3D mesh transmission over bandwidth-limited networks

Guan, Wei

2008

Guan, W. (2008). 3D mesh transmission over bandwidth-limited networks. Master’ s thesis, Nanyang Technological University, Singapore.

https://hdl.handle.net/10356/15166

https://doi.org/10.32657/10356/15166

3D Mesh Transmission over Bandwidth-Limited Networks

A Thesis
Submitted to the School of Computer Engineering
of the Nanyang Technological University

by

Guan Wei

in fulfillment of the Requirement for the degree of
Master of Engineering by Research

August 2008

Abstract

With the advance of wireless communications and the increased processing power of wireless devices, network-based graphics applications such as online virtual reality and online games are now being extended from wired broadband networks to wireless networks. However, it is a challenging task to transmit complex graphics models over wireless networks due to huge data volume of complex 3D graphics models and limited bandwidth of wireless channels. The objectives of this research is to re-exam the existing mesh compression techniques and adapt them for efficient 3D mesh transmission over bandwidth-limited networks.

By noticing that most of the existing 3D mesh coding algorithms transmit unnecessary invisible portions of 3D mesh models, we propose a novel view-dependent 3D model transmission scheme, where a 3D model is partitioned into a number of segments, each segment is then independently coded using the MPEG-4 3DMC coding algorithm, and finally only the visible segments are selected and delivered to the client. Moreover, we also propose analytical models to find the optimal number of segments so as to minimize the average transmission size. Simulation results show that such a view-based 3D model transmission is able to substantially save the transmission bandwidth and therefore has a significant impact on wireless graphics applications.

We further study wavelet-let based progressive 3D mesh transmission. In particular, we consider the latest wavelet based 3D mesh coding schemes, which convert an irregular mesh into a semi-regular mesh and directly apply the zerotree-like image coders to compress the wavelet vectors generated in the remeshing process. By noticing that the particular properties of semi-regular meshes are not being considered in the zerotree-like image coders, we propose to introduce a preprocessing step to scale up the vector wavelets generated in remeshing so that the inherent dependency of wavelets can be truly understood by the zerotree-like image compression algorithms. Furthermore, we propose to incorporate the illumination effects into the view-depend progressive mesh transmission system to further improve the performance. In this way, we not only avoid transmitting invisible parts but also spend less bits on visually unimportant bright/dark portions given a limited bandwidth. Experimental results show that significant quality improvement can be achieved by taking into account the illumination effects.

Acknowledgments

I am deeply indebted to my thesis adviser Prof. Cai Jianfei for his encourage, help, and support in so many ways; especially his absolute trust when I encountered stagnation and difficulties in research.

I would like to thank Prof. Zheng Jianmin for his constructive suggestions and continuous support during my entire research process. The discussions with him are always interesting and helpful. I am also glad to have the opportunities to collaborate with Prof. Chen Chang Wen and PhD student Zhang Juyong.

I have to express my appreciation to Lu Feng, Han Shuguo, Wu Min and many others for their concerns, kindness and help during my stay in Center for Multimedia and Network Technology (CeMNet). These friends are really supportive.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Computer graphics have penetrated into our daily life everywhere, from entertainment to education, from medical science to military, and they are playing more and more active roles. At the mean time, with the advance of network infrastructures and the increased processing power of terminal devices, graphics applications such as virtual reality and games are now being extended from computer-based to network-based. For applications running on networks, especially bandwidth-limited networks like wireless networks, an important challenge is how to transmit large amount of data or complex 3D models over such communication channels. Usually, 3D models are represented by triangular or polygonal meshes. A photorealistic 3D mesh model typically has thousands to millions of vertices and complicated topology that refers to the connectivity relationship among vertices. Thus, efficient model processing is indispensable for reducing storage space and transmission bandwidth requirements.

1.1 Background

To save transmission bandwidth, compression techniques are needed. In the past decade, many mesh compression schemes [Dee95, TR98, TR99, TG98] have been developed. In [Dee95], Deering introduced the concept of geometry compression and proposed a simple but low-complexity compression algorithm, which can achieve a six to ten compression ratio with only little loss in display quality. Taubin *et al.* [TR98] introduced a topological surgery (TS) method, where a mesh is represented by one vertex spanning tree and several simple polygons, and the connectivity information is losslessly encoded. The proposed TS algorithm has been adopted in the MPEG-4 standard for single-rate mesh coding, i.e. 3D mesh coding (3DMC). Rossignac [TR99] described an Edgebreaker algorithm to encode the topology information of a 3D mesh. Touma and Gotsman [TG98] proposed a valence-driven algorithm, which achieves the state-of-the-art compression performance, i.e. 1.5 bits per vertex (bpv) on average for encoding mesh connectivity.

In addition to single-rate mesh coding, many progressive 3D mesh compression algorithms [Hop96, TGHL98, PR00, KSS00] have also been developed. They allow to encode a mesh model once and decode it at multiple reduced rates with different level-of-details (LODs). Depending on the simplification or decimation approach used, progressive mesh coding can be classified into three categories: incremental decimation, vertex clustering and remeshing (or resampling). One example for the first category is the progressive mesh (PM) algorithm proposed in [Hop96], where Hoppe uses successive edge collapse operations for mesh simplification. The representative schemes for the second category include the progressive forest split (PFS) algorithm [TGHL98] and the compressed progressive mesh (CPM) algorithm [PR00], where the common idea is to group vertices to improve the compression performance at the cost of reduced granularity [PKK05].

The typical idea of the algorithms in the third category such as [KSS00] is to convert an irregular mesh into a semi-regular one, based on which a multi-resolution representation can be easily established. Such a conversion process is called remeshing. For example, Schroder *et al.* provide an efficient remeshing technique using the MAPS algorithm [LSS$^+$98]. In that scheme, an irregular mesh is first simplified into a base mesh. Each triangle in the original mesh can be mapped into an "internal" triangle within a base triangle. Then, the base mesh is subdivided, and the new vertices obtained through subdivision are mapped back to the vertices in the original mesh. Finally, the base mesh and these mapped vertices compose of a semi-regular mesh. In addition, the generated semi-regular mesh can be efficiently compressed using wavelet based image coding schemes [KSS00].

1.2 Motivation and Objective

Although numerous mesh compression schemes have been proposed in literature, it might not be efficient to directly apply them for transmission over bandwidth-limited networks such as wireless networks. One problem is that most of the existing approaches compress and transmit the entire 3D mesh model together. In fact, in many graphics applications, only one particular view of a 3D mesh model is needed. Transmitting the entire 3D model leads to a significant waste in precious wireless network bandwidth. Similarly, another problem is that most of the existing methods do not consider the illumination effects, which results in transmitting the perceptually unimportant dark or bright portions to the clients.

The objectives of this research is to re-exam the existing mesh compression techniques and adapt them for efficient 3D mesh transmission over bandwidth-limited networks. One key idea is to only transmit visible portions and the transmission of invisible parts should be avoided. The challenges include identifying the visible portions, quickly fetching them out from the compressed bitstream, and assembling them into a prioritized bitstream. Another main idea is to consider not only view dependency but also illumination dependency. The challenges include how to quantitatively measure the perceptual importance of individual parts and how to trade off among them.

1.3 Organization of Thesis

The rest of the thesis is organized as follows. In Chapter 2, we propose a segmentation-based view-dependent 3D mesh transmission scheme using single-rate mesh compression. In Chapter 3, we study wavelet-based progressive mesh compression. We first propose an improved 3D mesh coder. Then, we further develop a progressive 3D mesh transmission system, which takes into account not only the view dependency but also the illumination dependency. Finally, we conclude this thesis and discuss some potential research directions in Chapter 4.

Chapter 2

View-Dependent 3D Mesh Transmission

For wireless network based graphics applications, a key challenge is how to efficiently transmit complex three-dimensional (3D) models over bandwidth-limited wireless channels. Most existing 3D mesh transmission systems do not consider such a view-dependent delivery issue, and thus transmit unnecessary portions of 3D mesh models, which leads to the waste in precious wireless network bandwidth. In this chapter, we propose a novel view-dependent 3D model transmission scheme, where a 3D model is partitioned into a number of segments, each segment is then independently coded using the MPEG-4 3DMC coding algorithm, and finally only the visible segments are selected and delivered to the client. Moreover, we also propose analytical models to find the optimal number of segments so as to minimize the average transmission size. Simulation results show that such a view-based 3D model transmission is able to substantially save the transmission bandwidth and therefore has a significant impact on wireless graphics applications.

2.1 Background

Though there are several mesh transmission systems proposed by some researchers, the major drawback for most existing 3D mesh coding and transmission systems is that they compress and transmit the entire 3D mesh model together. In fact, in many graphics applications, only one particular view of a 3D mesh model is needed. Such a view-dependent delivery issue has not received much attention in the graphics compression community. This has not been a severe problem under the traditional graphics applications scenario where a 3D mesh model is compressed and rendered locally. However, for the contemporary wireless graphics applications, it becomes a significant waste to transmit the invisible portions of a 3D mesh model over wireless networks, where the bandwidth resource is extremely precious.

To solve this challenging problem originated from wireless graphics applications, in this chapter, we propose a view-dependent 3D model transmission scheme to efficiently transmit only the relevant portions of a 3D model to the client. The proposed scheme first splits a 3D model into many small segments based on hierarchical face clustering. Then, each segment is independently coded using the MPEG-4 3DMC coding algorithm. Finally, the server selects and transmits only the relevant segments that are related to the client's requests.

An initial attempt to solve this problem was proposed in [YKK05b], in which the authors presented a feasible approach for view-dependent progressive mesh coding and streaming. Our work differs from the one in [YKK05b] in the following aspects: 1) three criteria: normal similarity, coding redundancy and size uniformity, are proposed as merging cost to well control the segmentation process to fit different needs in view-based 3D model transmission; 2) the employed coding of segments is truly independent while the scheme in [YKK05b] needs a perfect base model; therefore, our scheme is more suitable for non-priority based 3D model transmission over lossy networks; 3) our proposed segment selection based on both viewing frustum and visible direction is much more efficient. Simulation results show that our proposed scheme can save more than 50% bandwidth at the cost of duplicate vertices stored at the server side.

Moreover, we also study the problem of finding optimal number of segments. We develop analytical models to estimate the overall compression size and the average transmission size under different numbers of segments, based on which the optimal number of segments can be easily derived. Simulation results demonstrate the accuracy of our proposed models.

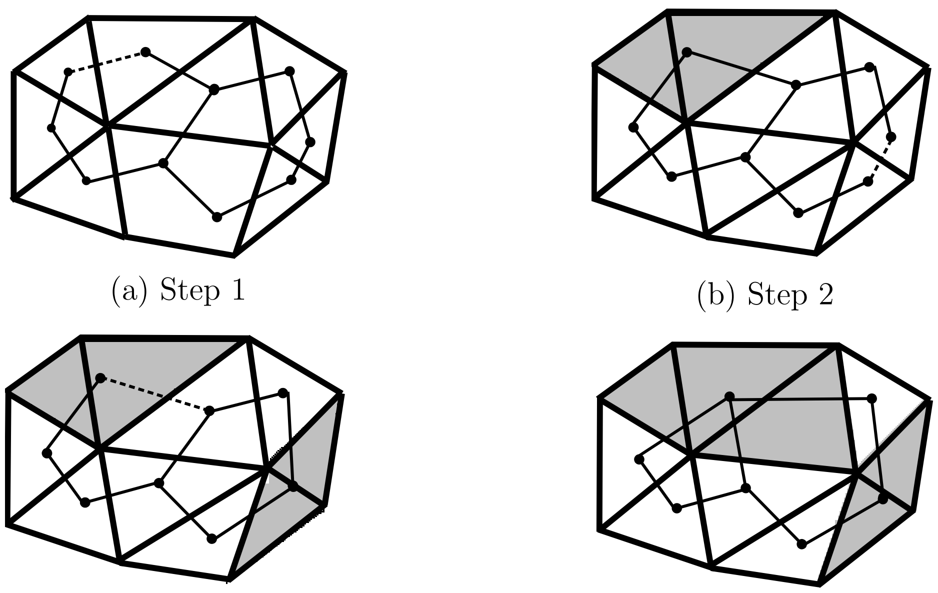
2.2 Segmentation of Triangular Meshes

2.2.1 Segmentation Algorithm

So far, many mesh segmentation algorithms have been proposed in literature. According to [AKM$^+$06], the existing mesh segmentation algorithms can be grouped into two categories: purely geometric sense and more semantics-oriented manner. In the first category [GWH01, CSAD04, SSGH01, ZSGS04], a mesh model is segmented into a number of patches that have some uniform property like curvature, while the algorithms in the second category [CDST97, MW99, STK02, LZ04] aim at identifying parts of an object corresponding to some semantic meanings. Our mesh segmentation algorithm belongs to the first category.

In particular, our mesh segmentation algorithm performs in a way that is similar to the hierarchical face clustering approach [GWH01, ASF06]. The algorithm starts from the given mesh model, where each triangle is initially considered as a segment. Then the neighboring segments are iteratively merged into a bigger segment. The algorithm calculates the cost of merging any two neighboring segments based on some criteria explained in Section 2.2.2. The merging with the minimum cost will be performed and the corresponding two segments are thus grouped into a bigger one. This process continues until the termination condition is satisfied.

In implementation, we use a dual graph to describe the relationship of the segments in the mesh. Each node in the dual graph corresponds to a segment, and if two segments are neighbors, there is a dual edge connecting the two corresponding nodes. Then merging two neighboring segments is equivalent to collapsing an edge. Figure 2.1 illustrates such a process.


(a) Step 1 (b) Step 2

(c) Step 3 (d) Step 4

Figure 2.1: The process of edge collapse, where the bold lines and the thin lines belong to the original mesh model and the corresponding dual graph, respectively. The dotted lines are the lines selected to be collapsed in the next step.

2.2.2 Merging Cost

Note that the performance of the proposed segmentation algorithm depends on the choice of the merging cost. In our view-dependent transmission scheme, the mesh is split into a number of segments and they are stored in the server side. To make the transmission efficient, we should avoid the unnecessary transmission of those invisible segments and meanwhile we should make the total compressed data as small as possible. To this end, we define our merging cost to be a linear combination of three components. Specifically, the merging cost $E_{i,j}$ for collapsing a dual edge connecting node i and node j is

$$E_{i,j} = E_{n(i,j)} + \alpha_1 E_{r(i,j)} + \alpha_2 E_{u(i,j)}, \tag{2.1}$$

where $E_{n(i,j)}$, $E_{r(i,j)}$, and $E_{u(i,j)}$ stand for the costs based on normal similarity, redundancy, and size uniformity criteria, respectively; and α_1 and α_2 are tradeoff constants, falling within [0, 1]. In general, both α_1 and α_2 are chosen to be small numbers to ensure that $E_{n(i,j)}$ is the dominant criterion. The detailed explanation and computation of these three components are given below.

2.2.2.1 Normal Similarity Criterion

It is easy to observe that triangles with similar normal directions are likely visible simultaneously and triangles with very different normal directions can hardly appear in the same view. Therefore the normal direction is considered to be the main criterion of merging and we compute the merging cost based on the normal variance.

Let C_i and C_j be two neighboring segments containing p and q triangles, respectively. Denote the areas of these triangles by $s_{i1}, s_{i2}, ..., s_{ip}$ and $s_{j1}, s_{j2}, ..., s_{jq}$, and denote the unit normal vectors of these triangles by $n_{i1}, n_{i2}, ..., n_{ip}$ and $n_{j1}, n_{j2}, ..., n_{jq}$. We compute a weighted mean of these normal vectors:

$$m_{ij} = \frac{\sum_{k=1}^{p} s_{ik} n_{ik} + \sum_{h=1}^{q} s_{jh} n_{jh}}{\| \sum_{k=1}^{p} s_{ik} n_{ik} + \sum_{h=1}^{q} s_{jh} n_{jh} \|}. \tag{2.2}$$

The angle between each triangle normal and the mean can be computed by $\alpha_{ik} = \arccos(n_{ik} \cdot m_{ij})$ and $\alpha_{jh} = \arccos(n_{jh} \cdot m_{ij})$, where $k = 1, 2, ..., p; h = 1, 2, ..., q$. Then the merging cost based on the normal similarity criterion is defined as

$$E_{n(i,j)} = \frac{\sum_{k=1}^{p} s_{ik} \alpha_{ik}^2 + \sum_{h=1}^{q} s_{jh} \alpha_{jh}^2}{\sum_{k=1}^{p} s_{ik} + \sum_{h=1}^{q} s_{jh}}. \tag{2.3}$$

2.2.2.2 Redundancy Criterion

Note that in our scheme the mesh is split into segments and each segment is coded independently. This will result in boundary duplication. To reduce the redundancy, we introduce a redundancy related term into the merging cost. Specifically, we consider the relative reduction in the number of the boundary vertices when merging two segments. If γ_i and γ_j are the numbers of vertices in segments C_i and C_j, and γ_{ij} is the number of vertices lying on both C_i and C_j, then we define the redundancy related cost as

$$E_{r(i,j)} = \frac{\gamma_i + \gamma_j - \gamma_{ij}}{\gamma_i + \gamma_j}. \tag{2.4}$$

When the number of boundary vertices γ_{ij} is large, which corresponds to small $E_{r(i,j)}$, merging C_i and C_j will reduce a lot of redundant vertices. This suggests the two segments should be grouped together.

2.2.2.3 Size Uniformity Criterion

For network-based applications, sometimes it is desired that the sizes of the segments in terms of the number vertices or the number of triangles do not vary too much such as in the cases of error-resilient transmission [YKK05a]. For this purpose, we introduce another criterion to force segmentation conducted uniformly. Let λ_i and λ_j be the number of triangles in segments C_i and C_j, and λ_{max} be the maximum number of triangles in all the segments. We define the size uniformity cost as

$$E_{u(i,j)} = \frac{\lambda_i + \lambda_j}{2\lambda_{max}}$$

This criterion encourages small segments to be merged. Therefore, the size of each segment in term of the number of triangles will be increased uniformly as the iteration goes on.

2.3 Segment Selection and Assembling for Transmission

After the mesh is split into segments and each segment is coded independently, the next step is to determine which segments should be transmitted given the viewing parameters provided by the client. In the following, we first present our proposed segment selection scheme that contains two components: viewing frustum and visible direction. Then, we discuss how to assemble the selected segments into a prioritized bitstream.

2.3.1 Viewing Frustum

In computer graphics, a viewing frustum, defined by a near plane, a far plane, and four side planes intersected at the view point (see Figure 2.2), is used to specify the visible region. Only those objects located within the frustum would be perceived. Therefore, once the server receives the frustum specification provided by a client, the server should determine which segments are lying outside the frustum and they should not be transmitted.

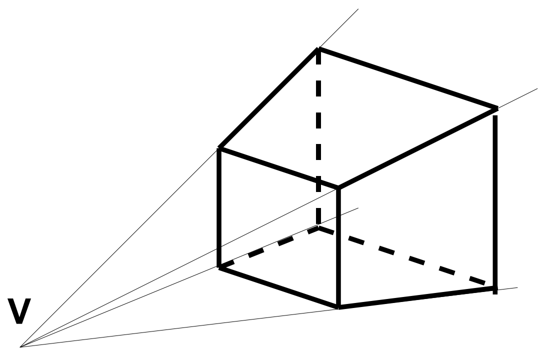


Figure 2.2: A viewing frustum.

To simplify the process of checking whether a segment and the viewing frustum overlap, the bounding volume technique is employed. Unlike the work in [YKK05b], where a bounding sphere is used, here we use an oriented bounding box. This is because our segments are usually quite flat due to the segmentation criteria and the oriented bounding box can enclose the flat shape more tightly. To construct an oriented bounding box, three orthogonal axes need to be specified first. Since the triangles within a segment are having similar normal vectors, we choose the average of these normal vectors as one axis direction. The other two orthogonal directions are arbitrarily chosen on the plane which is perpendicular to the first direction (see Figure 2.3). After this, an oriented bounding box aligning with these three axes is computed, which encloses all the vertices of the segment. In this way, each segment is associated with an oriented bounding box. To test if the segment and the viewing frustum overlap, we just check if the bounding box and the frustum overlap or not, which can be done relatively easily. If they do not overlap, the respective segment is assured to be outside the frustum and thus is not transmitted.

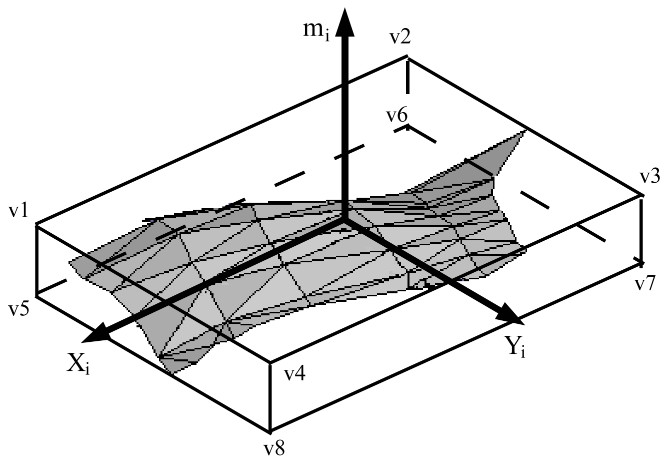


Figure 2.3: An oriented bounding box.

In particular, as shown in Fig. 2.3, the bounding box is described by eight vertices, which are obtained as follows. We first calculate the weighted average normal vector m_i for segment C_i as

$$m_i = \frac{\sum_{k=1}^{p} s_k n_k}{\sum_{k=1}^{p} s_k}, \tag{2.5}$$

where p is the number of triangles in the segment. Then, for all the vertices in the segment, we calculate the maximum and minimum coordinates along the m_i axis. Similarly, we calculate the extreme values for the other two orthogonal axis, and the combinations of these extreme values are the coordinates of the eight vertices that bound the segment.

We consider a segment is outside the viewing frustum if all its eight bounding box vertices v_1 to v_8 are beyond any one of the six sides of the frustum. Mathematically, we define that a segment is outside the frustum if the following is true

$$OR_{j=1-6}(AND_{i=1-8}(v_i \cdot f_j \geq t_j)), \tag{2.6}$$

where f_j and t_j, $j = 1, \ldots, 6$, are the normal directions and the corresponding thresholds for each of the six sides of the viewing frustum, respectively. An illustration is shown in Fig. 2.4.

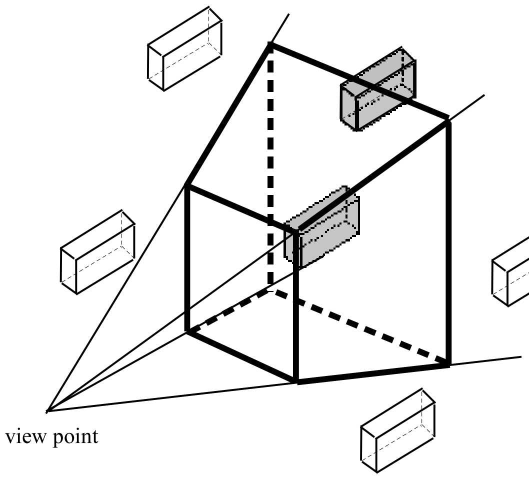


Figure 2.4: The white bounding boxes are outside the viewing frustum while the gray ones are overlapping with or inside the frustum.

2.3.2 Visible Direction

Even if the segments are within the viewing frustum, they may not be seen by the viewer if they are facing away from the viewer. Therefore, for each segment, we construct a cone called a blind-viewing cone. The blind-viewing cone is the range of view directions from which the viewer cannot see the segment. Once the server receives the viewing information from the client, we compare the viewing direction against the blind-viewing cone. If the viewing direction is contained by the cone, that means the segment is not visible and thus there is no need to transmit it.

To construct the blind-viewing cone, we construct the minimum normal cone first. Let a segment be composed of p triangles with normal vectors $n_1, n_2, ..., n_p$. The minimum normal cone is defined as the cone that contains all the normal vectors n_i and has the smallest cone angle (see Figure 2.5). The problem of finding the minimum normal cone is equivalent to a well-known problem called "Minimal Enclosing Circle", which can be solved by many methods. In our scheme, we modify Elzinga and Hearn's algorithm [EH72] to find our minimum normal cone. The detailed algorithm is described as follows.

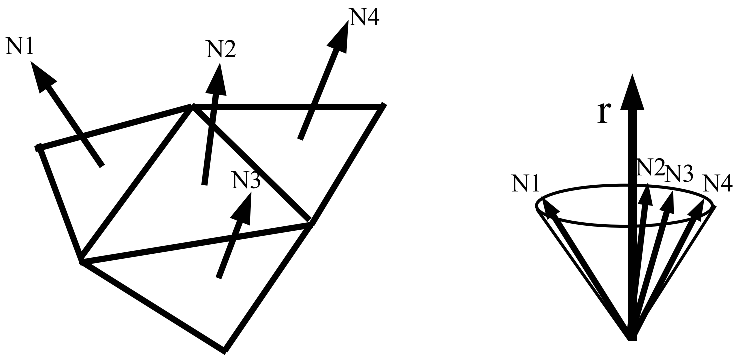


Figure 2.5: The minimum normal cone.

Step 1: From one apex, draw a cone c with the representative normal vector r such that all the normal vectors lie within the cone. We will make the cone smaller in the subsequent steps.

Step 2: Make the cone smaller by finding the normal vector n_i with the largest angle from the representative normal vector r, and drawing a cone with the same representative normal vector r and the conical surface passing through the normal vector n_i.

Step 3: If the conical surface passes through 2 or more normal vectors, go to step 4. Otherwise, make the opening angle smaller by moving the representative normal vector towards normal vector n_i.

Step 4: At this step, we know the cone is definitely passing through 2 or more normal vectors. If the conical surface contains a normal-vector-free section that is greater than half of the conical surface, the cone can be further reduced. Let n_i and n_j be the normal vectors at the boundary of the free section. While keeping n_i and n_j on the conical surface, we make the cone smaller by moving the representative normal vector away from the normal-vector-free section that is bounded by n_i and n_j until we have either case (a) or case (b).

- Case (a): The opening angle is the angle between n_i and n_j. Then the smallest cone is achieved. The new representative normal vector is $r = \frac{n_i+n_j}{|n_i+n_j|}$, and the half opening angle is $\alpha = \frac{1}{2}\arccos(n_i \cdot n_j)$.
- Case (b): The conical surface touches another normal vector n_k. Check whether there is a normal-vector-free section that is greater than half of the conical surface.

  (i) If no such section exists, then the minimal cone is determined by normal vectors n_i, n_j and n_k. The new representative normal vector be $r$can be easily obtained by solving the following equations

  $$\begin{cases} (r - \frac{n_i+n_k}{2}) \cdot (n_i - n_k) = 0 \\ (r - \frac{n_i+n_j}{2}) \cdot (n_i - n_j) = 0 \\ |r| = 1. \end{cases} \tag{2.7}$$

  The half opening angle is $\alpha = \arccos(r \cdot n_i)$.

  (ii) Otherwise, update the normal-vector-free section and go back to step 4.

Once the minimum normal cone described by the central axis vector r and the cone angle α is found, we can compute the blind-viewing cone. In particular, let $\vec{V}$ denote the normalized viewing direction vector that begins at the center of the viewing frustum and ends at the view point. As shown in Fig. 2.6, if the angle between $\vec{V}$ and r is large than $\frac{\pi}{2} + \alpha$, the segment is invisible. In other words, the blind-viewing cone is defined by the opposite central axis $-r$ with a cone angle θ, where $\theta = \frac{\pi}{2} - \alpha$.

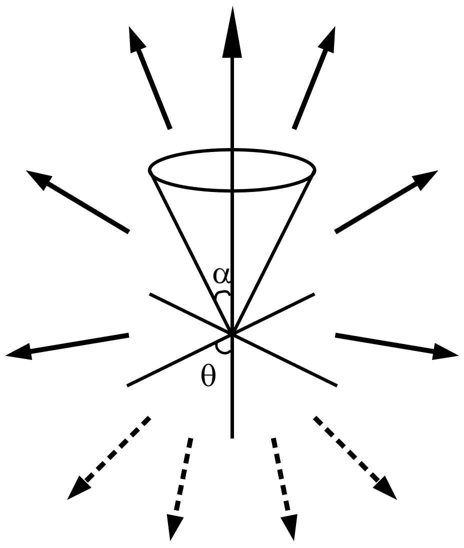


Figure 2.6: The solid arrows represent the view directions that can see the segment, while the dotted arrows represent the view directions that cannot see the segment.

2.3.3 Segment Assembling

In order to adapt to different network conditions, it is highly desired that the selected mesh segments can be organized with priorities. In our research, we make use of the average normal vector to determine the priority of each segment. The basic idea is to compute the effective area from the viewer's point of view. In particular, for a segment C_i with an average normal vector m_i, we first calculate the total area S, the sum of all the p triangular areas within the segment, i.e. $S = s_1 + s_2 + ... + s_p$. Then, the effective area for segment C_i is simply computed as $S_e = S \times (m_i \cdot \vec{V})$. Fig. 2.7 shows one example.

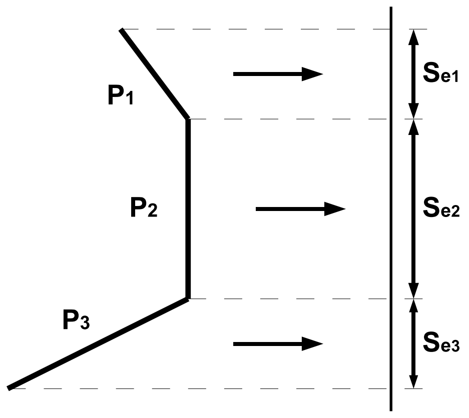


Figure 2.7: Three segments with effective area S_{e1}, S_{e2}, and S_{e3}.

Moreover, in order to take the compression cost into consideration, the calculated effective area is further normalized as $T_e = \frac{S_e}{P}$, where P stands for the compressed segment size. The segments with larger values of T_e are given higher priorities, and all the segments are organized according to the order of T_e. This is very useful for bandwidth-limited channels. For example, when there is no enough bandwidth, we can simply discard those segments with low priorities. In other words, given a certain bandwidth, we try to transmit as much as possible from the viewer's point of view.

2.4 Optimal Number of Segments

In addition to mesh segmentation and segment selection, another fundamental problem is how to determine the optimal number of segments in mesh segmentation so that the average view-dependent mesh transmission size can be minimized. Let n denote the number of segments. As we know, when n is small, the average size of a segment is large. The bandwidth will be severely wasted in the case that only a small portion of a segment is visible since we have to transmit the entire segment. On the other hand, when n is large, the average size of segment is small and the transmission of unnecessary invisible portions can be largely avoided. However, large n leads to more segment boundary duplication and thus increases the compression size. Therefore, the relationship between the number of segment and the average transmission size is not strictly increasing or decreasing.

In the following, we propose analytical models to estimate the overall compression size S and the the average transmission size S_t under different numbers of segments. Although the derivation is not rigorous, our simulation verifies the accuracy of the proposed models.

2.4.1 Overall Compression Size

Denote $|T|$ and $|V|$ as the total number of triangles and the total number of vertices in the original mesh. Assuming the mesh is uniformly segmented, each segment averagely contains $\frac{|T|}{n}$ triangles and the $|V|$ vertices are distributed into the n segments with some redundancies. In general, compared with connectivity information, geometry information, which is composed of vertices coordinates, contributes much more in compression size. Thus, we can deem that the compression size is approximately proportional to the number of vertices and relevant to the coding scheme. Mathematically, we express the overall compression size as

$$S(n) = a|V'|, \tag{2.8}$$

where a is a constant for a certain coding scheme and $|V'|$ is the total number of vertices in all the segments, different from $|V|$ in that it takes into account the overlap boundary vertices. Note that when there is only one segment ($n = 1$), $S(1) = a|V|$.

$|V'|$ can be approximately derived by estimating the average number of vertices on a segment boundary. In particular, considering $|V|$ vertices within the total $|T|$ triangles, the density of vertices is $\frac{|v|}{|T|}$ and the density along a line is $\sqrt{\frac{|V|}{|T|}}$. For a segment with $\frac{|T|}{n}$ triangles, the boundary perimeter length of the segment is proportional to $\sqrt{\frac{|T|}{n}}$. Thus, the average number of vertices along the segmentation boundary is proportional to $\sqrt{\frac{|V|}{n}}$.

Assuming each boundary vertex appears twice (for most of the cases), the total vertices in all the segments can be approximately derived as

$$|V'| = |V| + a_1\sqrt{\frac{|V|}{n}} \times n = |V| + a_1\sqrt{n|V|}. \tag{2.9}$$

Substituting Eq. (2.9) back to Eq. (2.8), we obtain

$$S(n) = a|V| + a_2\sqrt{n|V|}. \tag{2.10}$$

The expression can be further simplified as

$$\frac{S(n)}{S(1)} = 1 + c_1\sqrt{n}. \tag{2.11}$$

Note that by off-line performing the segmentation for $n = 1$ and another sample value of n, we can easily obtain $S(1)$ and c. Then, the established model can be used to estimate the total compression size under any other number of segments.

2.4.2 Average Transmission Size

In order to calculate the average transmission size, we need to find out the average number of visible segments. On one extreme, when there is only one segment, the viewer can see it in all views. On the other extreme, when the number of segments approach to infinite, on average the viewer can see half of the segments from any view. Note that here we do not consider the viewing frustum constraint since it only scales down the average transmission size and does not affect the optimal number of segments. Therefore, for a given number of segments, the percentage of visible segments is somewhere between 0.5 and 1.

To make the problem simpler, we assume that the normal vectors for each triangle are uniformly distributed. In other words, we assume that in each view there are approximately the same number of triangles. By performing the Gauss map on a 3D mesh, we uniformly distribute all the triangles onto a sphere.

In general, only half of the sphere can be seen from a particular view, but we need to consider those boundary segments, which usually contain large invisible portions. Assuming n is large (no less than 20), each segment is small enough and we can deem that each boundary segment has only a small visible portion. In this way, the total transmission area in the sphere can be expressed as the sum of half of the sphere surface area and the band near the equator shown in Fig. 2.8, i.e. $S_v = 2\pi r^2 + 2\pi r d$, where r is the radius of the sphere and d is the average diameter of a segment. Since each segment on average has an area of $\frac{4\pi r^2}{n}$ in the sphere, the diameter of a segment should be proportional to $\sqrt{\frac{4\pi r^2}{n}}$. Thus, the average number of visible segments can be derived as

$$N(n) = \frac{S_v}{\frac{4\pi r^2}{n}} = \frac{n}{2} + c_2\sqrt{n}. \tag{2.12}$$

Finally, the average transmission size becomes

$$\begin{aligned} S_t(n) &= \frac{S(n)}{n} N(n) \\ &= S(1)(\frac{1}{2} + c_1 c_2 + \frac{c_2}{\sqrt{n}} + \frac{c_1\sqrt{n}}{2}), \end{aligned} \tag{2.13}$$

where the parameters can be obtained through off-line measuring a few (S_t, n) sampling pairs. Based on the model in Eq. (2.13), we can easily derive that the minimal average transmission size is achieved at $n = \frac{2c_2}{c_1}$, which is the optimal number of segments.

2.5 Simulation Results

2.5.1 Results of Mesh Segmentation

In this section, we evaluate our proposed criteria for mesh segmentation. We first verify the normal similarity criterion. The four mesh models listed in Table 2.1 are used as test models. Fig. 2.9 shows the mesh segmentation results. It can be seen that triangles with similar normal directions are grouped together and the segments are relatively flat, which demonstrate the effectiveness of the normal similarity criterion.

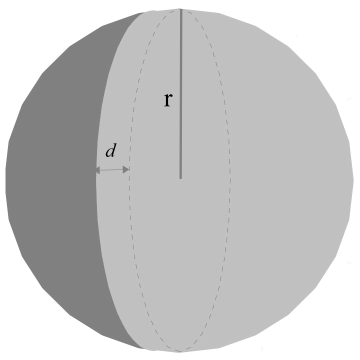


Figure 2.8: The dark part is the invisible segments while the part left is to be transmitted.

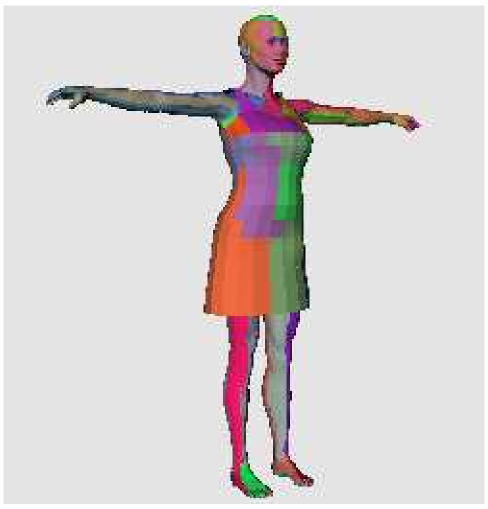
Table 2.1: The comparison of different viewing frustum based segment selection algorithms.

model	#. of vertices	#. of triangles	#. of segments	#. of selected segments [YKK05b] alg.	our alg.
Woman	18,207	35,713	300	132.8	123.1
Bunny	34,834	69,451	400	175.4	149.9
Horse	48,485	96,966	500	218.1	193.4
Head	160,940	316,948	700	322.7	284.4

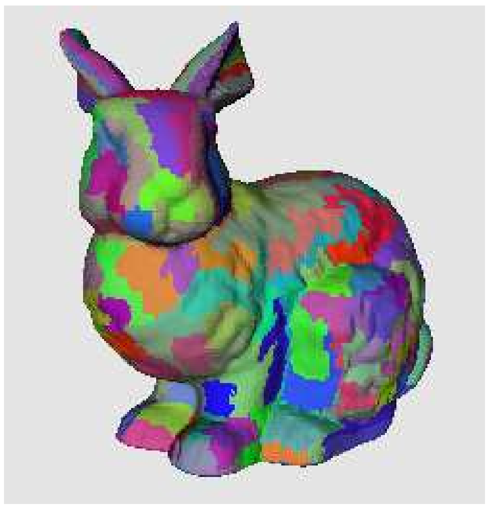
We then compare the cases with and without the additional redundancy criterion. As shown in Fig. 2.10, the algorithm with the additional redundancy criterion achieves better compression performance, up to about 10 kbytes reduction in compression size. The gain is more prominent for relatively larger models and in general grows with the increased number of segments.

We further compare the cases with and without the additional uniformity criterion. The results are shown in Fig. 2.11. It can be seen that this third criterion indeed results in segments with relatively uniform sizes. For example, without the criterion the segment on torso (orange-red color) in Fig. 2.11(a) is much larger than other segments while it is being divided when using the additional uniformity criterion.

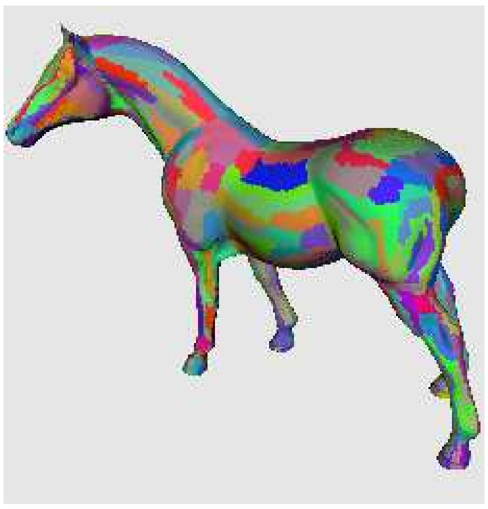
Note that in the following experiments, for simplicity we only use the first two segmentation criteria with the same weight and set the third criterion weight to zero.



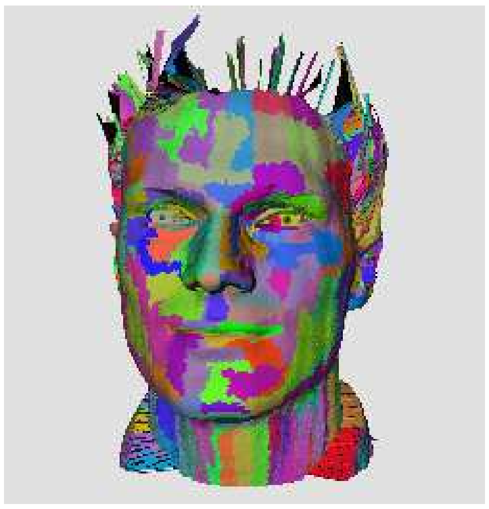
2.9.a: Woman



2.9.b: Bunny

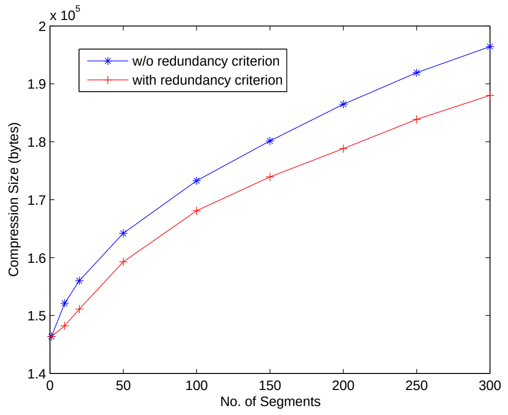


2.9.c: Horse

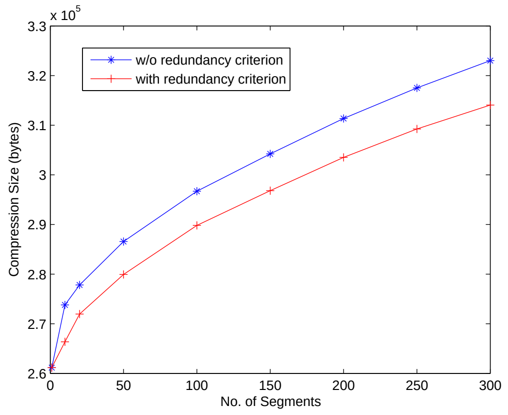


2.9.d: Head

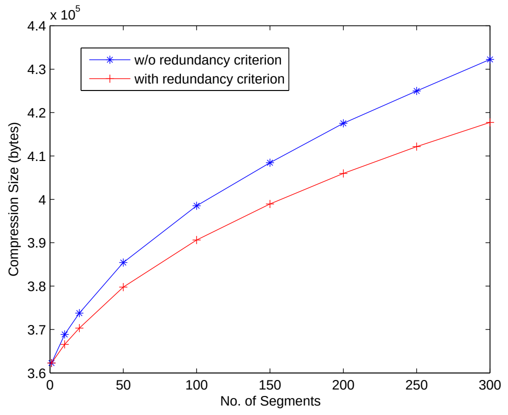
Figure 2.9: The segmentation results using the proposed normal similarity criterion (all segments are used to show the results). The numbers of segments for each model are given in Table 2.1.



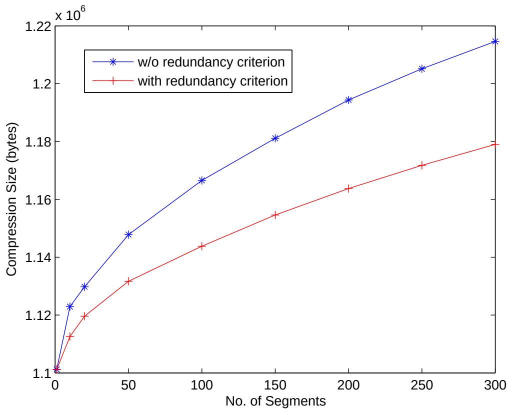
2.10.a: Woman



2.10.b: Bunny

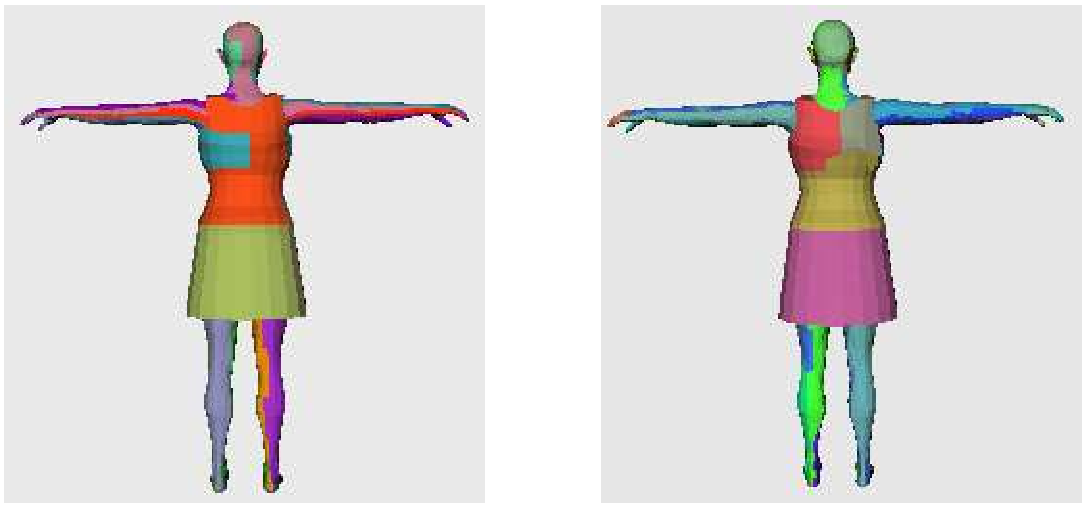


2.10.c: Horse



2.10.d: Head

Figure 2.10: The segmentation results with or without the additional redundancy criterion.



2.11.a: without uniformity criterion

2.11.b: with uniformity criterion

Figure 2.11: The segmentation results using the additional uniformity criterion.

2.5.2 Results of Segment Selection

To simplify the simulations, we only construct the top and bottom planes of the view frustum and assume the other four planes (near, far, left and right) have no limitation on segment visibility. In particular, let O be the center of the model, and M and N are the midpoints from the center to the topmost point and the bottommost point respectively, as shown in Fig. 2.12. We define the viewing point V on the horizontal plane passing through center O and perpendicular to vector MN. The length of VO is set to twice of the length of MN and these two vectors define a vertical plane. The projects of the two culling planes (top and bottom) of the view frustum on the vertical plane are labelled as VP and VQ in Fig. 2.12, and the dashed line indicates the range of the target model on the vertical plane. After fixing a view point, the variation of the view angle is through randomly selecting a point C on MN and moving the crossing points A and B accordingly to change the two culling planes of the viewing frustum.

We compare our proposed view frustum based segment selection algorithm with the

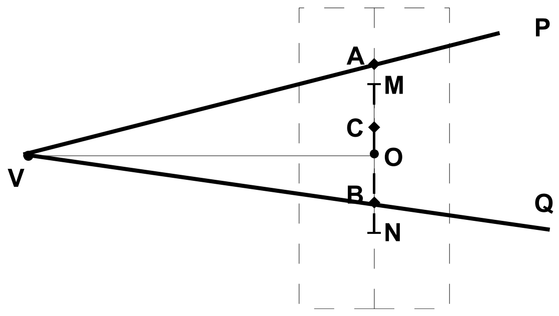


Figure 2.12: The construction of viewing frustum.

one proposed in [YKK05b], which uses a sphere instead of a box to bound a segment. The comparison results are shown in Table 2.1. Note that in the number of selected segments, all the visible segments are being selected by both algorithms while our algorithm chooses less invisible segments than the one in [YKK05b], which demonstrates that our proposed algorithm is more efficient. This is mainly because the bounding box we use can enclose each segment more tightly.

2.5.3 Results of View-dependent 3D Mesh Transmission

We compare our proposed scheme with the one without any segmentation preprocess, i.e. using 3DMC to code the entire 3D mesh model. Table 2.2 summarizes the coding and transmission results. In the table, we use the ratio between the total size of our proposed algorithm and the total size of 3DMC, i.e. B/A, to indicate the overhead of segmentation. The overhead is mainly because we code each segment independently and the boundary vertices have to be counted more than once. However, compared with 3DMC, on average our proposed algorithm requires about 50% less bits for a particular view. This bandwidth saving is indicated by the C/A values in the table, which are the average values over 20 randomly selected view angles.

In addition to the bandwidth saving, our proposed framework also supports progressive transmission and rendering as described in Section 2.3.3. Fig. 2.13 shows such an example. In particular, the entire bunny model is divided into 380 segments. In

Table 2.2: The results of 3D mesh coding and transmission.

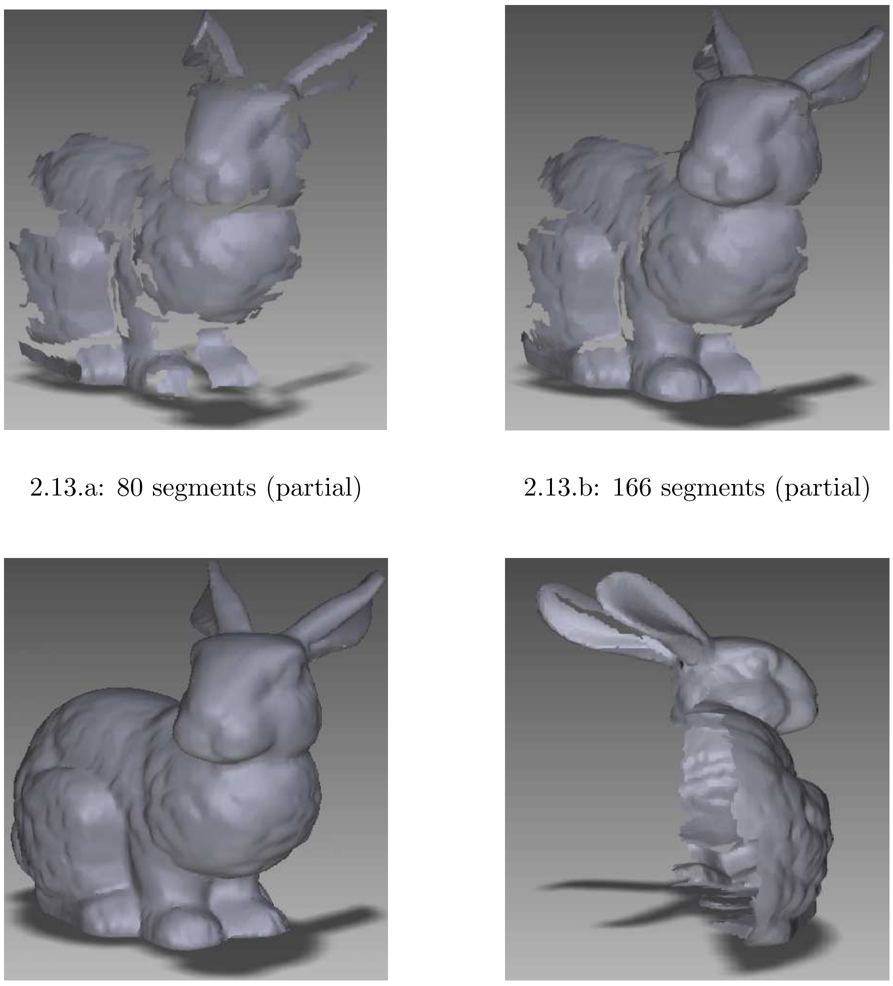
model	3DMC	proposed algorithm				comparison	
	total size (A) (bytes)	#. of seg.	total size (B) (bytes)	#. of trans. seg.	trans. data (C) (bytes)	B/A	C/A
woman	146,385	270	193,897	73	74,845	1.325	0.511
bunny	261,180	380	328,694	138	121,710	1.258	0.466
horse	364,269	450	450,923	146	167,993	1.238	0.461
head	1,101,173	800	1,287,979	373	492,615	1.170	0.447

Fig. 2.13(a), the most contributable 80 segments are transmitted, based on which the client can basically recognize the shape. With the 166 segments in Fig. 2.13(b), the model becomes clearer. With the 226 segments in Fig. 2.13(c), the complete view-based model is received, and the side view is displayed in Fig. 2.13(d) to show the non-transmitted parts.

2.5.4 Results of Optimal Number of Segments

We first verify the discovered relationship between the overall compression size and the number of segments. Fig. 2.14 compares the real compression sizes with the estimated compression sizes using Eq. (2.11) under different numbers of segments. We can see that the estimated results match the measured results very well, especially when the number of segments is more than 20. This demonstrates the accuracy of our proposed model.

We further verify the derived relationship between the average transmission size and the number of segments described in Eq. (2.13). Fig. 2.15 shows the results for the bunny model. Although the theoretical results are not exactly equal to the real results, they still match quite closely, especially at the region around the minimum average transmission size. Thus, this demonstrates that using our proposed theoretical model is sufficient to find the optimal number of segments.



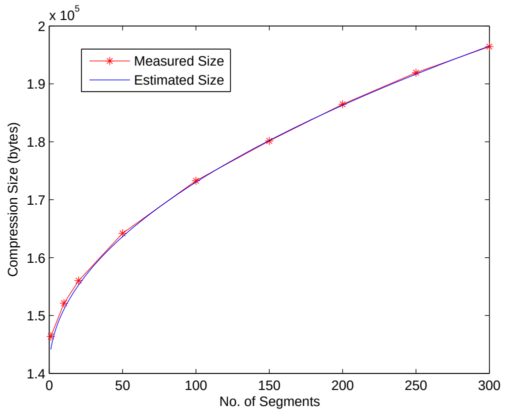
2.13.a: 80 segments (partial)

2.13.b: 166 segments (partial)

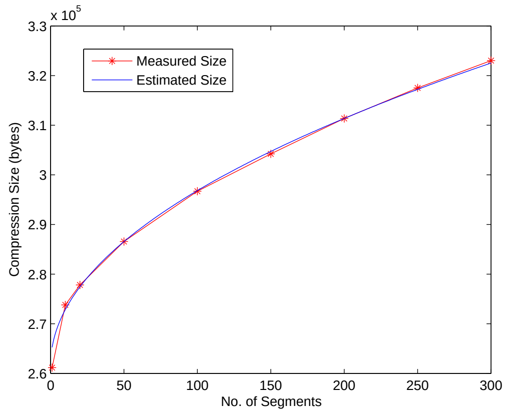
2.13.c: 226 segments (full)

2.13.d: the side-view of result (c)

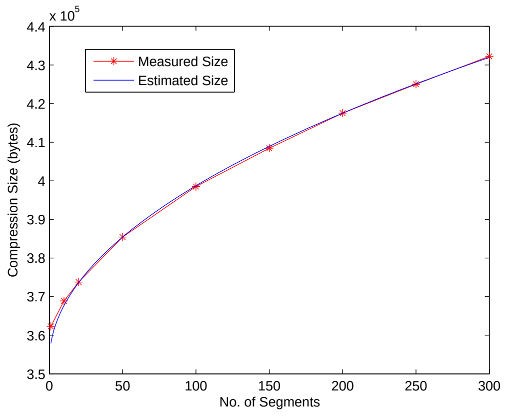
Figure 2.13: An example to illustrate the view-dependent transmission process for bunny.



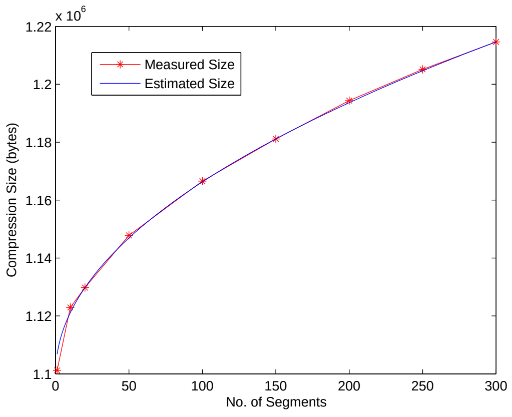
2.14.a: Woman



2.14.b: Bunny



2.14.c: Horse



2.14.d: Head

Figure 2.14: The results of the overall compression sizes under different number of segments.

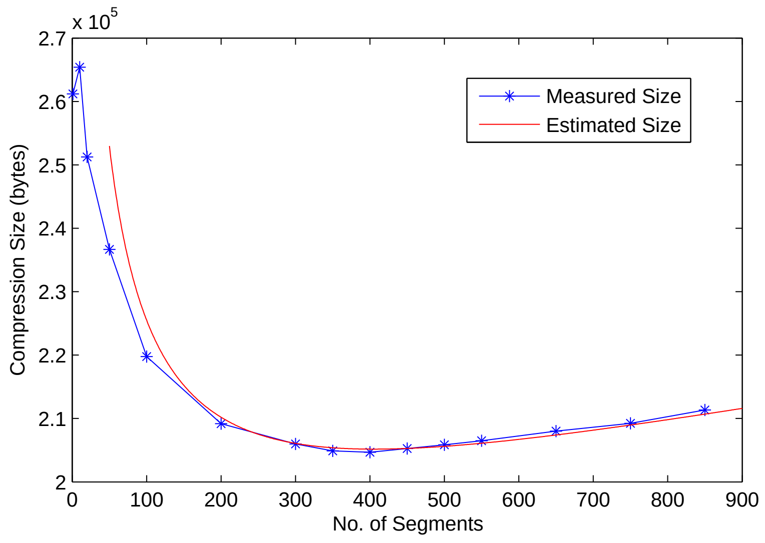


Figure 2.15: The results of the average transmission sizes under different numbers of segments for the bunny model.

Chapter 3

Wavelet-Based Progressive 3D Mesh Transmission

3.1 Background

Considering the time-varying characteristic of wireless channels, progressive mesh coding is highly desired. With progressive compression techniques, a complex 3D mesh model only needs to be encoded once and can be transmitted and decoded at multiple bit rates. In literature, many progressive mesh compression schemes have been proposed including [Hop98, TGHL98, PR00, LK98, KSS00]. In our research, we consider progressively compressing triangular meshes. A triangular mesh is typically irregular in topology. The latest progressive mesh coding technique is to covert irregular mesh into semi-regular meshes, which have regular vertices (valence 6) almost everywhere except for those vertices that are related to the control points in the coarsest level mesh. Such a conversion process is called remeshing. For example, Schroder *et al.* provides an efficient remeshing technique using the MAPS algorithm [LSS$^+$98]. In that scheme, an irregular mesh is first simplified into a base mesh. Each triangle in the original mesh can be mapped into an "internal" triangle within a base triangle. Then, the base mesh is subdivided, and the new vertices obtained through subdivision are mapped back to the vertices in the original mesh. Finally, the base mesh and these mapped vertices compose of a semi-regular mesh.

In addition, the generated semi-regular mesh can be efficiently compressed using wavelet based image coding schemes [KSS00].

However, unlike wavelet coefficients in image compression, which are independent of each other, the vector wavelet coefficients generated during remeshing are dependent. Such a characteristic of semi-regular meshes is not being considered in most of the existing progressive mesh coders. In this research, we propose an improved wavelet based 3D mesh coder. The basic idea is to introduce a preprocessing step to scale up the vector wavelets generated in remeshing so that the inherent dependency of wavelets can be truly understood by the zerotree-like image compression algorithms. Although the idea is simple, the weights used in the scaling process are obtained by thoroughly analyzing the distortions of wavelets at different refinement levels. Experimental results show that compared with the state-of-the-art wavelet based 3D mesh compression scheme, our proposed mesh coder can achieve significant quality improvement with only slight complexity increase.

Moreover, based on the generated semi-regular mesh that has subdivision connectivity, Sim *et al.* further developed a view-dependent mesh streaming system [SKKL05]. The system consists of three parts: preprocessing, mesh partitioning, and bit optimization for progressive streaming. In preprocessing, an irregular mesh is converted into a semi-regular mesh using the MAPS algorithm. Then, the mesh is partitioned into many segments. Each segment is progressively encoded using the SPIHT algorithm. Finally, the system optimally allocates bits for each segment according to the given viewing direction and progressively transmits them to the receiver side. Hoppe [Hop97] also proposed a view-dependent refinement algorithm, where a mesh is represented as PM (progressive mesh). Initially, a coarsest mesh is rendered. Then the algorithm iteratively checks each vertex whether it needs to be split (refined) or not. A vertex will only be refined if it is within the viewing frustum, facing towards the viewer, and the screen-space error is larger than a predefined threshold.

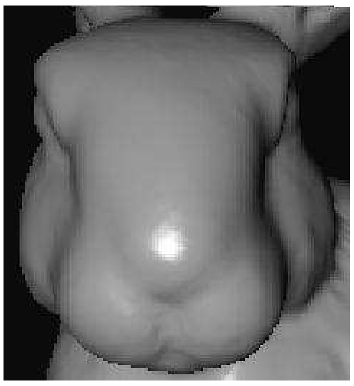
Although the view dependency is considered in both [Hop97] and [SKKL05], they did not take into account the illumination effects, which results in rendering the perceptually unimportant dark portions and thus wastes the transmission bandwidth. In this chapter, we further propose an illumination and view dependent 3D mesh transmission system. In particular, we propose a novel distortion model, which consists of both illumination distortion and geometry distortion. In addition, in order to calculate the rate and distortion in real time, we further propose a simplified distortion model. Experimental results show that significant gain can be achieved when the illumination effects are being considered in addition to the view dependency.

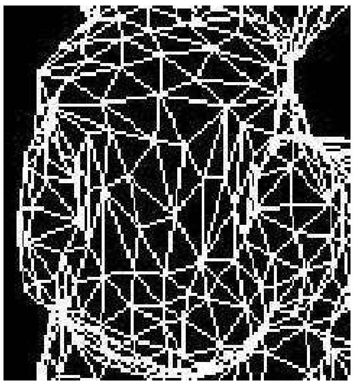
3.2 Wavelet Based Progressive Mesh Compression

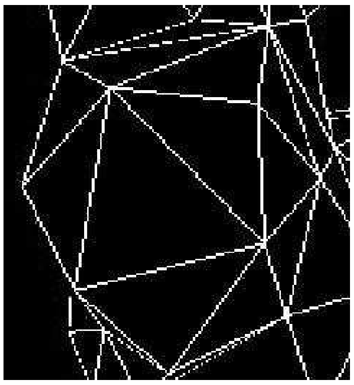
3.2.1 Existing Wavelet based Progressive Mesh Coding

In this section, we review the common process in existing wavelet based progressive mesh compression, particularly using the compression scheme described in [SKKL05] as an example.

A typical wavelet based progressive mesh coder consists of two major components: remeshing and coding. In [SKKL05], the remeshing component applies the MAPS algorithm [LSS$^+$98] to convert an irregular triangular mesh into a semi-regular mesh with subdivision connectivity, as illustrated in Fig. 3.1. In particular, the original mesh is simplified into a base mesh and the base mesh is then refined into a semi-regular mesh through a series of subdivision and vertices adjustments. The butterfly subdivision (see Fig. 3.2(a)) is used in [SKKL05] to predict the next level vertices from the current level. The prediction errors, i.e. the difference between original vertices and predicted vertices, are treated as wavelets (see Fig. 3.2(b)). In summary, the remeshing process generates a base mesh and a sequence of wavelets, based on which the semi-regular mesh can be fully reconstructed.


3.1.a: solid face


3.1.b: irregular mesh


3.1.c: base mesh

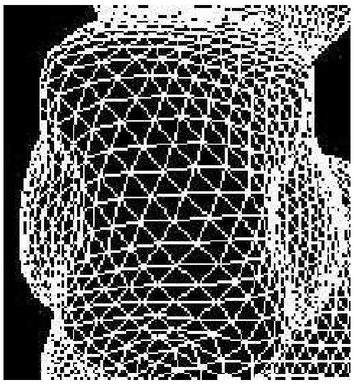
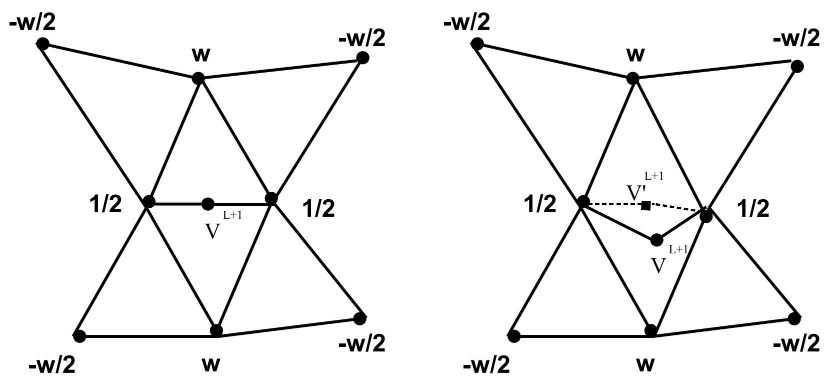

3.1.d: semi-regular mesh

Figure 3.1: The remeshing process of bunny head.



3.2.a: butterfly 3.2.b: wavelet

Figure 3.2: The butter-fly subdivision. In (b), the wavelet is the displacement between original and predicted vertex position, i.e. $V^{L+1} - V'^{L+1}$.

In the coding process the base mesh and the wavelets are encoded separately. Typically, the base mesh is encoded losslessly by a single-rate mesh compression scheme such as the MPEG-4 3DMC algorithm while the wavelets are encoded progressively by a wavelet-based image coder such as the zerotree coder [Sha93] or the SPIHT coder [SP96]. In particular, in [SKKL05], vertices are organized into edge-based trees to have the hierarchical parent-offspring relationship so that zerotree-like coders can be applied. Each vertex on base mesh edges becomes a root node of an edge-based tree. Fig. 3.3 shows the structure of a edge-based tree. In this way, all the vertices can be covered by edge-based trees without overlapping.

Note that a wavelet here is the displacement between the predicted vertex and the corresponding vertex on the original model. In other words, each wavelet is actually a displacement vector $(\Delta x, \Delta y, \Delta z)$. In [KSS00], three independent zerotree coders are applied to encode the three components separately. In [SKKL05], the SPIHT algorithm is applied to first encode the scalar of $sgn(\Delta x) \cdot \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2}$, followed by the encoding of Δy and Δz if the scalar is significant.

Through the above review, we can see that the existing wavelet based 3D semi-regular

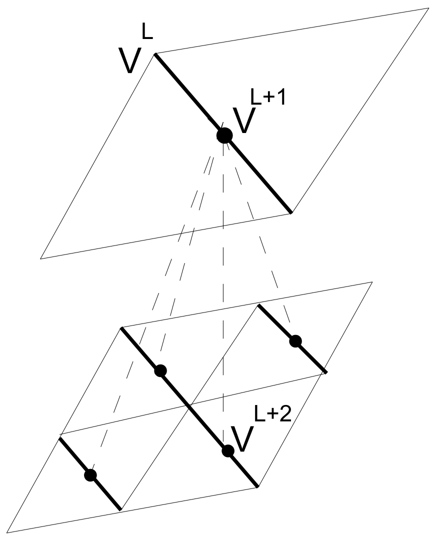


Figure 3.3: An example of an edge-based tree, where V^{L+2} is one of the four offsprings of V^{L+1}.

mesh coders directly extend the state-of-the-art wavelet-based image coders to compress the wavelet vectors generated in the remeshing process. The main problem of such type of approaches is that the particular properties of semi-regular meshes are not being taken into consideration. Specifically, in image compression, wavelet coefficients of an image at different levels are independent of each other. A distortion of a coefficient at a coarser level does not affect the distortion of any other coefficient at a finer level. However, in 3D semi-regular mesh compression, the situation becomes totally different. That is the wavelets in the lth level affect the vertices positions in the $(l+1)$th level and the subsequent levels due to the subdivision operations. Thus, with the same error, a wavelet at a lower level contributes more to the total distortion than a wavelet at a higher level. In other words, a wavelet at a lower level is more important than a wavelet at a higher level.

It is well known that the principle of zerotree-like coders is to send the higher order bits of the larger magnitude coefficients first. Directly applying zerotree-like coders to encode semi-regular meshes only consider the wavelet magnitudes without taking into account the dependency and the inherent unequal importance among different levels of wavelets. Thus, the generated progressive bitstream is not embedded since it is not guaranteed that the earlier portions of the progressive mesh bitstream is always more important than the later parts.

3.2.2 Proposed Wavelet based Progressive Mesh Coding

Fig. 3.4 shows the diagram of our proposed wavelet based progressive mesh coder. Basically, we add one new component, "Scaling", between the remeshing component and the coding component. The purpose of the scaling component is to give different weights to wavelets in different levels so that their magnitudes can truly represent their importance. One advantage of such a system is that it can directly employ the exiting remeshing and zerotree-like coding algorithms.

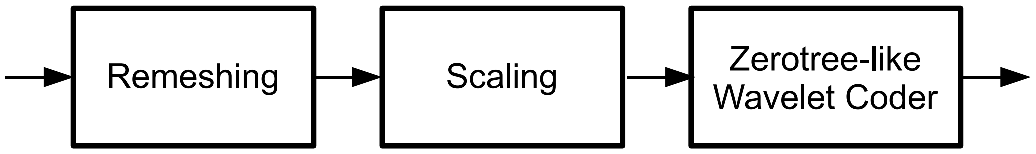


Figure 3.4: The diagram of our proposed wavelet based progressive mesh coder.

The key question in our proposed mesh coder is how to set the weights for different wavelets. In order to address this question, we need to analyze the distortion. In our research, we consider using geometric distortion to measure the reconstructed mesh quality. The geometry distortion is typically defined as the mean square error between original vertices and reconstructed vertices, i.e.

$$D(M', M) = \frac{1}{|V|} \sum_{v_i \in M, v_i' \in M'} \|v_i - v_i'\|^2$$

where $|V|$ is the total number of vertices, and v_i and v_i' denote the vertices in the original semi-regular mesh M and the reconstructed semi-regular mesh M', respectively.

Since in the progressive representation vertices are represented by wavelets, it is highly desired to compute the distortion through wavelets so that repeatedly reconstructing the vertices can be avoided. As pointed out in [SKKL05], the mean square error of vertices $\|v_i - v_i'\|^2$ can be approximated by the mean square error of wavelets $\|w_i - w_i'\|^2$. However, they are not exactly equal due to the dependency among the vertices at different levels.

In [SKKL05], a distortion model was proposed, which considers the error propagation effect due to subdivision. In particular, as shown in Fig. 3.5(a), the distortion for the center solid circle in the lth level will propagate to the other three types of vertices in the $(l+1)$th level with the weights of $1/2$, w and $-w/2$, respectively. Considering the valence of six feature for a semi-regular mesh, the error propagation effect (including itself) from a particular level to the next level is approximated by a weighting factor [SKKL05]

$$\begin{aligned} W &= 1^2 + 6 \times (\frac{1}{2})^2 + 6 \times w^2 + 12 \times (-\frac{w}{2})^2 \\ &= 2.5 + 9w^2 \end{aligned} \tag{3.1}$$

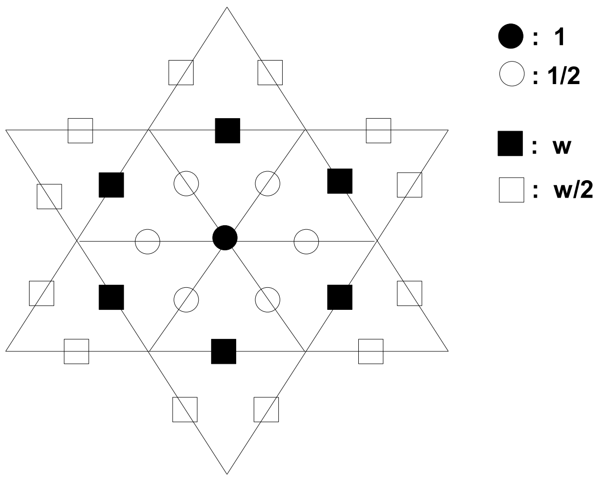
where w ranges from 0 to $1/8$. Further taking into account the error propagation among multiple levels, the weight for a distortion in the lth level is defined as [SKKL05]

$$W_l = W^{L-l}, \quad l = 1, 2, \ldots, L \tag{3.2}$$

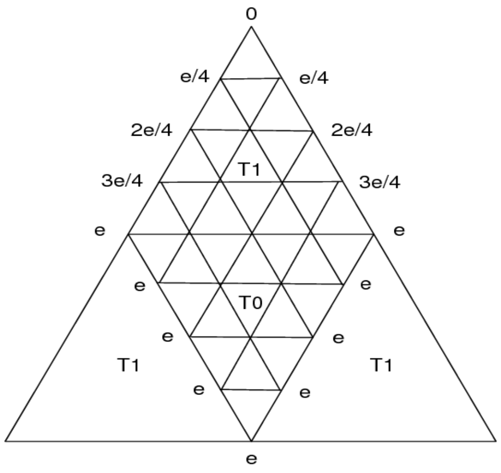
where L is the total number of subdivision levels.

Note that although the authors in [SKKL05] did consider the dependency among the wavelets at different levels in their distortion model, they only applied it for the distortion estimation and did not use it to improve the coding performance. Moreover, their derived weights shown in Eq. (3.1) and (3.2) are not accurate. In the following, we present our developed distortion model.

We first consider the simplest case, i.e. $w = 0$, and assume all the wavelets at level 1 (base mesh is at level 0) have an uniform error e (see Fig. 3.5(b)). We study how the



3.5.a: error propagation



3.5.b: wavelet errors after three times subdivision

Figure 3.5: The error propagation weights and wavelet errors.

distortions at level 1 are propagated to subsequent levels. In particular, considering a vertex on an edge is shared by two adjacent triangles, it is clear that the distortion at level 1 for this particular triangle is $3e^2/2$. As shown in Fig. 3.5(b), after the first level subdivision, the base triangle is divided into two types of triangles: T_0 and T_1. It can be observed that the vertices generated at different subdivision levels in T_0 always have the same error e while the new vertices in T_1 are of different wavelet errors, having a pattern of $(k+1)$ vertices with an error of $(k/2^{L-1})e$, $k = 1, 2, \ldots, L$. Adding all the distortions together and removing the overlapping, we derive the total distortion for this particular triangle as

$$d_L = (\frac{5}{16}4^L + \frac{1}{4})e^2 \tag{3.3}$$

It also means that the distortion of $3e^2/2$ introduced at level l will result in a total distortion of d_{L-l+1}. Therefore, the weight for a distortion at the lth level becomes

$$W_l = \frac{d_{L-l+1}}{3e^2/2} = \frac{5}{6}4^{L-l} + \frac{1}{6}, \quad l = 1, 2, \ldots, L \tag{3.4}$$

Finally, the overall distortion is calculated as

$$D = \sum_{l=1}^{L} \sum_{w_i \in M^l} W_l \cdot \|w_i - w_i^{'}\|^2 \tag{3.5}$$

where $w_i \in M^l$ means that w_i is used in the lth level refinement.

We would like to point out that although the weight in Eq. (3.4) is derived in the case with the subdivision parameter $w = 0$, it can be directly used for the other cases with $w \neq 0$. This is because w ranges from 0 to 1/8 and the corresponding error propagation is insignificant as shown in Eq. (3.1).

3.3 Progressive Transmission System

Here, we consider applying the developed 3D mesh coder for progressive 3D mesh transmission. Our key idea in this research is to consider the view and illumination dependencies. Specifically, we try to avoid the transmission of the invisible portions and the perceptually unimportant dark or bright portions of a 3D model. In order to do this, a 3D mesh needs to be partitioned into many segments. However, unlike the view-dependent transmission scheme developed in Chapter 2, where a customized segmentation algorithm is developed, the adopted 3D mesh coder naturally organizes vertices into edge-based trees, each of which can be considered as a segment.

In particular, in our research, a semi-regular mesh is divided into two components: base mesh and edge-based forest. The vertices from the first time subdivision of the base mesh are taken as roots for each edge-based tree. Clearly, the number of trees in the forest is the same as the number of edges in the base mesh. We use the MPEG-4 3DMC (3D mesh coding) algorithm to encode the base mesh while employing our proposed wavelet-based progressive mesh coder to encode each edge-based tree separately. In this way, the base mesh is losslessly transmitted to the receiver, while the edge-based trees are progressively delivered according to the available bit budget.

Now, the key question is how to optimally allocate bits to each segment or tree so that the total available bit budget can be well utilized. In other words, we need to decide which segment should be refined at which level.

In order to achieve that, we need to develop the rate-distortion (R-D) function for each segment, which will be described in detail in next section. Suppose we have the R-D function for each segment, denoted as $D_k = f_k(R_k), k = 1, 2, ..., n$, where D_k and R_k are the distortion and the rate for the kth segment respectively, and n is the total number of segments, the bit allocation problem can be formulated as follows: given a certain bit budget R, how to find the optimal R_k values that minimize $\sum_{k=1}^{n} D_k = \sum_{k=1}^{n} f_k(R_k)$.

This is an n-dimension constrained optimization problem. A common solution is to use the Lagrange multiplier method. In particular, minimizing the distortion is the same as minimizing the following equation with some value of λ,

$$\sum_{k=1}^{n}(f_k(R_k) + \lambda R_k). \tag{3.6}$$

By differentiating the E.q. (3.6) with respect to $R_k, k = 1, 2, ..., n$, we obtain n equations: $f_k'(R_k) + \lambda = 0, k = 1, 2, ..., n$. Together with the constraint: $\sum_{k=1}^{n} R_k = R$, theoretically the $n + 1$ variables, i.e. λ and $R_k, k = 1, 2, ..., n$, can be derived. In practice, there is usually no derivable formula for the above optimization problem. Instead, the greedy algorithm is typically used to search the optimal solution in a discrete solution space.

3.4 Distortion Model and Its Simplification

There always exists a trade-off between quality and bit rate. For 3D mesh models, one of the key challenges is how to measure the quality. A popular metric is the Hausdorff distance, which measure the distances between geometric shapes. However, this type of metric is not the same as the perceived visual quality since it is defined purely from the geometry point of view. Several research studies have been conducted to find more appropriate quality metrics. In particular, Hoppe [Hop97] proposed a view-dependent mesh metric called screen-space geometric error, which measures the geometric error after projecting geometry models in the viewing direction. Luebke *et al.* [LE97] pointed out that the silhouette regions are more important than interior regions and employed a tighter screen-space error threshold in the simplification process. Reddy [Red97] proposed to select LOD (level of details) according to a fundamental perceptual model, and an updated version of the perceptual model was further proposed in [Red01]. Although the perceptual model is not comprehensive, it does achieve some desirable effects such as preserving silhouette boundaries and high contrast details. However, the perceptual model requires tremendous computations.

In our research, we define the overall distortion as a combination of geometry distortion and illumination distortion, i.e.

$$D = D^u + \lambda D^s, \tag{3.7}$$

where we use the screen-space distortion D^s to measure the geometry distortion, D^u denotes the illumination distortion, and λ is a user defined weight. In the following, we describe our developed models for D^u and D^s in detail.

3.4.1 Geometry Distortion

As we have discussed in Section 3.2, the geometry distortion is typically defined as the mean square error between original vertices and reconstructed vertices, i.e.

$$D^g(M', M) = \frac{1}{|V|} \sum_{v_i \in M, v_i^{'} \in M'} \|v_i - v_i^{'}\|^2$$

where $|V|$ is the total number of vertices, and v_i and $v_i^{'}$ denote original and reconstructed vertices in M and $M^{'}$ respectively.

Since in the progressive representation vertices are represented by wavelets, it is highly desired to compute the distortion through wavelets so that repeatedly reconstructing the vertices can be avoided. In Section 3.2, we derived that the overall geometry distortion could be estimated as

$$D^g(M', M) = \sum_{l=1}^{L} \sum_{w_i \in M^l} W_l \cdot \|w_i - w_i^{'}\|^2 \tag{3.8}$$

where $w_i \in M^l$ means that w_i is used in the lth level refinement.

Next, we consider the factor of viewing direction. In our research, we assume that the viewer is located at a far distance and thus orthogonal projection is used, i.e., the viewing direction is the same for all the vertices.

Let $\vec{V}$ denote the normalized viewing direction. We use the screen-space error metric, which calculates the geometric error projected on the screen. Specifically, the total view-dependent screen space error for the kth segment is

$$\begin{aligned} D_k^s &= \sum_{l=1}^{L} \sum_{v_i \in M_k^l} a_i \cdot |(v_i - v_i^{'}) \times \vec{V}|^2 \\ &= \sum_{l=1}^{L} \sum_{v_i \in M_k^l} a_i \cdot W_l \cdot |(w_i - w_i^{'}) \times \vec{V}|^2 \end{aligned} \tag{3.9}$$

where a_i is either 0 or 1, indicating whether the vertex is visible or not.

We can easily obtain a_i by checking the angle between the vertex normal $\vec{N_i}$ with the view direction $\vec{V}$. If the angle is less than $\pi/2$, the vertex is visible; otherwise, it is invisible. To simplify the computation, for all the vertices within one segment k, we assign the same normal direction $\vec{N^k}$ since the vertices in one segment are around the the neighborhood of a base mesh edge and their normal directions are close to each other. We calculate $\vec{N^k}$ as the average of two neighboring base triangle normals. In this way, the computation of a_i can be expressed as

$$a_i = \begin{cases} 1 & (\vec{N^k} \cdot \vec{V}) > 0; \\ 0 & (\vec{N^k} \cdot \vec{V}) \leq 0. \end{cases}$$

3.4.2 Illumination Distortion

In this study, we use the Blinn-Phong shading model as the illumination model (shown in Fig. 3.6). Let $\vec{L}$ be the normalized vector for a directional light source, and $\vec{N}_i$ be the vertex normal vector of v_i. The intensity I_i on the vertex v_i is given by

$$I_i = \begin{cases} I_a k_a + I_d k_d(\vec{N}_i \cdot \vec{L}) & (\vec{N}_i \cdot \vec{L}) > 0; \\ +I_s k_s(\vec{N}_i \cdot \vec{H})^m, & \\ I_a k_a & (\vec{N}_i \cdot \vec{L}) \leq 0; \end{cases}$$

where k_a, k_d and ks are the ambient, diffuse and specular material coefficients, I_a, I_d and I_s are the corresponding luminance, and $\vec{H}_i$, called the halfway vector, is defined as $\frac{\vec{L}+\vec{V}}{|\vec{L}+\vec{V}|}$.

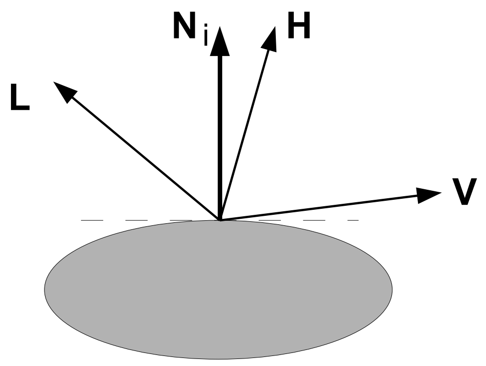


Figure 3.6: The Blinn-Phong shading model.

To calculate the illumination always involves calculating the normal vectors of vertices. The common way to compute vertex normal is through averaging the surrounding surface normals, which requires to reconstruct vertex positions. In order to avoid that, we propose another way for calculating vertex normal. The basic idea is to average surrounding edges. Specifically, we approximate a vertex normal as

$$\vec{N}_i = \frac{1}{6}\sum_{j=1}^{6}(v_i - v_{i,j}), \tag{3.10}$$

where $v_{i,j}, j = 1, 2, ..., 6$ are the six surrounding vertices for a vertex v_i. One problem of the above approximation is that the vertex normal is sensitive to the edge length. Thus, we further modify it as

$$\vec{N}_i = \frac{1}{6}\sum_{j=1}^{6}\frac{(v_i - v_{i,j})}{E_{i,j}} \tag{3.11}$$

where $E_{i,j}$ is the edge length of $(v_i - v_{i,j})$.

The vertex position error of v_i will affect the normal vector at v_i, and thus the illumination of its neighborhood will also be affected. We define the illumination distortion d_i at v_i as

$$d_i = b_i \cdot S_i \cdot (I_i - I_i^{'})^2 \tag{3.12}$$

where $I_i^{'}$ is the corresponding intensity for the reconstructed vertex $v_i^{'}$, S_i is the surrounding triangle area, i.e. $S_i = \sum_{j=1}^{6} S_{i,j}$, and b_i is to project S_i on the screen, which is defined as

$$b_i = \begin{cases} \vec{N^k} \cdot \vec{V} & (\vec{N^k} \cdot \vec{V}) > 0; \\ 0 & (\vec{N^k} \cdot \vec{V}) \leq 0. \end{cases}$$

Based on the illumination model in Eq. (3.10), the illumination error can be derived as follows:

$$\begin{aligned} I_i - I_i^{'} &= I_d k_d(\vec{N}_i \cdot \vec{L}) + I_s k_s(\vec{N}_i^{'} \cdot \vec{H})^m \\ &\quad - I_d k_d(\vec{N}_i \cdot \vec{L}) + I_s k_s(\vec{N}_i^{'} \cdot \vec{H})^m \\ &= I_d k_d[(\vec{N}_i - \vec{N}_i^{'}) \cdot \vec{L}] \\ &\quad + I_s k_s[(\vec{N}_i \cdot \vec{H})^m - (\vec{N}_i^{'} \cdot \vec{H})^m] \\ &\doteq I_d k_d[(\vec{N}_i - \vec{N}_i^{'}) \cdot \vec{L}] \\ &\quad + I_s k_s m[(\vec{N}_i - \vec{N}_i^{'}) \cdot \vec{H}](\vec{N}_i \cdot \vec{H})^{m-1}, \end{aligned} \tag{3.13}$$

where $\vec{N}_i^{'}$ is the corresponding normal for the reconstructed vertex $v_i^{'}$. According to

Eq. (3.11), we can derive the normal vector change as

$$\begin{aligned}\vec{N}_i - \vec{N}_i^{'} &= \frac{1}{6}\sum_{j=1}^{6}\frac{(v_i - v_{i,j}) - (v_i^{'} - v_{i,j})}{E_{i,j}} \\ &= \frac{1}{6}\sum_{j=1}^{6}\frac{v_i - v_i^{'}}{E_{i,j}} \\ &\doteq \frac{w_i - w_i^{'}}{E_i} \qquad (3.14)\end{aligned}$$

where $\frac{1}{E_i} = \frac{1}{6}\sum_{j=1}^{6}\frac{1}{E_{i,j}}$.

Substituting Eq. (3.14) back to Eq. (3.13) together with using $(\vec{N}^k \cdot \vec{H})^{m-1}$ to approximate $(\vec{N}_i \cdot \vec{H})^{m-1}$, and further substituting Eq. (3.13) back to Eq. (3.12), we derive

$$\begin{aligned}d_i &= b_i\frac{S_i}{E_i^2}\{c_1^2[(w_i - w_i^{'}) \cdot \vec{L}]^2 + c_2^2[(w_i - w_i^{'}) \cdot \vec{H}]^2 \\ &\quad +2c_1c_2[(w_i - w_i^{'}) \cdot \vec{L}][(w_i - w_i^{'}) \cdot \vec{H}]\} \qquad (3.15) \\ &= b_i\frac{S_i}{E_i^2}\{c_1^2[(w_i - w_i^{'}) \cdot \vec{L}]^2 + c_2^2[(w_i - w_i^{'}) \cdot \vec{H}]^2 \\ &\quad +2c_1c_2[(w_i - w_i^{'}) \cdot \vec{H}']^2 - c_1c_2|w_i - w_i^{'}|^2[1 - (\vec{L} \cdot \vec{H})]\}.\end{aligned}$$

where $\vec{H}' = \frac{\vec{L}+\vec{H}}{|\vec{L}+\vec{H}|}$, $c_1 = I_d \cdot k_d$ and $c_2 = I_s \cdot k_s \cdot m(\vec{N}^k \cdot \vec{H})^{m-1}$.

Finally, the total illumination distortion for the kth segment can be expressed as

$$D_k^u = \sum_{l=1}^{L}\sum_{v_i \in M_k^l} W_l \cdot d_i, \qquad (3.16)$$

where W_l is to compensate the approximation of $\|v_i - v_i^{'}\|^2$ by $\|w_i - w_i^{'}\|^2$.

3.4.3 Simplified Distortion Model

For practical applications of 3D mesh transmission, given the viewing and lighting parameters, we need to preform optimal bit allocation among different segments in real-time. Our previous developed distortion model described by Eq. (3.9) and Eq. (3.16) are too complex to compute. This is because for one set of viewing and lighting parameters, we need to compute the distortions for each vertex under different bit rates, which is impractical. Therefore, in this subsection, we further develop a simplified distortion model. The basic idea is to separate the wavelet distortion from the viewing and lighting parameters. In particular, we approximate any dot product in the form of $[(w_i - w_i^{'}) \cdot \vec{A}]^2$ into $|w_i - w_i^{'}|^2 \cdot (\vec{N^k} \cdot \vec{A})^2$, where $\vec{A}$ denotes a vector and $\vec{N^k}$ is the normal vector for the kth segment.

With such a simplification, the screen-space distortion for the kth segment becomes,

$$D_k^s = a_k|\vec{N^k} \times \vec{V}|^2 \sum_{l=1}^{L} \sum_{v_i \in M_k^l} W_l |w_i - w_i^{'}|^2. \tag{3.17}$$

The illumination distortion for the kth segment becomes,

$$\begin{aligned} D_k^u = b_k\{ & c_1^2(\vec{N^k} \cdot \vec{L})^2 + c_2^2(\vec{N^k} \cdot \vec{H})^2 + 2c_1c_2(\vec{N^k} \cdot \vec{H}')^2 \\ & -c_1c_2[1 - (\vec{L} \cdot \vec{H})]\} \cdot (\sum_{l=1}^{L} \sum_{v_i \in M_k^l} W_l \frac{S_i}{E_i^2} |w_i - w_i^{'}|^2), \end{aligned} \tag{3.18}$$

where a_k and b_k are defined in Eq. (3.10) and (3.13).

To compute S_i and E_i, we make use of the two base triangles and their edges since in the one-to-four subdivision the shape of the four off-spring triangles in the next level is similar to that of the two base triangles. In particular, we denote the two base triangle areas as $S_{k,1}$ and $S_{k,2}, and$the surrounding edge length as $E_{k,j}, j = 1, 2, .., 6$, as shown in Fig. 3.7. For a particular vertex v_i, we have the following approximations: $\frac{1}{E_i} = \frac{1}{6}\sum_{j=1}^{6} \frac{1}{E_{k,j}} \times 2^l$ and $S_i = \frac{S_{k,1}+S_{k,2}}{2} \times \frac{1}{4^l}$. Thus, we derive

$$\frac{S_i}{E_i^2} = \frac{S_{k,1} + S_{k,2}}{72 \times (\sum_{j=1}^{6} \frac{1}{E_{k,j}})^2}, \tag{3.19}$$

which is a constant for all the vertices within one segment.

Based on the developed simplified distortion model, for each segment we only need to calculate $\sum_{l=1}^{L} \sum_{v_i \in M_k^l} W_l |w_i - w_i^{'}|^2$ once, which is independent of view and lighting

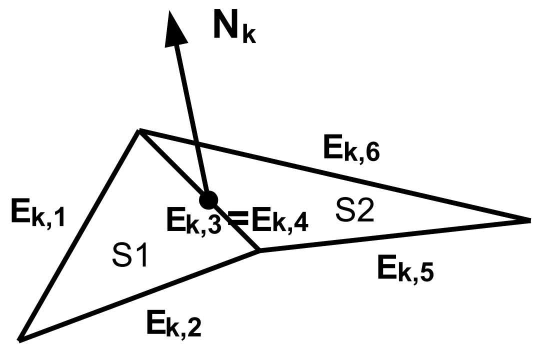


Figure 3.7: The estimations of edge length and triangle area.

parameters and can be pre-computed. During the online transmission, we only need to calculate the view and lighting dependent portions in Eq. (3.17) and (3.18). In this way, the view and lighting dependent transmission can be achieved in real time.

3.5 Simulation Results

3.5.1 Results of Progressive Mesh Compression

We implement the proposed wavelet based 3D mesh coder by adopting the MAPS algorithm [LSS$^+$98] for remeshing and SPIHT for encoding wavelet coefficients. We use the two 3D graphics models, "Venus" and "Armadillo", as the test models.

First, we evaluate the accuracy of our proposed distortion model. Fig. 3.8 shows the results of the distortion estimation, where our proposed distortion model uses the derived weight given in Eq. (3.4) while the previous model in [SKKL05] uses the weight given in Eq. (3.1). Note that all the distortion results here are normalized with respect to the distortion of the base mesh. It can be seen that the estimated distortions by our proposed model closely match the actual distortions at each bit rate, much more accurate than using the previous model. This is mainly because we assign more accurate weights to the distortions at each level.

Second, we compare the wavelet based 3D mesh coders with and without the scaling

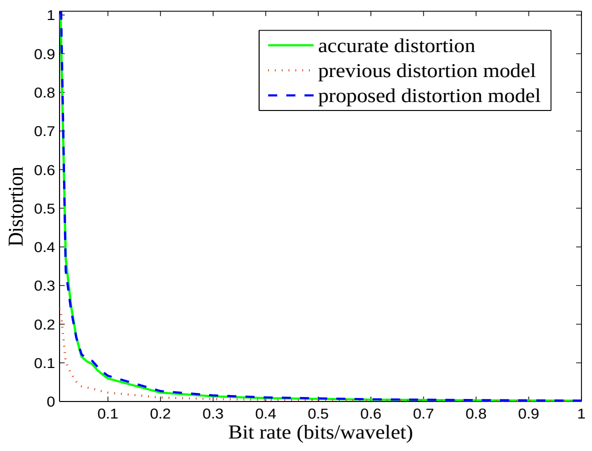
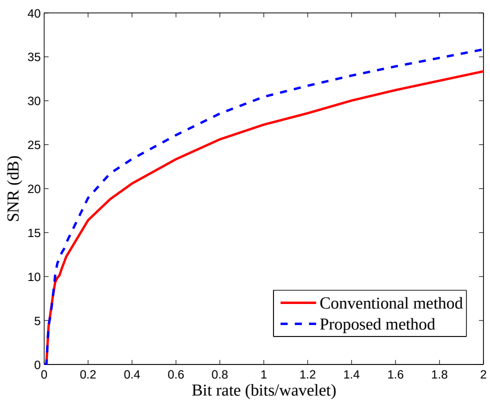
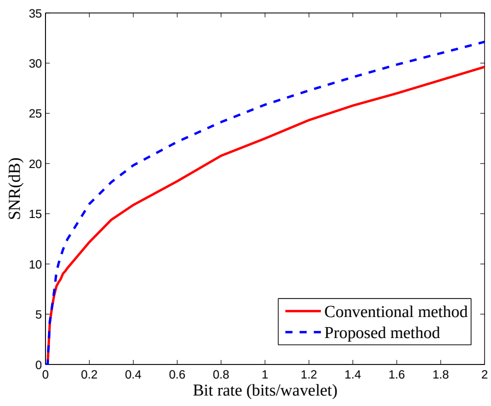


Figure 3.8: Distortion estimation for Venus.



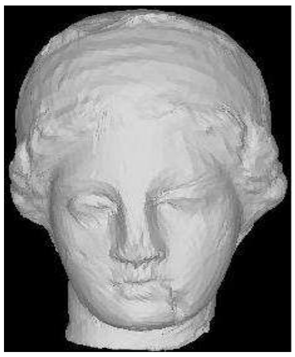
3.9.a: Venus



3.9.b: Armadillo

Figure 3.9: The comparison of the rate-SNR performance.

components. The rate-SNR performance is shown in Fig. 3.9 and the reconstructed models at some particular bit rates are shown in Fig. 3.10. From Fig. 3.9, we can see that our proposed coder significantly outperforms the state-of-the-art coder proposed in [SKKL05] with 2 $\sim$ 3 dB gain at most of the bit rates. The reconstructed models in Fig. 3.10 further demonstrate that our proposed coder achieves much better perceived visual quality, especially at low bit rates.


3.10.a: [SKKL05]

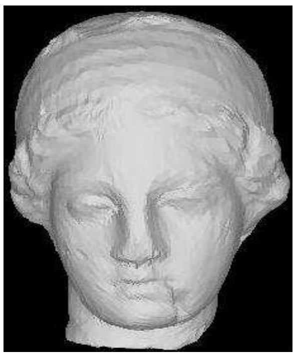

3.10.b: Proposed

Figure 3.10: The reconstructed models at 0.2 bpv.

3.5.2 Results of View and Illumination Dependent 3D Mesh Transmission

In this section, we test the performance of our proposed view and illumination dependent 3D transmission system. We compare our proposed distortion model with the conventional screen-space distortion model that does not take into account the illumination distortion.

Fig. 3.11 shows the comparison results. It can be seen that under the same number of vertices our proposed distortion model can achieve much better visual quality. This is because we give more refinement for the important parts while allocating less number of vertices for unimportant parts. On one hand, similar to other view-dependent methods, we do not refine the invisible parts. On the other hand, we also consider the lighting effects through giving higher priorities to the parts with high contrast, which leads to the better performance. In particular, as shown in Fig. 3.11, for a front view direction, the invisible back side is not refined at all. For the front side, the approach based on the screen-space distortion heavily refines the parts that contribute to the boundaries equally throughout the face. However, with shading enabled, some boundaries become visually more important than others. Our algorithm gives more refinement for these

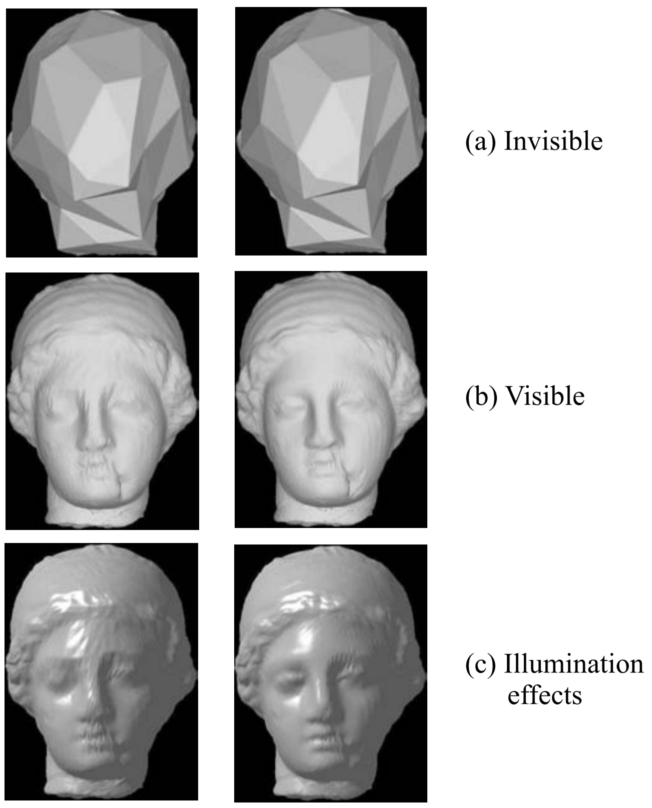


Figure 3.11: The performance comparison. Left: using space screen distortion metric. Right: using our proposed distortion model.

visually important boundary parts. The experimental results for other models are shown in Fig. 3.12.

Fig. 3.13 gives the progressive transmission performance with 2% $\sim$ 40% of the total number of vertices. It can be seen that the performance gain of our proposed system is more prominent at lower total available number of vertices.

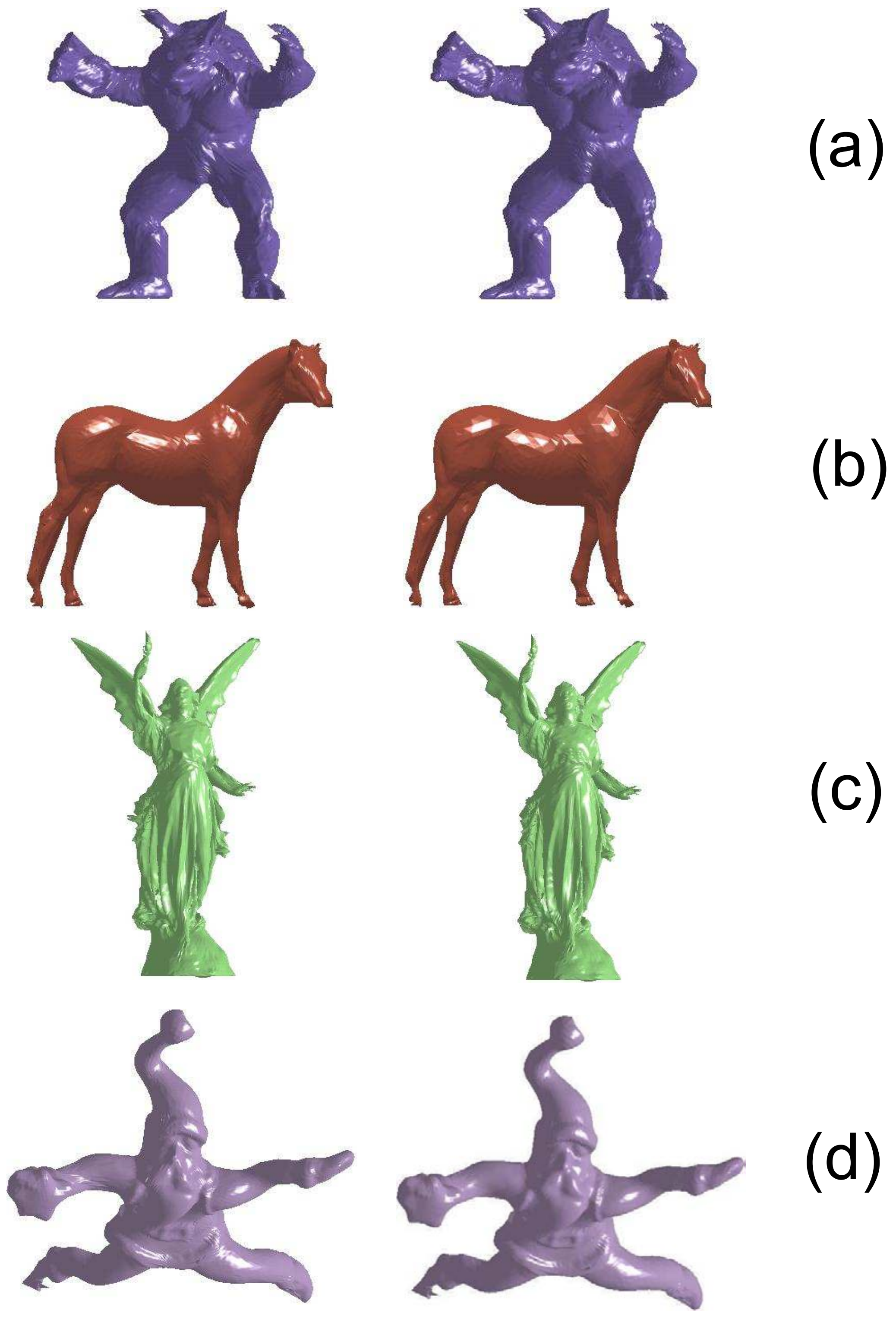


Figure 3.12: (a) Horse (b) Santa (c) Armadilo (d) Lucy. For each model, the left side is the result using space screen distortion metric while the right side is the result using our proposed distortion model. For all the models, we set $\lambda = 0.5$.

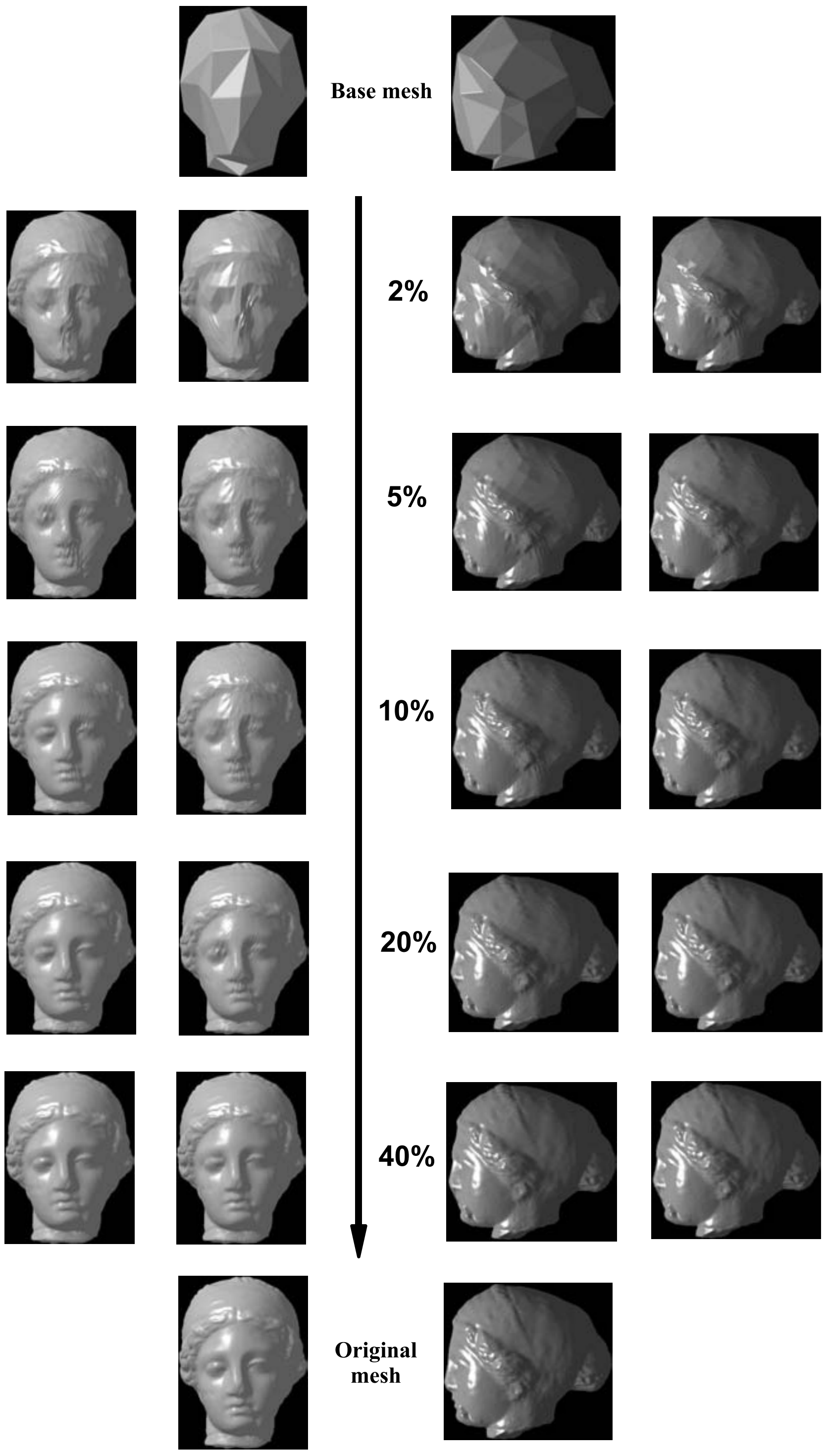


Figure 3.13: The progressive transmission performance with $I_a = I_d = I_s = 1.0, k_a = 0.2, k_d = 0.8, k_s = 1.0$ and $m = 10$. The four columns from left to right: using our proposed distortion model (front-view), using space screen distortion metric (front-view), using space screen distortion metric (side-view), and using our proposed distortion model (side-view).

Chapter 4

Conclusion and Future Work

4.1 Conclusion

3D mesh transmission in wireless environment is a challenge task due to the limited transmission bandwidth. In this thesis, we have proposed a view-dependent 3D mesh transmission scheme to save the transmission bandwidth. First, we introduce the view dependent property in the single rate transmission system. To enable view-dependent transmission, a mesh segmentation algorithm is developed, which divides the mesh based on the normal directions of the triangles and also takes coding redundancy and non-uniformity into consideration. Through mesh segmentation, it is possible to transmit the necessary segments instead of the mesh as a whole. In this way, the transmission size is largely reduced. The normalized effective surface area of each segment is used to determine the transmission priority. Moreover, rather than dividing the mesh into a fixed number of segments, we have developed analytical models to find the optimal number of segments. Through this optimization, we can further reduce the transmission size in single-rate transmission system.

Another contribution of this thesis is that we have proposed an improved wavelet-based progressive mesh coder. In particular, we proposed to scale up the vector wavelets generated in remeshing so that the inherent importance of wavelets can be truly utilized by the zerotree-like compression algorithms. As a byproduct, a distortion model is developed to accurately estimate the distortion of the reconstructed mesh without going through the actual reconstruction process. We have further applied the developed progressive mesh coder for progressive mesh transmission, where we consider not only view dependency but also illumination dependency. Specifically, the proposed transmission system partitions a semi-regular mesh into disjoint segments, and optimally allocates bits to each segment according to the proposed distortion model, which consists of both geometry distortion and illumination distortion. The experimental results demonstrated that by incorporating the illumination dependency significant quality improvement can be achieve, especially at low bit rates.

4.2 Future Work

Our current work can be further extended in several directions. First, it is interesting to study the occlusion issue when multiple mesh objects co-exist in one scene, where the occluded segments are not needed to be transmitted. In our framework, we do not consider the occlusion effects. This is due to the computational complexity of the existing occlusion detection algorithms and real-time requirements of our system. It is desired to develop fast occlusion detection algorithms with low complexity to further reduce the transmission size.

Second, we could consider the packet loss and error corruption problems in wire- less environment. In our current research, the transmission network is assumed to provide error-free delivery services. However, in hostile wireless environment, losses and corruptions are unlikely to avoid. It is meaningful to introduce robustness into our proposed framework. For example, we could introduce some overheads in each segment so that in the case of segment loss, we could well conceal the loss by utilizing neighboring segments to reconstruct the lost segment. In addition, error correction techniques can be combined with our prioritized mesh segments to ensure a robust transmission.

Thirdly, our current research only considers the geometry and topology information of 3D mesh models. Practical realistic 3D models are more complex due to additional attributes such as texture. When extending our framework to textured 3D mesh models, the task becomes even more challenging. The main problems are how to derive the texture distortion models and how to trade off between the geometry distortion and the texture distortion. Besides, a progressive representation of texture is needed to conduct the progressive transmission.

Last, our current distortion models are basically mean square error based, which does not match the perceptual quality. It would be interesting to apply perception distortion models into our framework. However, the existing perceptual models are too complex and computation-intensive, which is not suitable for real-time applications. Thus, the challenge is how to simplify perceptual distortion models while still keeping their perceptive characteristics. This could be one of the future directions.

References

[AKM$^+$06] M. Attene, S. Katz, M. Mortara, G. Patane, M. Spagnuolo, and A. Tal. Mesh segmentation: a comparative study. In *Proc. Shape Model. Int. (SMI'06)*, pages 14–25, 2006.

[ASF06] M. Attene, M. Spagnuolo, and B. Falcidieno. Hierarchical mesh segmentation based on fitting primitives. *the Visual Computer*, 22(3):181–193, 2006.

[CB98] G. Zhuang C. Bajaj, V. Pascucci. *Compression and coding of large CAD models*. Technical Report, University of Texax, 1998.

[CDST97] B. Chazelle, D. Dobkin, N. Shourhura, and A. Tal. Strategies for polyhedral surface decomposition: An experimental study. *Computational Geo.: Theory and Applications*, 7(4-5):327–342, 1997.

[CG02] L. Kobbelt C. Gotsman, S. Gumhold. Simplification and compression of 3D meshes. In *Tutorials on Multiresolution in Geometric Modelling*, 2002.

[CLB99] G. Zhuang C. L. Bajaj, V. Pascucci. Single resolution compression of arbitrary triangular meshes with properties. *Comput. Geom. Theory Appl.*, 14(1-3):167–186, Nov. 1999.

[CSAD04] D. Cohen-Steiner, P. Alliez, and M. Desbrun. Variational shape approximation. *ACM Trans. Graphic.*, 23(3):905–914, 2004.

[Dee95] M. Deering. Geometric compression. In *Computer Graphics Proc. - Annu. Conf. Series*, pages 13–20, Aug. 1995.

[EH72] D. J. Elzinga and D. W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6:379–394, 1972.

[GS98] S. Gumhold and W. Strasser. Real time compression of triangle mesh connectivity. In *Computer Graphics Proc. - Annu. Conf. Series*, pages 133–140, 1998.

[GWH01] M. Garland, A. Willmott, and P. Heckbert. Hierarchical face clustering on polygonal surfaces. In *Proc. ACM Symp. on Interact. 3D Graph.*, pages 49–58, 2001.

[HDD+98] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Mesh optimization. In *ACM SIGGRAPH*, pages 123–132, 1998.

[Hop96] H. Hoppe. Progressive meshes. In *ACM SIGGRAPH*, pages 99–108, 1996.

[Hop97] H. Hoppe. View-dependent refinement of progressive meshes. In *SIGGRAPH'97, Computer Graphics Annual Conferences Series*, pages 189–198, Aug. 1997.

[Hop98] H. Hoppe. Efficient implementation of progressive meshes. volume 22, pages 327–336, 1998.

[ISO01] ISO/IEC 14496-2. *Coding of Audio-Visual Objects: Visual*, Jul. 2001.

[KSS00] A. Khodakovsky, P. Schroder, and W. Sweldens. Progressive geometry compression. In *ACM SIGGRAPH*, pages 271–278, 2000.

[LE97] D. Luebke and C. Erikson. View-dependent simplification of arbitrary polygonal environments. In *Proc. of SIGGRAPH*, 1997.

[LK98] J. Li and C.-C.J. Kuo. Progressive coding of 3-d graphic models. *Proc. IEEE*, 86(6):1052–1063, Jun. 1998.

[LSS+98] A. W. F. Lee, W. Sweldens, P. Schroder, L. Cowsar, and D. Dobkin. MAPS: Multiresolution adaptive parameterization of surfaces. In *ACM SIGGRAPH*, pages 95–104, 1998.

[LZ04] R. Liu and H. Zhang. Segmentation of 3D meshes through spectral clustering. In *Pacific Conf. on Comp. Graph. and App.*, pages 298–305, 2004.

[MW99] A. Mangan and R. Whitaker. Partitioning 3D surface meshes using watershed segmentation. *IEEE Trans. On Visual. And Comp. Graph.*, 5(4):308–321, 1999.

[PH97] J. Popovic and H. Hoppe. Progressive simplicial complexes. In *ACM SIGGRAPH*, pages 217–224, 1997.

[PK00] J. Peng and C.-C.J. Kuo. Geometric compression for interactive transmission. *IEEE Visualization, Proceedings of the Conference on Visualization*, pages 319–326, 2000.

[PKK05] J. Peng, C.-S. Kim, and C.-C. Kuo. Technologies for 3D mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16:688?33, 2005.

[PR00] R. Pajarola and J. Rossignac. Compressed progressive meshes. *IEEE Trans. Vis. Comput. Graph.*, 6(1):79–93, 2000.

[Red97] M. Reddy. *Perceptually Modulated Level of Detail for Virtual Environments.* University of Edinburgh, 1997.

[Red01] M. Reddy. Perceptually optimized 3d graphics. *IEEE Computer Graphics and Applications*, 21(5):68–75, 2001.

[Sha93] J. M. Shapiro. Embedded image coding using zerotrees of wavelet coefficients. *IEEE Trans. Signal Processing*, 41(12):3445 – 3462, Dec. 1993.

[SKKL05] J.-Y. Sim, C.-S. Kim, C.-C. Kuo, and S.-U. Lee. Rate-distortion optimized compression and view-dependent transmission of 3D normal meshes. *IEEE Trans. Circuit and Syst. Video Technol*, 15(7):854–868, Jul. 2005.

[SP96] A. Said and W. A. Pearlman. A new fast and efficient image codec based on set partitioning in hierarchical trees. *IEEE Trans. Circuit and Syst. Video Technol*, 6(3):243–250, June 1996.

[SSGH01] P. Sander, J. Snyder, S. Gortler, and H. Hoppe. Texture mapping progressive meshes. In *ACM SIGGRAPH*, pages 409–416, 2001.

[STK02] S. Shlafman, A. Tal, and S. Katz. Metamorphosis of polyhedral surfaces using decomposition. In *EUROGRAPHICS*, pages 219–228, Sep. 2002.

[Tau99] G. Taubin. 3D geometry compression and progressive transmission. In *EUROGRAPHICS - State of the Art Report*, Sep. 1999.

[TG98] C. Touma and C. Gotsman. Triangle mesh compression. In *Proc. Graphics Interface*, pages 26–34, Jun. 1998.

[TGHL98] G. Taubin, A. Gueziec, W. Horn, and F. Lazarus. Progressive forest split compression. In *ACM SIGGRAPH*, pages 19–25, 1998.

[TR98] G. Taubin and J. Rossignac. Geometry compression through topological surgery. *ACM Trans. Graphics*, 17(2):84–115, Apr. 1998.

[TR99] G. Taubin and J. Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Trans. Visual. Comput. Graphics*, 5(1):47–61, Jan. 1999.

[YKK05a] Z. Yan, S. Kumar, and C.-C. Kuo. Mesh segmentation schemes for error resilient coding of 3D graphic models. *IEEE Trans. Circuit and Syst. Video Technol*, 15(1):138–144, Jan. 2005.

[YKK05b] S. Yang, C.-S. Kim, and C.-C. Kuo. A progressive view-dependent technique for interactive 3D mesh transmission. *IEEE Trans. Circuit and Syst. Video Technol*, 14(11):1249–1264, Nov. 2005.

[ZSGS04] K. Zhou, J. Synder, B. Guo, and H.-Y. Shum. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *EUROGRAPHICS/SIGGRAPH Symp. On Geo. Proc.*, pages 45–54, 2004.

Author's Publications

1. Juyong Zhang, Jianfei Cai, Wei Guan and Jianmin Zheng, Re-examination of applying wavelet based progressive image coder for 3D semi-regular mesh compression, *In proceeding of IEEE International Conference on Multimedia & Expo (ICME'08)*, 2008.

2. Wei Guan, Juyong Zhang, Jianfei Cai and Jianmin Zheng, Illumination and view dependent progressive transmission of semiregular meshes with subdivision connectivity, *In the proceeding of ACM Graphite*, 2007.

3. Wei Guan, Jianfei Cai, Jianmin Zheng and Chang Wen Chen, Segmentation based view-dependent 3D graphics model transmission, *IEEE Trans. on Multimedia, special issue on multimedia applications in mobile/wireless context, vol. 10, no. 5, pp. 724-734*, Aug 2008.

4. Wei Guan, Jianfei Cai and Jianmin Zheng, View-based 3D model transmission via mesh segmentation, *In proceeding of IEEE International Conference on Multimedia & Expo (ICME'07)*, 2007.

5. Wei Guan, Jianmin Zheng and Jianfei Cai, Including and optimizing shape parameters in Doo-Sabin subdivision surfaces for interpolation, *In proceeding of ACM Solid and Physical Modeling Symposium (SPM'07)*, 2007.