration. In *Proceedings of 1999 ACM Symposium on Interactive 3D Graphics*, pages 199–206, 1999.
- [3] C. Bajaj and D. Schikore. Error-Bounded Reduction of Triangle Meshes with Multivariate Data. *SPIE*, 2656:34–45, 1996.
- [4] C.-C. Chen and J.-H. Chuang. Viewcell-Dependent Geometry Simplification Using Depth. In *Computer Graphics Workshop 2002*, June 2002.
- [5] S. E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In R. Cook, editor, *Proc. SIGGRAPH '95*, pages 29–38, August 1995.
- [6] S. E. Chen and L. Williams. View Interpolation for Image Synthesis. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 279–288, August 1993.
- [7] P. Cignoni, C. Montani, C. Rocchini, and R. Scopigno. A General Method for Recovering Attribute Values on Simplified Meshes. In *Proceedings of IEEE Visualization '98*, pages 59–66, October 1998.



- [8] P. Cignoni, C. Montani, C. Rocchini, R. Scopigno, and M. Tarini. Preserving Attribute Values on Simplified Meshes by Resampling Detail Textures. *The Visual Computer*, 15(10):519–539, 1999.
- [9] P. Cignoni, C. Rocchini, and R. Scopigno. Metro: Measuring Error on Simplified Surfaces. *Computer Graphics Forum*, 17(2):167–174, 1998.
- [10] J. H. Clark. Hierarchical Geometric Models for Visible Surface Algorithms. *Communications of the ACM*, 19(10):547–554, October 1976.
- [11] J. D. Cohen. Concepts and Algorithms for Polygonal Simplification. In *SIGGRAPH 99 Course Tutorial #20*, pages C1–C34, 1999.
- [12] J. D. Cohen and D. Manocha. *The Visualization Handbook*, chapter 20. Model Simplification, pages 393–411. Elsevier, 2005.
- [13] J. D. Cohen, D. Manocha, and M. Olano. Simplifying Polygonal Models Using Successive Mappings. In *Proceedings of IEEE Visualization '97*, pages 395–402, October 1997.
- [14] J. D. Cohen, D. Manocha, and M. Olano. Successive Mappings: An Approach to Polygonal Mesh Simplification with Guaranteed Error Bounds. *International Journal of Computational Geometry and Applications*, 13(1):61–94, February 2003.
- [15] J. D. Cohen, M. Olano, and D. Manocha. Appearance-Preserving Simplification. In *Proc. SIGGRAPH '98*, pages 115–122, New York, NY, USA, 1998. ACM Press.
- [16] D. Cohen-Or, G. Fibich, D. Halperin, and E. Zadicario. Conservative Visibility and Strong Occlusion for Viewspace Partitioning of Densely Occluded Scenes. *Computer Graphics Forum*, 17(3):243–253, 1998.
- [17] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312, New York, NY, USA, 1996. ACM Press.

- [18] L. Darsa, B. Costa, and A. Varshney. Walkthroughs of Complex Environments using Image-based Simplification. *Computers & Graphics*, 22(1):55–69, 1998.
- [19] L. Darsa, B. C. Silva, and A. Varshney. Navigating Static Environments Using Image-Space Simplification and Morphing. In *Proceedings of 13th Symposium on Interactive 3D Graphics*, pages 25–34, 1997.
- [20] G. Das and D. Joseph. The Complexity of Minimum Convex Nested Polyhedra. In *Proceedings of 2nd Canadian Conference on Computational Geometry*, pages 296–301, 1990.
- [21] P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and Rendering Architecture from Photographs: A hybrid geometry- and image-based approach. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, pages 11–20, August 1996.
- [22] P. E. Debevec, Y. Yu, and G. D. Borshukov. Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping. In *Eurographics Rendering Workshop*, pages 105–116, June 1998.
- [23] X. Decoret, G. Schaufler, F. X. Sillion, and J. Dorsey. Multi-Layered Impostors for Accelerated Rendering. *Computer Graphics Forum*, 18(3):61–73, September 1999.
- [24] F. Durand, G. Drettakis, J. Thollot, and C. Puech. Conservative Visibility Preprocessing using Extended Projections. In Kurt Akeley, editor, *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pages 239–248, July 2000.
- [25] I. Eckstein, V. Surazhsky, and C. Gotsman. Texture Mapping with Hard Constraints. *Computer Graphics Forum*, 20(3):95–104, 2001.
- [26] M. S. Floater. Parametrization and Smooth Approximation of Surface Triangulations. *Computer Aided Geometric Design*, 14(4):231–250, 1997.
- [27] M. S. Floater. Mean Value Coordinates. *Computer Aided Geometric Design*, 20(1):19–27, 2003.

- [28] M. Garland. Multiresolution Modeling: Survey & Future Opportunitites. In *Eurographics '99 – State of the Art Reports*, pages 111–131, 1999.
- [29] M. Garland and P. S. Heckbert. Surfaces Simplification Using Quadric Error Metrics. In *Proc. of SIGGRAPH '97*, pages 209–216, 1997.
- [30] M. Garland and P. S. Heckbert. Simplifying Surfaces with Color and Texture Using Quadric Error Metrics. In *Proceedings of IEEE Visualization '98*, pages 263–269, October 1998.
- [31] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*, chapter 4, page 228. Addison-Wesley, September 1993.
- [32] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The Lumigraph. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996.
- [33] N. Greene, M. Kass, and G. Miller. Hierarchical Z-Buffer Visibility. In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH '93 Proceedings)*, pages 231–238, August 1993.
- [34] P. Heckbert and M. Garland. Survey of Surface Simplification Algorithms. In *SIGGRAPH '97, Multiresolution Surface Modeling, Course Notes No. 25*. ACM SIGGRAPH, 1997.
- [35] H. Hoppe. Progressive Meshes. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, pages 99–108, August 1996.
- [36] T. Hudson, D. Manocha, J. D. Cohen, M. Lin, K. Hoff, and H. Zhang. Accelerated Occlusion Culling using Shadow Frusta. In *Proceedings of 13th Symposium on Computational Geometry*, pages 1–10, 1997.
- [37] A. Khodakovsky, N. Litke, and P. Schröder. Globally Smooth Parameterizations with Low Distortion. *Proc. SIGGRAPH 2003, ACM Trans. on Graphics*, 22(3):350–357, 2003.
- [38] H. Kim and K. Wohn. Multiresolution Model Generation with Geometry and Texture. In *Seventh International Conference on Virtual Systems and Multimedia (VSMM'01)*, pages

780–789. IEEE, Oct 2001.

- [39] S. Kumar, D. Manocha, B. Garrett, and M. Lin. Hierarchical Back-face Culling. In *7th Eurographics Workshop on Rendering*, pages 231–240, 1996.
- [40] M. Levoy and P. Hanrahan. Light Field Rendering. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, pages 31–42, August 1996.
- [41] P. Lindstrom and G. Turk. Fast Memory Efficient Polygonal Simplification. In *Proceedings of IEEE Visualization '98*, pages 279–286, 1998.
- [42] P. Lindstrom and G. Turk. Image-Driven Simplification. *ACM Trans. on Graphics*, 19(3):204–241, July 2000.
- [43] D. Luebke, B. Watson, J. D. Cohen, M. Reddy, and A. Varshney. *Level of Detail for 3D Graphics*. Elsevier Science Inc., New York, NY, USA, 2002.
- [44] L. McMillan and G. Bishop. Plenoptic Modeling: An Image-Based Rendering System. In R. Cook, editor, *Proc. SIGGRAPH '95*, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995.
- [45] E. Puppo and R. Scopigno. Simplification, LOD, and Multiresolution - Principles and Applications. In *Eurographics '97 Tutorial Notes*, 1997.
- [46] J. Rohlf and J. Helman. IRIS Performer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 381–394, New York, NY, USA, 1994. ACM Press.
- [47] P. V. Sander, S. J. Gortler, J. Snyder, and H. Hoppe. Signal-Specialized Parametrization. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering*, pages 87–98, Aire-la-Ville, Switzerland, 2002. Eurographics Association.
- [48] P. V. Sander, J. Snyder, S. J. Gortler, and H. Hoppe. Texture Mapping Progressive Meshes. In *Proc. SIGGRAPH 2001*, pages 409–416, New York, NY, USA, 2001. ACM Press.

- [49] G. Schaufler, J. Dorsey, X. Decoret, and F. X. Sillion. Conservative Volumetric Visibility with Occluder Fusion. In Kurt Akeley, editor, *Computer Graphics (SIGGRAPH 2000 Proceedings)*, pages 229–238, July 2000.
- [50] G. Schaufler and W. Stürzlinger. A Three-Dimensional Image Cache for Virtual Reality. In *Proceedings of Eurographics '96*, pages 227–236, August 1996.
- [51] M. Segal, C. Korobkin, R. Widenfelt, J. Foran, and P. Haeberli. Fast Shadows and Lighting Effects Using Texture Mapping. In *Computer Graphics (SIGGRAPH '92 Proceedings)*, pages 249–252, July 1992.
- [52] S. M. Seitz and C. R. Dyer. View Morphing. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, pages 21–30, August 1996.
- [53] J. Shade, D. Lischinski, D. H. Salesin, T. DeRose, and J. Snyder. Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments. In H. Rushmeier, editor, *Proc. SIGGRAPH '96*, pages 75–82, August 1996.
- [54] J. W. Shade, S. J. Gortler, L.-W. He, and R. Szeliski. Layered Depth Images. In *Proc. SIGGRAPH '98*, pages 231–242, July 1998.
- [55] F. Sillion, G. Drettakis, and B. Bodelet. Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery. In *Proceedings of Eurographics'97*, pages 207–218, Budapest, Hungary, September 1997.
- [56] G. Turk. Re-Tiling Polygonal Surfaces. In *SIGGRAPH '92: Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, pages 55–64, New York, NY, USA, 1992. ACM Press.
- [57] A. Xu, S. Sun, and K. Xu. Texture Information Driven Triangle Mesh Simplification. In M.H. Hamza, editor, *Computer Graphics and Imaging - CGIM2005*, pages 73–77, Honolulu, Hawaii, Aug 2005. ACTA Press.

- [58] H. Zhang and K. E. Hoff III. Fast Backface Culling Using Normal Masks. In *Proceedings of 13th Symposium on Interactive 3D Graphics*, pages 103–106, 1997.
- [59] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility Culling Using Hierarchical Occlusion Maps. In *Computer Graphics*, volume 31, pages 77–88, 1997.
- [60] M. Zhang, Z. Pan, and P.-A. Heng. A Near Constant Frame-rate Rendering Algorithm Based on Visibility Computation and Model Simplification. In *Proceedings of VSMM 2002*, pages 387–398, 2002.
- [61] K. Zhou, X. Wang, Y. Tong, M. Desbrun, B. Guo, and H.-Y. Shum. TextureMontage: Seamless Texturing of Arbitrary Surfaces From Multiple Images. *Proc. SIGGRAPH 2005*, *ACM Trans. on Graphics*, 24(3):1148–1155, 2005.

