Ubiquitous Scalable Graphics: An End-to-End Framework using Wavelets

by

Fan Wu

A Dissertation
Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

In partial fulfillment of the Requirements for
the Degree of Doctor of Philosophy

in
Computer Science

by

November 19th, 2008

APPROVED:

Professor Emmanuel Agu Advisor	Professor Michael Gennert Committee Member
Professor Robert Lindeman Committee Member	Professor Holly Rushmeier External Committee Member Computer Science Department Yale University
Professor Michael Gennert Head of Department	

Abstract

Advances in ubiquitous displays and wireless communications have fueled the emergence of exciting mobile graphics applications including 3D virtual product catalogs, 3D maps, security monitoring systems and mobile games. Current trends that use cameras to capture geometry, material reflectance and other graphics elements means that very high resolution inputs is accessible to render extremely photorealistic scenes. However, captured graphics content can be many gigabytes in size, and must be simplified before they can be used on small mobile devices, which have limited resources, such as memory, screen size and battery energy. Scaling and converting graphics content to a suitable rendering format involves running several software tools, and selecting the best resolution for target mobile device is often done by trial and error, which all takes time. Wireless errors can also affect transmitted content and aggressive compression is needed for low-bandwidth wireless networks. Most rendering algorithms are currently optimized for visual realism and speed, but are not resource or energy efficient on mobile device. This dissertation focuses on the improvement of rendering performance by reducing the impacts of these problems with UbiWave, an end-to-end framework to enable real time mobile access to high resolution graphics using wavelets. The framework tackles the issues including simplification, transmission, and resource efficient rendering of graphics content on mobile device based on wavelets by utilizing 1) a *Perceptual Error Metric (PoI) for automatically computing the best resolution of graphics content for a given mobile display* to eliminate guesswork and save resources, 2) *Unequal Error Protection (UEP)* to improve the resilience to wireless errors, 3) an *Energy-efficient Adaptive Real-time Rendering (EARR) heuristic* to balance energy consumption, rendering speed and image quality and 4) an *Energy-efficient Streaming Technique*. The results facilitate a new

class of mobile graphics application which can gracefully adapt the lowest acceptable rendering resolution to the wireless network conditions and the availability of resources and battery energy on mobile device adaptively.

Acknowledgements

This dissertation and the growth in my knowledge over the last few years owe a great deal to many professors, colleagues, and friends. First among them is my advisor, Prof. Emmanuel O. Agu. He inspired my interest in mobile graphics research and gave me direction by suggesting interesting problems. It has been my luck to have him as my advisor. His technical and editorial advice was essential to the completion of this dissertation. I express my sincere thanks for his support, advice, patience, and encouragement throughout my graduate studies. His persistence in tackling problems, confidence, and great teaching will always be an inspiration.

My thank goes to the members of my Ph.D. committee, Prof. Michael Gennert, Prof. Robert Lindeman and Prof. Holly Rushmeier, who provided valuable feedback and suggestions to my dissertation proposal talk and dissertation drafts. All these helped to improve the presentation and content of this dissertation. I thank Prof. Matthew Ward for his time and efforts discussing with me in my research qualifying exam.

I would like to thank Clifford Lindsay for his valuable discussions on my research work. The friendship of all the other pervious and current ISRG members is much appreciated. They have contributed to many interesting and good-spirited discussions related to this research.

I thank the wonderful professors in the Computer Science department for both their serious lectures and casual chats. I thank the system support staff in our department for providing a well-maintained computing environment and utilities for my research needs. I am thankful for the financial supports I have received from my advisor and the department during my study and research in WPI.

Finally, I would like to thank my wife, Dr. Li Jiang for her understanding and love in my life. Her support and encouragement was in the end what made this dissertation possible.

My parents receive my deepest gratitude and love for their dedication and the many years of support during my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Recently, graphics on mobile devices is becoming popular because untethered computing is convenient and increases the productivity of workers. The following scenario demonstrates how mobile graphics applications can be used.

Motivating real estate mobile graphics use scenario: *Ann is an architect who works for Ulo corporation. Ulo corporation is a multi-national architectural film with clients and workers in 50 countries across the world. Ulo maintains a large database of high-resolution 3D architectural drawings of various types of buildings. In order to accommodate workers with PDAs, laptops and cell phones with graphics capability, different teams of architects work on different projects that are maintained in Ulo's database. Initially, an Ulo team visits a client and after preliminary discussions, retrieves possible design solutions and shows them to the client. These serve as starting points of the design process. After the client selects a viable option and requests modifications, the architects annotate the diagrams and return to Ulo's office to make necessary amendments. Periodically, the architects return to the client to show progress and seek more feedback, towards a mutually agreeable design. Some of Ulo's clients are not connected to the Internet. In such cases, Internet hotspots can serve as*

valuable affordable meeting locations.

In the scenario above, mobility in the home viewing software allowed teams to retrieve new architectural designs for clients on the spot after the client rejected the first one and it was convenient to avoid driving back to their office with clients. Although videos of the homes could have been used in this scenario, graphics allows teams to answer client what if question about snow. Clients could also interact with the homes and take a closer look at aspects that were important to clients. Indeed, mobile graphics is exploding and new applications are emerging. Computers can reduce border on long commutes by playing mobile versions of their favorite games during commutes. Other mobile graphics applications include telesurgery, security monitoring systems, 3D maps, and educational animations. Mobile graphics applications offer a new commercial opportunity especially considering that the total number of mobile devices sold annually far exceeds the number of personal computers sold. The mobile gaming industry already reports revenues in excess of $2.6 billion worldwide annually, and is expected to exceed $11 billion by the year 2010 [114].

Mobile graphics, which involves running networked computer graphics applications on mobile devices across wireless network, is a fast growing segment of the networks and graphics industries. The quest for visual realism in graphics is endless. A trend has emerged whereby real world scenes are now digitized to capture scene geometry, lighting, textures and material properties that can be used later to generate visually stunning graphics scenes. However, rendering 3D graphics on mobile devices still faces some fundamental problems including:

- **High-precision capture of graphics content is creating massive data:** The quest to make graphics scenes indistinguishable from the real word is endless. Today, movies and computer games have become extremely realistic because more precise geometry, materials and lighting are used. Classic techniques such as modeling object geometry using software packages and rendering (drawing) using Phong's shading equation [12] are now rarely used when extreme realism is desired. The current trend in graphics

is to place cameras around real objects and digitize scene attributes that can be used later for rendering. Today, almost every scene attribute can be captured from the real world including scene geometry (meshes) [42] [43] [44] [45], object reflectance properties [39], object texture [41] and scene lighting [40]. Several graphics research groups focus entirely on capturing elements of real graphics scenes. However, the increased precision of cameras today yields captured graphics content that is extremely large. For instance, in 1999, a team of 30 researchers from Stanford and the University of Washington spent a year in Italy, and digitized Michelangelo's statue to create a mesh representation. This geometric model can be obtained from their website and used to create highly realistic images. The largest models they captured had 2 billion faces and would require hundreds of gigabytes of memory to render. Even powerful desktop personal computers do not have enough memory to load a model of that size. Many such models are available on the Stanford group's website [45].

- **Different mathematical representations:** Each captured element, such as scene geometry(meshes), object material properties and scene illumination (lighting) is stored in a different mathematical representation. Each representation has many different file formats and each file format is only supported by certain graphics tools. This leads time-consuming conversion between different content's formats.

- **Manual LoD selection:** Since there is no metric for automatically determining the best resolution for each mobile device configuration, the scaling process is currently manual, involves trial-and-error to determine the best resolution for the specific mobile device. This manual approach is limiting with the hundreds of mobile device available in numerous configurations, such as the memory, battery energy and screen size. Essentially, there are no automatic sizing feature that makes it possible for two users to access the same graphics scene with a cell phone and headmounted displays respectively, and automatically download content at the best resolution for their devices.

- **Low wireless bandwidth and high error rate:** Wireless channels can have low bandwidth and high Bit Error Rate(BER). Users experience long transmission times on low bandwidth wireless network links and some latency due to retransmission of damaged packets. These sometimes affect the usability of real-time graphics applications such as internet multiplayer games.

- **Limited mobile resources:** Mobile devices tend to be limited in resources such as memory, CPU power, disk space, screen resolution and battery energy while graphics applications require large amount of these resources. Mobile devices also do not have adequate hardware support of graphics. These limitations make it difficult to process high resolution meshes and textures, or run sophisticated rendering algorithms that are necessary for visual photorealism. While the area of LoD management is rich, previous approaches focussed on controlling frame rates, but did not consider energy conservation on mobile devices.

In summary, the graphics content must be simplified before they can be used on small mobile devices. Scaling and converting graphics content to a suitable rendering format involves running several software tools, converting between mathematical representations and selecting the best resolution for a target mobile device is often done by trial and error, which all takes time. Wireless errors can also affect transmitted content and aggressive compression is needed for low-bandwidth wireless networks. At the mobile device, most rendering algorithms are currently optimized for visual realism and speed, but are not resource or energy efficient on mobile devices.

Therefore, This dissertation focuses on the improvement of rendering performance by reducing the impacts of these problems with UbiWave, an end-to-end scalable framework to enable real time mobile access to captured graphics using wavelets. The framework tackles the issues including simplification, transmission, and energy efficient rendering of graphics content on mobile device based on wavelets. The results facilitate a new class of mobile graphics application which can gracefully adapt the lowest acceptable rendering resolution

to the wireless network conditions and the availability of resources and battery energy on mobile device adaptively.

1.2 The Dissertation

The UbiWave uses wavelets to store and distribute all captured graphics content. Several groups have independently carried out research to capture graphics content. These include the capture of geometry [42, 43], texture [41], lighting [40] and material reflectance [39]. Many of these groups have made this content available. One of our goals is to make this content available to a wide spectrum of mobile devices at multiple levels of detail and save the energy. The content is encoded using wavelets, transmitted to the mobile device based on its configuration and rendered on the mobile device with less energy consumption adaptively. Wavelets can render graphics content at a continuum of levels of detail and thus directly address the heterogenity of mobile devices. Using wavelets as the common graphics standard to represent content is attractive because wavelets are already being applied to many graphics problems including simplifying geometry, material properties, textures, lighting and also for speeding up rendering. Using wavelets also offers aggressive compression rates and seamless integration with existing MPEG video and JPEG2000 image standards where wavelets are already in use. Our UbiWave framework tackles a gamut of issues including simplification, transmission, and resource efficient rendering of scanned content on mobile devices. To save scarce mobile resources, a perceptual metric for automatically computing the best resolution of graphics content for a given mobile display is presented. The perceptual error metric, Point of Imperceptibility (PoI) is evaluated by specially designed user studies. To improve the error resilience to wireless errors, a Forward Error Correction (FEC) scheme based on wavelets, Unequal Error Protection(UEP), is presented. To save energy consumption while maintaining acceptable image quality at real-time frame rates, an energy-efficient Adaptive Real-time Rendering (EARR) and an energy-efficient mesh streaming technique are presented. Us-

ing these four components, the UbiWave framework eliminates the time-consuming manual processes currently required to prepare captured content for a mobile device. This reduces the cost of creating most mobile graphics applications including mobile games and movies. The results also facilitate a new class of mobile graphics application which can adapt their rendering resolution to the availability of resources.

1.3 Contributions

The main contributions of this dissertation are the design, implementation and evaluation of UbiWave. The framework uses wavelets to store and render all scalable graphics contents on heterogeneous ubiquitous computing devices based the availability of their resources. The specific contributions of the dissertation will include:

1. **A complete end-to-end framework for scalable ubiquitous graphics using wavelets.** To render large 3D graphics content and uniform all different graphics mathematical representations, we construct a uniform system framework for scalable ubiquitous graphics using wavelets. If all graphics content is converted to wavelets, the wireless network always transmits base representations and trees of coefficients. Networking for graphics can be optimized around this structure. directly connect the mobile devices with repositories of captured content and vary the rendering quality of graphics on mobile devices *in real time* as resource availability changes. (Chapter 2)

2. **Pareto-Based Perceptual error metrics for automatically determining the lowest acceptable resolution of graphics content on a given mobile device's display.** To save scarce mobile resources, one of our themes is to render graphics contents at a level of realism that is just adequate for each type of mobile device. We develop a metric for automatically determining the levels-of-detail beyond which improvements are not perceivable on a given mobile display. A comprehensive user study is performed to validate PoI metric. (Chapters 4)

3. **Forward Error Correction scheme(UEP)to make wavelets-encoded graphics content more resilient to wireless errors.** Wireless channels have much higher error rates than wired networks. While many network protocols can retransmit packets that have errors, the additional roundtrip delay incurred is unacceptable for many real time applications such as real-time games. We propose a coding scheme that adds redundant bits to wavelets data prior to transmission for different parts of the transmitted wavelets content, depending on the amount of information it contains. (Chapter 5)

4. **An Energy-efficient Adaptive Real-time Rendering (EARR) heuristic.** As we know, battery energy of mobile device is the most limiting resource. To save battery energy while maintaining real-time rendering, the heuristic adaptively changes LoDs or CPU allocation to compensate for the changing demands of application elements in order to maintain a constant real time rendering frame rate. It reduces energy consumption by up to 60% while maintaining acceptable image quality at real-time frame rates. (Chapter 6)

5. **A workload predictor to adaptively predict frame rendering times.** Most of the resource management methods use trial and error. This will decrease the performance of the methods. To make our EARR heuristic predictable, we need a workload predictor to accurately predict its workload from frame to frame. The relative errors between predicted workload and real measured workload are bounded in 0.18 in our test scene. (Chapter 6)

6. **A dynamic CPU scheduler to save energy used by 3D applications.** To dynamically allocate the CPU resources, a dynamic CPU scheduler is needed in our EARR heuristic. Our CPU scheduler allocates the proper percentage of CPU resources to the 3D applications to save energy. (Chapter 6)

7. **An Energy-efficient 3D streaming technique.** Wireless network is low bandwidth comparable with wired network. To enable scalable transmission 3D graphics mod-

els in wireless network and avoid wasting transmitting energy in mobile device, an energy-efficient mesh streaming technique is presented in UbiWave. Most of previous approaches focused on the wireless network efficiency, but did not consider the energy consumption in mobile devices. (Chapter 7)

1.4 Roadmap

The remainder of this dissertation is organized as follows: Chapter 2 provides overview to our approach in this dissertation; Chapter 3 discusses related research in the areas of this dissertation; Chapter 4 describes our perceptual error metric(PoI); Chapter 5 describes our Forward Error Correction scheme(UEP); Chapter 6 describes our Energy-efficient Adaptive Real-time Rendering (EARR) heuristic; Chapter 7 describes wavelets-based multiresolution 3D streaming in UbiWave; Chapter 8 outlines possible future work; and finally Chapter 9 summarizes this dissertation and draws conclusions.

Chapter 2

Our Approach

This chapter presents the framework of our approach, UbiWave, a wavelets-based framework for scalable transmission of large meshes and graphics content.

2.1 Background on Wavelets

This section reviews basic concepts of wavelets and its current applications in computer graphics.

Wavelets, which originated from the work of Fourier, are a mathematical tool that can represent input functions at multiple resolution [16]. Figure 2.1 shows the process of wavelets transformation.

Wavelets can *decompose* input functions to yield a coarse (rough) base function, plus a tree of detail coefficients, as shown in Figure 2.1. *Reconstructing* the original function starts from the coarse base function. Its resolution is then successively improved by adding more levels of the detail coefficient tree. In UbiWave, our system for ubiquitous graphics all rendering inputs such as meshes [28], textures [17] and material reflectance properties [36] are converted and distributed as decomposed wavelets (base + coefficient tree) to facilitate scalable rendering on heterogeneous computing devices even when inputs are extremely large captured files.

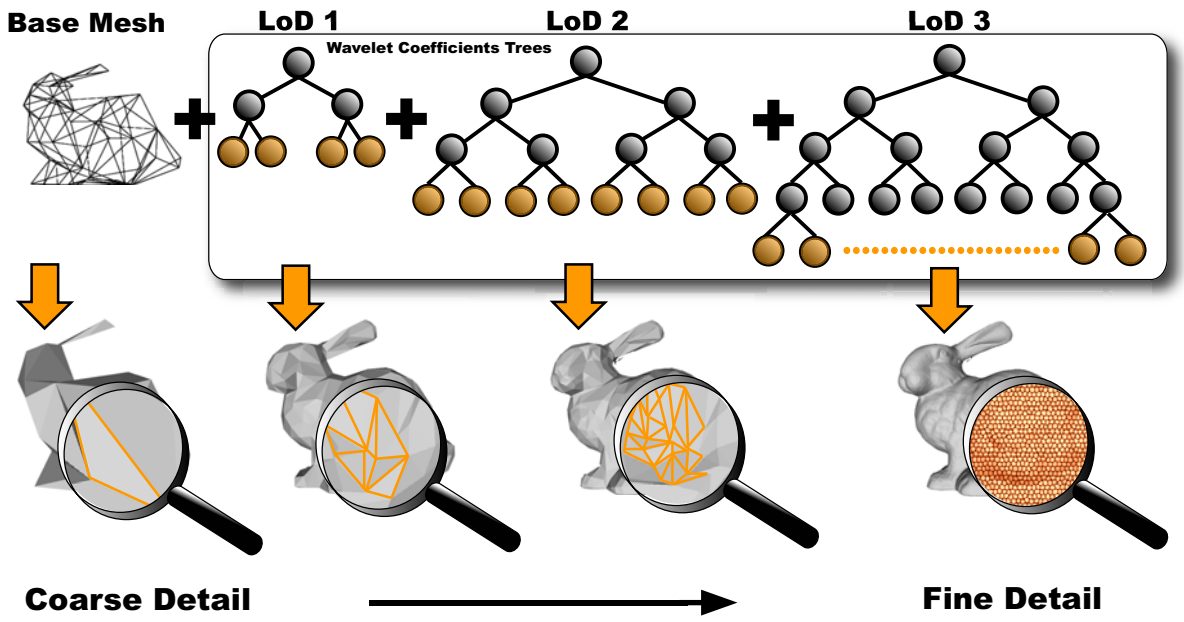


Figure 2.1: Wavelets-based Multiresolutions

While wavelets has been applied to many diverse fields, we limit our review here to research that uses wavelets in computer graphics. Wavelets have been used in a wide range of applications including graphics and image processing, ray tracing [38], information retrieval [30], FBI fingerprint storage [18], and geographic modeling. Today, published work has shown that almost all aspects of a graphics scene can be decomposed using wavelets including meshes, textures, material and reflectance properties. Schroeder [87] was one of the first to use wavelets in computer graphics and used wavelets to compress geometric and evaluate global illumination rendering equations.

- *Meshes:* Lounsbery proposed wavelets-based 3D compression [19, 28] by applying wavelets transforms to an arbitrary 3D mesh at several detail levels. During wavelets decomposition of meshes, a mesh is subdivided and deformed to make it fit the surface to be approximated. The original high resolution mesh is processed to generate a base connectivity file along with a sequence of smooth and detail cofficients that express the difference between successive levels of detail. Reconstruction starts with the base

mesh. As more wavelets coefficients are included, a higher resolution mesh will be rendered. These steps can be repeated at the required resolution levels. A hierarchy of meshes is obtained from the simplest one M^0, called base mesh, to the original mesh M^∞. The wavelets transform of meshes removes a large amount of correlation between neighboring vertices. This hierarchy of meshes at different resolutions is the basis of *multiresolution analysis* [28]. To make the mesh approximation M^{j-1} as close as possible to the original mesh M^j, the lifting scheme [31] is used. Valette's scheme [34] tries to convert the connectivity simplification to 1:4 subdivision as much as possible. If 4:1 simplification is not possible, other groups of three or two faces are chosen, or some faces are left unchanged. Several methods for performing wavelets transforms on meshes are based on interpolating subdivision schemes such as the Butterfly [21] that defines both interpolating and smoothing parts. The Loop [27] wavelets transform is an approximating scheme that has the advantage that the inverse transform uses Loop subdivision and produces the smoothest surfaces. After wavelets decomposition, adaptive arithmetic coding is often used to compress the size of the transmitted mesh and coefficients. In wavelets decomposition, a mesh is subdivided and deformed to make it fit the surface to be approximated. It consists of basic smooth coefficients and wavelets detailed coefficients. As more and more wavelets coefficients are included, a mesh of better resolution will be rendered. These steps can be repeated at the required resolution levels. We obtain a hierarchy of meshes from the simplest one M^0, called base mesh, to the original mesh M^j. Following [19], wavelets decomposition can be applied to the geometrical properties of the different meshes that are linked by the following matrix relations:

$$C^{j-1} = A^j C^j \tag{2.1}$$

$$D^{j-1} = B^j C^j \tag{2.2}$$

$$C^j = P^j C^{j-1} + Q^j D^{j-1} \tag{2.3}$$

where C^j is the $v^j \times 3$ matrix representing the coordinates of the vertices of M^j, v^j is the number of vertices for each mesh M^j. D^{j-1} is the $(v^j - v^{j-1}) \times 3$ matrix of the wavelets coefficients at level j. A^j and B^j are the analysis filters, P^j and Q^j are the synthesis filters. To ensure the exact reconstruction of M^j from M^{j-1} and D^{j-1}, the filter-bank must satisfy the following constraint:

$$[\frac{A^j}{B^j}] = [P^j|Q^j]^{-1} \tag{2.4}$$

To make the mesh approximation M^{j-1} as close as possible to the original mesh M^j, the lifting scheme [31] is used. Valette's scheme [34], [33] tries to convert the connectivity simplification to 1:4 subdivision as much as possible. If 4:1 simplification is not possible, other groups of three or two faces are chosen, or some faces are left unchanged.

- *Textures and images:* Techniques to compress images and textures using wavelets have also been proposed. A 2D wavelets transform that can be obtained by a separable decomposition in the horizontal and vertical directions [26]. Image compression based on the Discrete Wavelets Transform (DWT) is used in the JPEG2000 image standard [17]. Wavelets decomposition of textures and images is slightly different from that of meshes. In a preprocessing step, a nonstandard 2-D Haar wavelets decomposition of images is performed, which involves one step of horizontal pairwise averaging and differencing on the pixel values in each row of the image, followed by applying vertical pair-wise averaging and differencing to each column of the result.

- *Material reflectance and BRDFs:* Wavelets have been used to represent material reflectance or Bidirectional Reflectance Distribution Functions (BRDFs). In [36], reflections were encoded from one incident direction using a spherical wavelets representation, which can represent a slice of the BRDF with several hundreds of coefficients. [37] extended this work and represents 4D reflectance functions using 4D basis wavelets functions stored in a compact wavelets coefficient tree that keeps only the highest coefficients to reconstruct the BRDF and thresholding the rest to zero.

If captured content is available as decomposed wavelets, heterogeneous mobile devices can retrieve resolutions suitable for their use. Wavelets achieve aggressive compression, which is also useful for low cellular network bandwidths. Wavelets also support progressive refinement since users can view the increasingly improved intermediate results after receiving coefficients. Finally, using wavelets for graphics content facilitates integration of emerging mobile graphics standards with existing MPEG4 video and JPEG2000 image standards, where wavelets are already used.

2.2 UbiWave

As introduced in chapter 1, Current trends in using cameras to capture geometry, material reflectance and other graphics elements means that very high resolution inputs is accessible to render extremely photorealistic scenes. However, since mobile devices and wireless networks have limited resources, scaling of high-resolution content and conversion to a format that is suitable for rendering, is usually required. The dissertation presents our approach, UbiWave, an end-to-end framework that encompasses all stages including retrieving captured content, transmission and rendering on the mobile device. This wavelets-based framework ties in current trends in the capture of graphics content with our directions in mobile graphics. Figure 2.2 is an overview of our approach. Captured content is encoded using wavelets (on the left of figure). When retrieved, the content is tailored to the resources of a mobile device

and wireless network, transmitted wirelessly to the mobile device where it is rendered (right of the figure). The realism of rendering on the mobile device can be varied to accommodate mobile device constraints on the screen size and battery energy adaptively. Essentially, small devices such as a cell phone on a GPRS cellular data network or a laptop on a broadband WiMax network, can render the same scene, access the same rendering parameters from the same content databases, but automatically achieve the best resolutions for their configuration with less energy consumption. The network can be used in several ways. It can be programmed into a software download tool that downloaders of scanned content can use offline. In a more ambitious scenario, the quality of rendered images in mobile graphics applications would be varied dynamically based on available resources. For instance, the geometry of rendered objects and the quality of shading of a mobile flight simulator could be gracefully degraded as the devices's battery dies.

To achieve our end-to-end vision in UbiWave, we developed several novel algorithms. The shaded boxes in Figure 2.2 are novel algorithms and techniques in UbiWave. Our UbiWave has following benefits and solved the problems introduced in chapter 1.

1. **Uniform Representation and Increased Productivity.** Captured content will be more accessible to many heterogenous devices with minimal effort, speeding up prototyping of mobile graphics applications. Groups that spend months capturing content would just need a few extra hours to run software that converts captured content to a pre-agreed wavelets representation. Our envisioned framework takes the wavelets-encoded content as input and virtually eliminates manual processes currently required to scale and size graphics content for a target device.

2. **Pareto-Based Perceptual error metrics for different mobile device's display.** To save scarce mobile device, we proposed a perceptual error metrics for automatically rendering at the lowest level-of-detail that does not show visual artifacts, called the *Point of Imperceptibility (PoI)*.

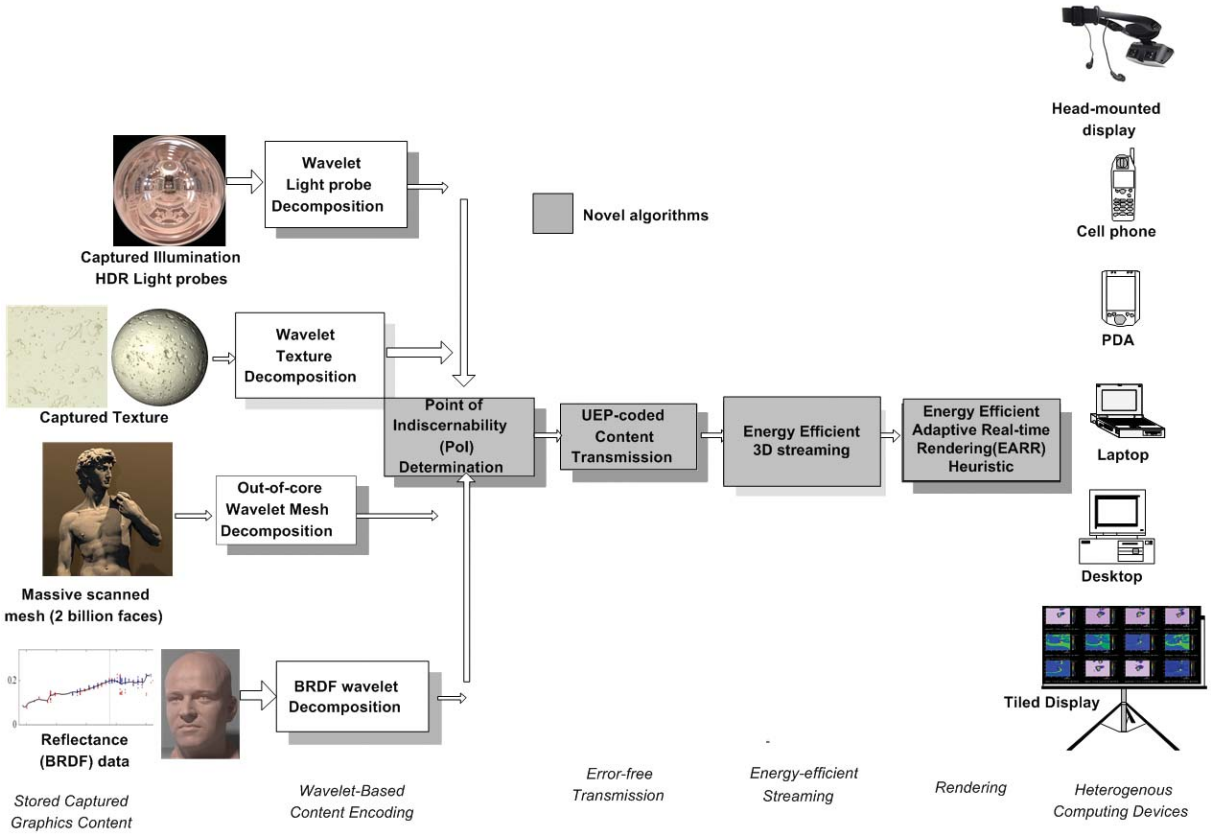


Figure 2.2: The overview of our mobile graphics approach

3. **Forward Error Correction scheme to make wavelets-encoded graphics content more resilient to wireless errors.** We propose a coding scheme that assigns redundant FEC bits to wavelets data prior to transmission for different parts of the transmitted wavelets content, depending on how important the content is.

4. **An Energy-efficient Adaptive Real-time Rendering (EARR) heuristic.** To balance energy consumption, rendering speed and image quality, we proposed the heuristic adaptively changes LoDs or CPU allocation to compensate for the changing demands of application elements in order to maintain a constant real time rendering frame rate.

5. **An Energy-efficient 3D Streaming.** We present an energy-efficient 3D streaming technique to enable scalable 3D content streaming in wireless network and avoid data

transmission which can not maintain real-time rendering speed in mobile device.

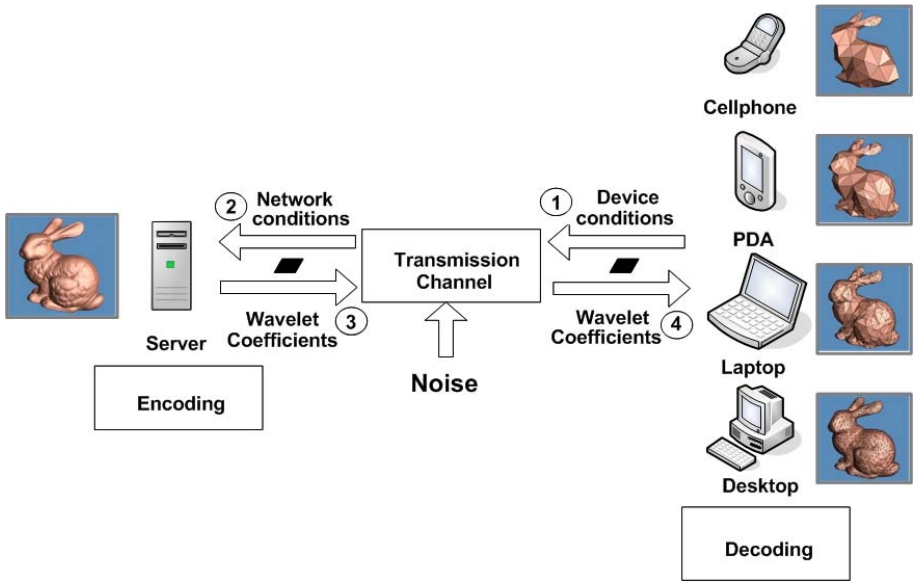


Figure 2.3: Proposed System Framework

Figure 2.3 shows our proposed system framework. The server only needs to send basic mesh connectivity information and corresponding wavelets coefficients to mobile devices, saving bandwidth and memory. The system works using the following three steps:

1. *Mesh preprocessing:* To speed up rendering, we perform wavelets decomposition as a pre-process at a server. In this pre-processing step, The server processes the original high resolution mesh to generate the base connectivity file and coefficient files for different levels of detail and calculates our perceptual metric for different screen sizes using different mesh and image LoDs. This computed data (or plot) is stored along with the corresponding meshes or images.

2. *Receiving mobile parameters:* At runtime, the mobile device transmits certain parameters to the server, so that the server can decide what LOD to transmit to a given mobile device. The transmitted mobile parameters include two parts: mobile device specification and wireless channel conditions. The mobile device specification includes its

screen size, CPU, memory and battery energy. The wireless channel parameters include measured bandwidth and error rate measured in the area around the mobile device.

3. *Server decision on what wavelets LOD to send:* After server receives the mobile parameters, it decides which level of wavelets coefficients will be sent to the mobile device using our perceptual error metrics for simplification to render the lowest level-of-detail that is just adequate for each type of mobile device. And unequal error protection coding scheme can protect the most important package in the high error rate wireless network.

4. *Client decision on what wavelets LOD to render:* After client, mobile device receives mesh data, it decides which level of wavelets coefficients will be rendered to the mobile device using energy-efficient adaptive real-time rendering heuristic. Typically, this decision is based on mobile device screen size, available CPU resources and user requirement. It can be expressed in the general form:

$$f(CPU, energy, screensize, bandwidth, error\ rate...) = level\ of\ coef. \tag{2.5}$$

2.3 Chapter Summary

This chapter presents a wavelets-based multiresolution framework for scalable graphics content transmission and rendering. The basic concepts of wavelets and its current applications in computer graphics. Then the UbiWave framework and how our approach benefits the mobile graphics is introduced. The related work of the dissertation is reviewed in the next chapter, followed by the discussion of the algorithms and techniques in each novel block.

Chapter 3

Related Work

This chapter reviews the research work related to the work in this dissertation. Five relevant research areas are covered including scalable graphics systems, perceptual error metrics for simplification, error protection coding schemes for wireless transmission of wavelets-encoded meshes, heuristic for energy-efficient rendering and 3D streaming technique.

3.1 Systems for Scalable Graphics

Previously proposed techniques to reduce the bandwidth and resource usage of graphics applications but do not use wavelets include image based simplification [75] [76] [77] [72], geometry compression [96, 97, 98, 99, 100, 101, 102, 103], and progressive transmission [89, 90, 91, 92, 93, 94, 95]. Alternate scalable graphics representations such as the use of points [127, 128, 129, 130, 131, 132, 133, 134] has also been proposed. Points supports scalable rendering and transmission but does not achieve aggressive compression rates. Spherical harmonics [137, 138, 139, 140, 141] can also be used to factorize low frequency lighting and speed up rendering, but not geometry or high frequency lighting. A few related graphics systems are also worth mentioning because they do try to adapt resource usage of graphics applications to the host machines. The ARTE system [11, 47] implements primarily vertex-based techniques such as polygon simplification and LoD techniques, but does not use wavelets

or consider error-resilience techniques against wireless channel errors. Repo3D [46] is a distributed graphics library that proposes an object-oriented framework for distributing input graphics models, but does not use wavelets or compress graphics content. The remote rendering pipeline [53] uses polygonal LoD techniques, progressive transmission and incremental encoding but not wavelets. [49, 50, 51, 52] have also proposed other graphics architectures for mobile devices, [48] combines multiple compression techniques to improve performance. We adopt wavelets-based multiresolution analysis for simplification because in addition to facilitating simplification, wavelets also achieve extremely aggressive compression ratios.

We present on a system solution for wavelets-based multiresolution. Our scheme only sends a base mesh and corresponding coefficients tree from the server side.

3.2 Perceptual Error Metric for Simplification

This section reviews the research work in error metric. The two most related bodies of work are surface-to-surface geometric simplification metrics and perceptual metrics.

3.2.1 Surface-to-Surface Geometric Simplification Metrics

Typically geometric metrics measure the deviation of the surface of a simplified version of a mesh from the original mesh. The Simplification envelopes algorithm [58] imposes a bound on the maximum geometric deviation between the original and simplified surface. Gueziecs approach to simplification [64, 65] uses a bounding volume approach to measure simplification error. Ronfard and Rossignac [66] measures for each potential edge collapse, the maximum distance between the simplified vertex and each of its supporting planes. Bajaj and Schikores plane mapping algorithm [67] uses a priority queue of vertex removal operations to simplify a mesh while measuring the maximum pointwise mapping distance at each step of the simplification. Garland and Heckbert [61] modify the error metric of Ronfard and Rossignac and propose a quadric error metric. Appearance-preserving simplification by

Cohen, Olano and Manocha [59], tries to bound the pixel-level shift of a particular point on the simplified objects surface.

In summary, previous mesh simplification error metrics [54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67] quantify how much a simplified mesh deviates from the original high-resolution mesh, but these metrics did not factor in the mobile screen dimensions. These simplification error metrics are insensitive to changes in screen size and using them unmodified would wrongly select the compute the same optimal mesh resolution for a tiny cell phone screen as it would for a larger laptop screen.

Tools such as METRO [14] and MESH [13] have been proposed to directly measure simplification errors, but do not factor in the device screen and behavior of human vision.

3.2.2 Perceptual Simplification Metrics

While surface-to-surface metrics focus mainly on the distortion of mesh geometry, visual effects such as lighting, shading and texturing also affect how perceivable simplification artifacts are. To account for these effects, elements of human vision have to be incorporated. A number of simplification metrics based on human perception have been developed. Rather than measure simplification errors in object space, perceptual metrics focus on how different mesh and image LoDs affect the contrast and frequency of pixel color changes. This theory is formalized as the CSF. Reddy [70] describes early work to guide LoD selection using a perceptual model. Reddy [70] analyzed the frequency content of objects and their LoDs in images rendered from multiple viewpoints. Reddy [71] presented a version of this approach for terrains. Lindstrom and Turk [72] describe an image-driven approach for guiding the simplification process itself. Luebke and Hallen [73] use the CSF to guide local view-dependent simplification based on the worst-case contrast and spatial frequency of features the simplification would induce in the rendered image. Williams et al [68] extends the work of Luebke and Hallen to 3D texture deviation.

In summary, The look of objects after rendering on a screen is considered by some pro-

posed perceptual metrics that model human vision [68, 23, 22, 29, 35, 70, 73], but also do not account for differences in mobile display sizes.

We focus on producing a closed form expression that can be computed easily, while accounting for geometric distortion, lighting effects and screen resolution.

3.3 Unequal Error Protection for Wavelets-encoded Meshes

Recent research efforts in the transmission of 3D objects over unreliable links have mostly focussed on still images and video sequences [82]. The compression and simplification of meshes is another active area of research. Very little research has attacked the issue of transmitting 3D graphics models over wireless networks. This is partly due to the fact that popular applications such as multiplayer games, which require this service have only recently emerged. Existing techniques for mitigating error while transmitting graphics models ranges from robust error coding to retransmission schemes for damaged network packets.

Two popular strategies for handling transmission errors are retransmission (Automatic Repeat-request,ARQ) and Forward Error Correction (FEC). ARQ schemes retransmission is used in many popular network protocols such as TCP/IP [112] and the IEEE 802.11 Wireless LAN standard [113]. However, when using ARQ techniques, a receiver has to wait one roundtrip delay every time a packet is retransmitted. In the worst case, the IEEE 802.11 standard will retransmit a packet up to 7 times. This retransmission delay is inappropriate for real time applications such as video streaming and mobile online games where latency can affect the user. For such real time applications or along satellite links where retransmission can take too long [107], FEC is preferred. FEC adds extra bits to transmitted data such that a receiver can detect and correct a small amount of bit errors. A retransmission-based error-resilient technique has been proposed by Bischoff and Kobbelt [83]. In their scheme, the base mesh is re-transmitted along with every Level-of-Detail (LoD) to guarantee that it is correctly received at the mobile client. However, the overhead of transmitting the base mesh can be

significant, making this scheme inefficient when packet loss rate is low.

The Hamming code [108] and Reed-Solomon [109] codes are two popular FEC schemes that perform well for most applications. However, wavelets-specified FEC schemes frequently outperform these codes for content that is encoded using wavelets. Wavelets-specific FEC techniques for images [110] and video transmission [111] have been proposed, but not for meshes. [106, 104, 105] previously applied Unequal Error Protection(UEP) to Compressed Progressive Meshes (CPM), but did not use wavelets encoding. We propose applying UEP for wireless transmission of wavelets-encoded meshes.

Bajaj *et al* [84] proposed several robust source coding methods for meshes. Even though this method adds a level of protection to the transmitted mesh, it does not adapt well to different ranges of channel packet loss rate. Yan *et al* [85] propose partitioning a 3D model into several segments that are then transmitted independently. However, they use experimental calibration to determine the number of error-protection bits assigned to different segments before transmission, which can be time-consuming. Our proposed technique applies an analytic distortion metric to determine the number of bits assigned per segment and does not require experimental calibration. MPEG-4 also uses error-resilient coding of 3D models that is similar to that proposed by Yan *et al* [85]. UEP is an error coding paradigm that assigns FEC bits based on the the amount of information a given segment contains. Al-Regib *et al* [106, 81] applies UEP to the CPM [86], a popular mesh representation in order to increase its resilience to transmission errors. As our main contribution, we apply UEP method to meshes that have been encoded using wavelets to make them more resilient to wireless errors. We note that UEP encoding of any content closely depends on a) the underlying structure of the content to be encoded and b) the ability to determine the relative importance of different parts of the mesh.

3.4 Heuristic for Energy-Efficient Rendering

The two bodies of work that are most related to our work are the areas of Level of Detail (LoD) management to achieve real-time rendering speeds, and energy management techniques for mobile devices.

3.4.1 LoD selection to achieve real-time frame rates

Funkhouser and Sequin [120], and Gobetti [122] both implement systems that bound rendering frame rates by selecting the appriopriate Level-of-detail (LoD). While Funkhouser and Sequin used discrete LoDs, Gobetti extended their work by using multiresolution representations of geometry. Wimmer and Wonka [123] investigated a number of algorithms for estimating an upper limit for rendering times on graphics hardware. The problem of maintaining a specified rendering speed has also been addressed in the Performer system [125], which reacts to changes in frame rate by switching LoDs. A model for predicting the time budget for rendering on mobile devices can be found in Tack et al [124].

3.4.2 Application-Directed Energy Management Techniques

Application-specific energy management schemes use either Dynamic Voltage and Frequency Scaling (DVFS) [115, 116, 119] or trade off the application's quality to increase energy efficiency [117, 118]. For instance, energy consumption can be reduced by intelligently reducing video quality [118] or document quality [117]. DVFS techniques save energy by dynamically reducing the processor's speed (or voltage) when possible and does not change the application's quality. GRACE-OS [116] proposes a DVFS framework for periodic multimedia applications. The GRACE-OS framework probabilistically predicts the CPU requirements of periodic multimedia applications in order to guide CPU speed settings. Chameleon [119] proposes CPU scheduling policies for a diverse applications including soft real-time (multimedia), interactive (word processor) and batch (compiler) applications. However, to the

best of our knowledge, dynamic CPU scheduling to conserve energy has not previously been applied to graphics applications. Moreover, our approach saves energy savings while maintaining acceptable frame rates and image quality.

3.5 3D Streaming

Streaming geometry involves piece-wise incremental transmission of mesh geometry from a server to a client. Streaming of multiresolution geometry is closely related with multiresolution representation and compression. Any type of multiresolution representation can be naturally extended to a view-independent geometry streaming framework. Moreover, using streaming, we can reduce the required network bandwidth between a server and a client with compressed multiresolution representation.

Progressive meshes was the first algorithm for progressive representation on meshes and was introduced by Hoppe [15]. This progressive representation is based on successive mesh simplification by edge contractions, which remove one vertex at a time. The inverse, that is, the reconstruction, is achieved by vertex splits. Khodakovsky et al. [25] presented a compression technique for semi-regular meshes. Valette and Prost [34] proposed a wavelets-based progressive compression scheme for irregular meshes.

Rusinkiewicz and Levoy proposed a new view-dependent streaming based on QSplat [134]. They provide a network based visualization for very dense polygon meshes but the splatting approach is not suitable when the client requires the full mesh connectivity. Therefore, a small number of errors during communication does not affect the global shape of the reconstructed mesh on the client side. However, loss of mesh connectivity can occur, since the technique ignores the original mesh connectivity.

Yang et al. [135] introduced a patch-based viewdependent streaming technique. They divide a mesh into several patches and compress each patch offline. In the streaming of a mesh, the entire connectivity information of the mesh is first transmitted to the client and

then the compressed patches are selected and streamed with respect to the client viewing information. With this approach, the resolution of the mesh cannot be changed smoothly on the client side.

Kim et al. [136] introduce a framework for view-dependent streaming of multiresolution meshes. The transmission order of the detail data can be adjusted dynamically according to the visual importance. This approach has to send the operation packets, which increases the network overhead. So it is not suitable for wireless network with low bandwidth.

3.6 Chapter Summary

This chapter presents the related work to each parts of our framework including scalable graphics systems, perceptual error metrics for simplification, error protection coding schemes for wireless transmission of wavelets-encoded meshes, heuristic for energy-efficient rendering and 3D streaming technique.

Chapter 4

Pareto-Based Perceptual Error Metric

This chapter presents the research work for Pareto-Based Perceptual Metric (PoI) for Simplification on Mobile Displays. This work has been published in [5, 6, 7, 8].

4.1 Overview

Our work focusses on a typical mobile graphics scenario shown in figure 4.1. Very high resolution graphics meshes and textures are stored on a server, and then simplified when requested by a mobile client. Meshes and textures are simplified on mobile devices for several reasons. First, mobile devices have limited battery energy, memory and disk space and lower resolution meshes and textures consume less of these scarce system resources. Secondly, increasing mesh and texture resolutions generally increases visual realism. However, above a certain Level-of-Detail (LoD), users cannot perceive these increases in mesh and texture resolution. We call this LoD the *Point of Imperceptibility (PoI)*. Essentially, increasing LoD above the PoI wastes mobile resources since users cannot perceive improvements in image quality.

In order to minimize wasting mobile resources, a mobile client should render meshes and textures that are as close as possible to its PoI. Our preliminary experiments [5, 6] showed that the level of detail users can perceive depends on the screen size: smaller screens show

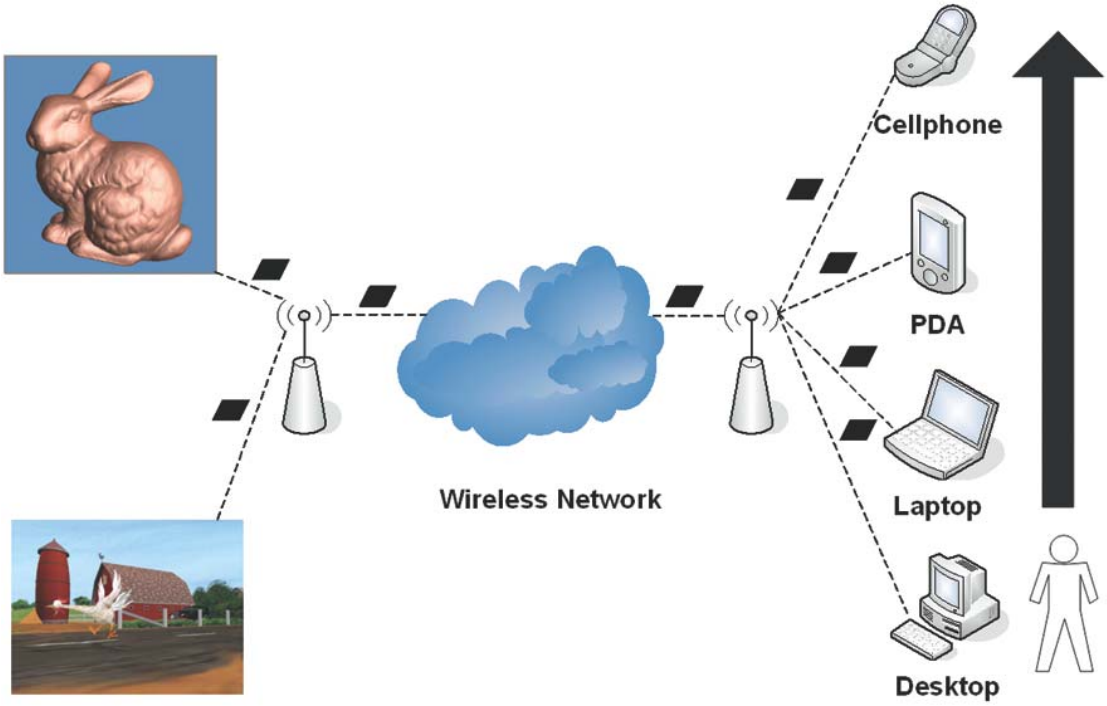


Figure 4.1: Mobile graphics scenario

less detail and hence have a lower PoI. For instance, we found that for a given mesh, a laptop's display had a PoI of 20K faces, while a cell phone's PoI was 5K faces for the same mesh. This represents a 4x change in the acceptable LoD level based on screen size. Previous work has neglected to directly relate selected LoD levels with target screen size. Other factors such as distance of the rendered object from the screen, object details and whether the user zooms in all affect the perceptibility of simplification artifacts. However, we focus primarily on how PoI changes with screen resolution.

In the scenario in figure 4.1, we need metrics that enable the server to compute the PoI that corresponds to a mobile device's screen size. Since so many different mobile display resolutions exist, experimentally determining PoI for each mobile display resolution would be impractical. Thus, we would prefer a closed form expression that can be easily computed on-the-fly to determine PoI. Walkthrough applications that are dynamically scaled down for mobile clients would benefit from a PoI metric. Follow-me applications graphics applica-

tions [69] in mobile environments are another class of applications that emphasize the need for a PoI that can be quickly computed based on screen resolution. In follow-me mobile applications, mobile users physically move between mobile devices but can access the same applications using these devices from different locations. The server would need to easily compute the PoI of the user's current device and then transmit graphics files that correspond to the PoI of that mobile display's resolution. We adopt wavelets-based multiresolution analysis for simplification because in addition to facilitating simplification, wavelets also achieve extremely aggressive compression ratios that are suitable for ultra-low bandwidth wireless links such as wide-area cellular phone networks.

Metrics for LoD selection while accounting for different target screen resolutions is a general problem that is addressed by this paper. We develop a metric that can be used to find the PoI of both meshes and textures (images). Our metric is developed in two distinct stages. First, we consider the geometry of test meshes without considering the effects of lighting. In addition to the influence of screen size, visual effects such as lighting and antialiasing make simplification artifacts less perceiveable and hence further reduce the PoI. Luebke and Hallen [73] showed that mesh lighting can reduce the perceptibility of simplification errors by a factor of 2-3. To account for the effects of lighting, we then extend our geometry-only metric using results from work on perceptual simplification metrics. In summary, our metric determines the mesh (and texture) LoD that corresponds to the PoI and takes as input 1) the original mesh (or texture) LoD 2) mobile device screen size and 3) lighting that will be applied to the mesh. We validate our proposed metric through extensive user studies.

Our metric generates a pareto distribution that corresponds to meshes or images at various LoDs. We use this pareto shape to determine thresholds on the perceptibility of mesh distortion on various mobile screen sizes. Since our metric explicitly factors in screen size, a family of slightly shifted pareto plots are generated for mobile displays at different resolutions. To account for reductions in error perception when meshes are lit and shaded, we use Contrast Sensitivity Function (CSF) curves that have become the basis for many perceptual

metrics in graphics. As a contribution, we are able to easily determine mesh undulation frequencies during our wavelets decomposition of meshes and use these frequencies as inputs to the CSF curve. The rest of the chapter is organized as following: Section 4.2 reviews the fundamental knowledge; Section 4.3 gives an overview of our approach; Section 4.4 presents the derivation of our metric; Section 4.5 presents user studies to validate our metric; Finally, Sections 4.6 summarizes the chapter. The results studied in this chapter are used in our Energy-efficient Adaptive Real-time Rendering Heuristic in chapter 6 and 3D streaming technique in chapter 7.

4.2 Background

4.2.1 Wavelets for Mesh Simplification

Our mobile graphics framework uses wavelets for representing meshes and textures at various Levels of detail *(called multiresolution analysis)*. While wavelets have been applied in many diverse fields, we now provide some background focussing mainly on research that has used wavelets for the multiresolution analysis of mesh geometry and textures. Schroeder [87] was one of the first to use wavelets in computer graphics, using wavelets to compress geometry and evaluate global illumination rendering equations. Multiresolution analysis using wavelets is described in [28], [24]. In wavelets decomposition, a mesh is subdivided and deformed to make it fit the surface to be approximated. This decomposition generates a base mesh, along with smoothness and detail wavelets coefficients. During reconstruction, starting with the base mesh, as more wavelets coefficients are included, a higher resolution mesh is generated. These reconstruction steps can be repeated until the desired mesh resolution (LoD) is achieved. Thus, considering all intermediate mesh LoDs generated during reconstruction, a hierarchy of meshes is obtained starting from the simplest one M^0, called base mesh,to the original mesh M^j.

In our mobile framework, our goal is to select the mesh (or texture) LoD that is suitable for

each mobile display's resolution. We note that many alternative vertex-based simplification algorithms have been proposed including vertex decimation [87], edge contraction [15], and valence-based simplification [20]. We simplify meshes and images using wavelets for two main reasons. First, wavelets decomposition achieves very aggressive compression ratios that significantly speed up mesh transmission over ultra-low bandwidth links such as cellular phone networks. After wavelets decomposition of a mesh, the total size of the base mesh plus wavelets coefficients is much smaller (over 100x smaller in our experiments) than the size of the original mesh. Secondly, since we can easily determine the frequencies of wavelets decomposition filters, we can use these filter frequencies to parameterize CSF (explained below) used in the perceptual aspects of our PoI metric that account for mesh lighting. In fact, using the wavelets filter frequency to directly parameterize the perceptual CSF curves is one of our contributions. Our mesh simplification implementation is based on the Loop wavelets transform. After wavelets decomposition, adaptive arithmetic coding is often used to compress the size of the transmitted mesh and coefficients.

4.2.2 L_p Norm distortion metric

The L_p norm is a popular metric to measure the level of surface-to-surface distortion introduced by wavelets-based simplification. Both the Hausdorff distance and the general mapping distance compute the final distance between two surfaces as the maximum of all the pointwise distances(known as the L_∞ norm). In some cases, we could take the average, the root mean square(the L_2 norm or Euclidean norm), or some other combination. Tack *et al* [32] derived a form of the L_p norm metric that we find useful for our work. Their expression is:

$$\begin{aligned} L_p(S_1,S_2) &= max(l^p(S_1,S_2), l^p(S_2,S_1)) \\ L_p(S_1,S_2) &= ((\frac{1}{A(S_1)}) \int_{a \in S_1} d(a,S_2)^p da)^{\frac{1}{p}} \end{aligned} \tag{4.1}$$

S_1 and S_2 in equation 4.1 are the surfaces to compare, $d(a,B)$ is the Euclidean distance between point a of surface S_1 and the closest point q on surface S_2. The $A(x)$ term is a function returning the area of the surface x in object coordinates. The MESH [13] software tool implements these error metrics by calculating the error per triangle and then averages the results for all triangles weighted by their respective areas.

$$L_p(S_1,S_2) = \frac{\sum_{i=0}^{F} A(T_i) l^p(T_i,S_2)}{\sum_{i=0}^{F} A(T_i)} \tag{4.2}$$

In Equation 4.2, F is the number of triangles in the surface and T_i is triangle i of the surface. Tack's Lp norm metric satisfies the conditions that: $L_p(A,B) = 0$ if A and B are of identical features, $L_p(A,B)$ is small if A and B have small dissimilar features and $L_p(A,B)$ is large if A and B have very dissimilar features.

4.2.3 Perceptual Simplification Metrics

Most research to incorporate elements of human vision into simplification metrics is based on the *Contrast Sensitivity Function (CSF)* [70, 73]. The CSF plots contrast sensitivity against spatial frequency of sinusoidal color changes, as shown in figure 4.2. In contrast grating tests, users are shown these sinusoidally changing images with varying degrees of contrast sensitivity and spatial change to determine the limits of human vision. The CSF essentially graphs the results of these contrast grating tests and determines the threshold of human vision for a single or average observer. The highest contrast and lowest spatial frequency exhibited by a complex rendered image at the pixel level essentially determines what point on the CSF curve that image lies. The CSF takes into account many of the non-linearities inherent in the human visual system.

Determining the highest contrast and lowest frequency exhibited by a given image is usually the most challenging aspect of using CSF curves to develop perceptual metrics. Extensive user studies using images rendered using different LoDs (calibration) can determine user

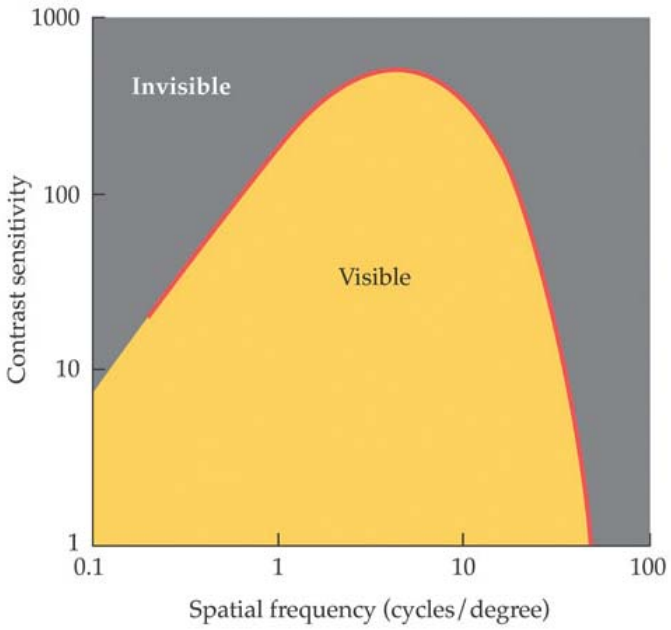


Figure 4.2: contrast sensitivity function

sensitivity thresholds. In extreme cases, an inverse Fourier transform of the final rendered image [63] was used to determine the frequency components of a rendered image. As one major contribution of our work, we use the frequency of the wavelets decomposition filters directly as the frequency of the CSF curves.

4.3 Our Approach for Perceptual Simplification

In this section, we give an overview of our approach with an emphasis on building intuition and presenting our hypotheses. Our proposed metric for imperciptible simplification extends the work of Tack *et al* [32]. Tack *et al* expressed the surface-to-surface L_p norm error due to mesh simplification but did not explicitly address how perceptible these errors were on different screen resolutions, or consider the effects of lighting on the final rendered mesh. We integrate the original mesh LoD, the target display size, and the effects of scene lighting on error perceptibility into a single expression that can easily be computed. We develop our PoI in two distinct phases. First, in section 4.4.1 only distortion in mesh geometry is

considered without considering the effects of lighting. Next, in section 4.4.2, we extend our PoI metric by integrating perceptual elements (using the CSF) to account for scene lighting.

At this point, we preview some of our final results and give a qualitative description of our general direction. Figure 4.3 shows sample pareto plots generated by the version of our metric that considers distortions in mesh geometry. Three plots are shown corresponding to three different target screen resolutions (laptop:640x720, PDA:240x320, cellphone:120x160). Starting with an original high-resolution mesh, we generate fourteen levels of detail. We then use our PoI metric to compute the root mean square error generated by an LoD on each of our three target screens and plot them. Essentially, our metric produces a family of plots, one for each target screen resolution. Based on figure 4.3, we hypothesize that:

- *Hypothesis 1:* Each of the curves in figure 4.3 follows a pareto distribution. Starting with the original mesh on the left of the plots, relatively low errors are generated as LoD is reduced up until a *knee point*. Beyond the knee point, reducing LoD levels result in sharp increases in error. We conjecture that a) users will be unable to perceive simplification errors to the left of the knee point b) the knee point corresponds to the Point of Imperceptibility (PoI); and c) To the right of the PoI (knee point), errors rise quickly and users easily perceive simplification errors.

- *Hypothesis 2:* Based on the results of Luebke and Hallan, we conjecture that lighting will further reduce the perceptibility of errors, essentially lowering PoI. Referring to figure figure 4.3, lighting will essentially shift our pareto plots to the right (knee point occurs at higher LoDs).

The original metric proposed by Tack *et al* and other previously mentioned surface-to-surface metrics are oblivious of the perceptibility of simplification errors when rendered on various screen sizes. As a further note, Tack's original expression would generate the same pareto plot (and not a family of plots) for all three target screen resolutions. Essentially, our goal is extend Tack's expression to account for changes in the pareto distribution plots to

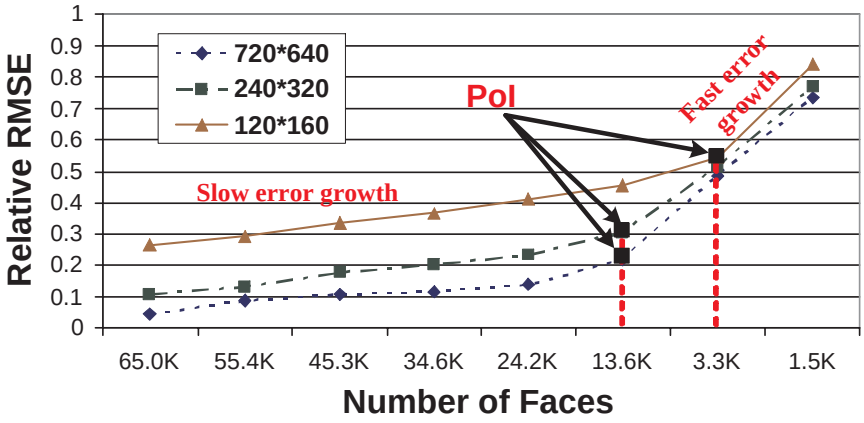


Figure 4.3: Sample pareto plots of our final PoI metric

account for different mobile screen sizes and then factor in the effects of lighting on error perception.

4.4 Point of Imperceptibility Error Metrics

4.4.1 Geometry-only PoI Metric

This section derives the first part of our metric that considers only the distortion of mesh geometry without factoring in the effects of lighting. Our derivation has three steps: 1) Calculate mesh distortion due to simplification; 2) Render the simplified mesh to a large virtual screen M_1; 3) Minify blocks of pixels of M_1 to a pixel of the mobile device's screen M_2. We can magnify if $M_2 > M_1$ as in a large tiled display. For screen-aligned images, only step 3 is performed; Figure 4.4 summarizes the steps to derive our metric. Equation 4.3 is our PoI metric for geometry only.

$$l^p(S_1,S_2) = \underbrace{(1-\frac{F_2}{F_1})\frac{\sum_{i=0}^{F} A(T_i) l^p(T_i,S_2)}{\sum_{i=0}^{F} A(T_i)}}_{Object-space} + \underbrace{E_p}_{Screen-space} \tag{4.3}$$

where F_1 is the number of triangles in the surface S_1, F_2 is the number of triangles in the

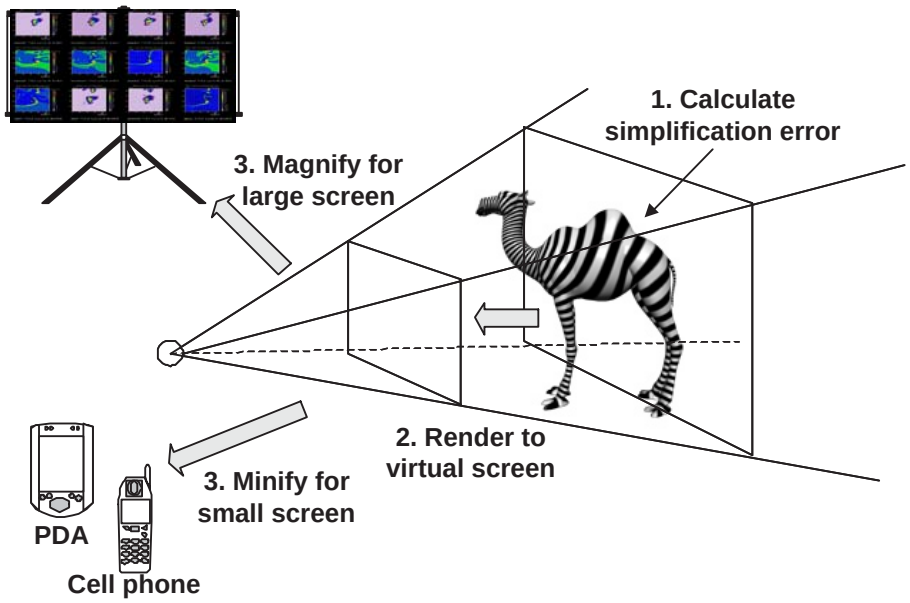


Figure 4.4: Steps in deriving our PoI metric

surface S_2. If $F_1 < F_2$, we can rewrite the factor $1 - \frac{F_2}{F_1}$ as $1 - \frac{F_1}{F_2}$.

The first part of Equation 4.3 deals with surface-to-surface LoD simplification errors in object space and the second term (E_p) deals with pixel-level minification errors as a result of rendering to different screen resolutions (see figure 4.5). A high-resolution mesh that is rendered to a small screen potentially incurs errors in both terms. In section 4.4.3, we apply our metric to texture simplification. A screen-aligned texture incurs errors only due to the second (E_p) term. Likewise, if the same mesh LoD (no surface simplification) is rendered to two different screen sizes, the error due to the first term is zero and the error due to the second term is calculated. For a target mobile display width, W (in pixels) and height H (in

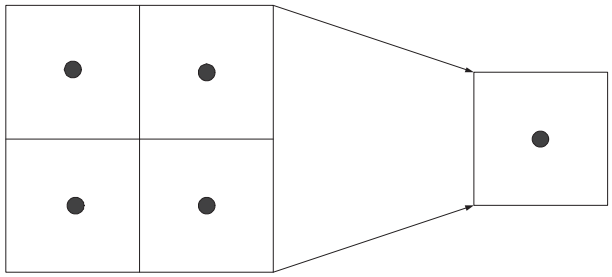


Figure 4.5: Minifying virtual screen pixels onto mobile screen

pixels), the term E_p is defined as:

$$\begin{aligned} E_p &= \sqrt[p]{\frac{1}{W_2 \times H_2} \sum_{i=1}^{W_2 \times H_2} (\frac{W_2 \times H_2}{W_1 \times H_1} \sum_{j=1}^{\frac{W_1 \times H_1}{W_2 \times H_2}} \sqrt[p]{S_p})^p} \\ \text{where } S_p &= \frac{1}{3}[(\frac{R_{i2} - R_{j1}}{256})^p + (\frac{G_{i2} - G_{j1}}{256})^p + (\frac{B_{i2} - B_{j1}}{256})^p] \end{aligned} \tag{4.4}$$

where W_1 and H_1 are the width and height of screen M_1 and W_2 and H_2 are the width and height of screen M_2. We assume that $W_1 > W_2$. Otherwise, W_1 and W_2 should be interchanged. In our system, we use relative Root Mean Square Error (RMSE) $(p = 2)$. In Equation 4.4, we calculate the screen space *RGB* error pixel by pixel and normalize it. As shown in figure 4.5, S_p calculates the average relative mean square error of *RGB* values between one pixel on the smaller screen and the corresponding group of pixels on the larger screen. If the screen sizes are not same, we compare the $\frac{W_1 \times H_1}{W_2 \times H_2}$ pixels on the large screen with one corresponding pixel on small screen and calculate the relative root mean square error between them.

We calculated and averaged our final error metric in equation 4.5 for all pixels on a target screen while considering four different meshes. Three different screen sizes were considered: 640x720 pixels for laptop, 240x320 pixels for PDA and 120x160 pixels for the cellphone. Figure 4.6 shows the computed errors which when plotted resemble a pareto distribution with a knee point. One way to calculate the knee point of the pareto plots, the slope of segments of the could be calculated. The point between two consecutive segments with the highest change in slope is the knee point (PoI).

4.4.2 Perceptural Metric

In this section we extend our PoI metric to account for lit meshes using the the CSF described in section 4.2.3. First, we note that effects such as lighting and shading can reduce the perceptibility (sharpness) of mesh edges, hiding differences in detail between LoDs. Essentially,

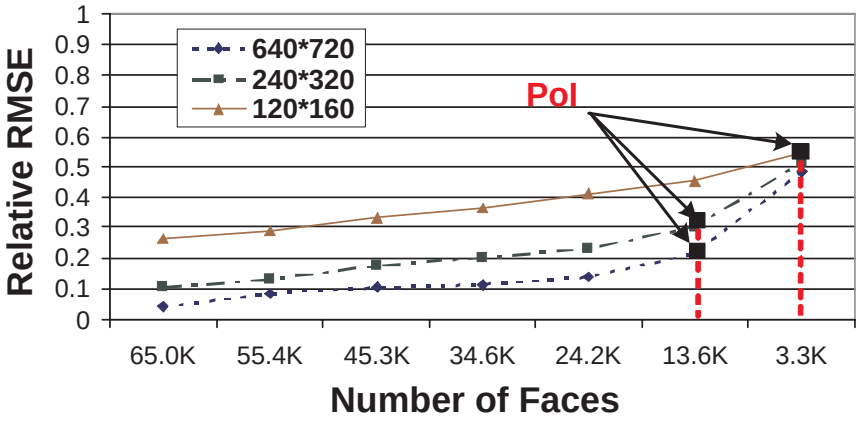


Figure 4.6: Our metric plotted for meshes at different LoDs using the final PoI metric

lighting and shading makes geometric distortion less visible. We can model this reduction in the perceptibility of errors as passing the rendered mesh image (sharp) through a filter that removes some of the distortion. To account for the error masking caused by lighting, we multiply our geometry-only expression (equation 4.3) by a factor $M_p(S_1,S_2)$. As before, this $M_p(S_1,S_2)$ factor considers the perceptibility of errors when rendering our lit mesh onto a large virtual screen of size S_1 and minifying the image onto a target mobile display of size S_2. Thus our new PoI expression takes the form:

$$l^p(S_1,S_2) = [(1-\frac{F_2}{F_1})\frac{\sum_{i=0}^{F} A(T_i) l^p(T_i,S_2)}{\sum_{i=0}^{F} A(T_i)} + E_p] \times M_p(S_1,S_2) \tag{4.5}$$

Next we derive an expression for $M_p(S_1,S_2)$. The human visual system is often modeled as a linear system and its response to visual excitation is expressed as a convolution of the input stimulus with the visual cortex's impulse response. Equivalently, to determine the perceptibility of a lit mesh, we can determine the eye's visual response by multiplying the wavelets transform of the mesh by the CSF. The CSF measures the response of human vision at different spatial frequencies. Mannos and Sakrison [74], after conducting a series of psychophysical experiments on human subjects, found that the CSF can be modeled by the function in the equation 4.6. Here f_s is spatial frequency in cycles per degree.

$$C_s(f_s) = [0.0499 + 0.2964 f_s] \times exp[-(0.114 f_s)^{1.1}] \tag{4.6}$$

where f_s is spatial frequency in cycles per degree. To integrate the CSF into our metric, during wavelets decomposition we determine the frequency ranges corresponding with each LoD. We then multiply each mesh frequency range with the CSF's response curve in that range. Figure 4.7 shows the CSF function mapped to frequency ranges obtained during wavelets decomposition of a mesh. This curve essentially defines how sensitive the human eye is to frequency ranges generated during wavelets transformation of the original mesh. Thus, for each frequency band a *sensitivity weight*, C_m can be computed by integrating the CSF curve in figure 4.7 over that frequency band. The weight measures the *average contrast sensitivity value* of the CSF curve in each band. We then multiply the wavelets coefficients at each LoD (frequency level) by the CSF sensitivity weights C_m corresponding to that frequency range. Wavelets transformation involves the iterative application of two mirror filters, L, a low-pass filter and H, a high-pass filter. Thus, by applying H to a discrete input with bandwidth $(0, \pi)$, a level of coefficients with bandwidth $(\pi/2, \pi)$ is acquired. Thus, after m iterations, the weight for level m is:

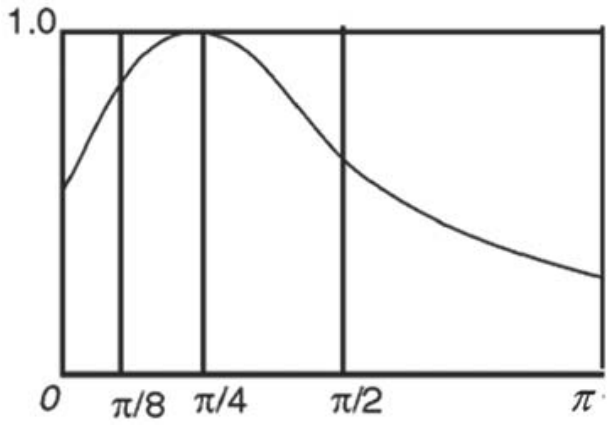


Figure 4.7: Contrast Sensitivity Function Curve

$$C_m = \frac{\int_{F_m} CSF(\omega) d\omega}{A(F_m)} \tag{4.7}$$

where F_m is the frequency subband $\left(\frac{\pi}{2^m}, \frac{\pi}{2^{m-1}}\right)$ and $A(F_m)$ is the width of the band.

We now describe how the *sensitivity weight,* C_m can be incorporated during wavelets transformation of meshes. Wavelets decomposition of a mesh yields a coarse mesh and a tree of wavelets coefficients as described in section 4.2. To determine the perceptibility of a mesh LoD, all wavelets coefficients at that tree level are multiplied by the the CSF sensitivity weight corresponding to that level. When a given mesh LoD is rendered to a screen, each wavelets coefficient *i* in that level of the wavelets tree refines (modifies) a mesh face at that LoD which in turn maps to a block of pixels when rendered. For each mesh LoD, we need to track which group of pixels are modified by each wavelets coefficients at that level. A brute force approach would be to render all LoD levels to a screen and determine what pixels each face maps to. However, the following method to track this relationship requires only one rendering of the original mesh. At the lowest level of the wavelets tree (finest LoD), each wavelets coefficient maps to a triangle in the original mesh which in turn maps to a group of pixels after rendering. By rendering the original mesh, we can track what group of pixels each triangle (wavelets leaf node) maps to. At any higher (coarser) level in the wavelets coefficient tree we can calculate what screen pixels each coefficient in that level maps to as the union of all pixels corresponding to all leaf nodes that are its children in the tree. Thus, for each pixel (i,j) on the target mobile device, we can multiply the wavelets coefficients in a given frequency band with the contrast sensitivity weight corresponding to that frequency band giving:

$$D_1(m,i,j) = C_m W(m,i,j) \tag{4.8}$$

Here C_m is the contrast sensitivity weight and $W(m,i,j)$ is the wavelets coefficient at level m and pixel location (i,j). Essentially, we quantify the perceptibility of error to the frequency

input at pixel (i, j) in the m^{th} sub-band frequency. Our perceptual comparison metric is then computed as:

$$M_p(S_1, S_2) = \frac{\sum_{m,i,j} |D_1(m,i,j) - D_2(m,i,j)|^2}{N_h \times N_v} \tag{4.9}$$

where D_1 and D_2 are error values of pixel i, j, when considering level m of the wavelets coefficients. N_h and N_v are the number of pixels in horizontal and vertical directions on the small screen. If the screen size are not the same, we calculate the screen error between one pixel on the smaller screen and the corresponding group of pixels on the larger screen (minification) as shown in figure 4.5. Figure 4.8 shows our final results using equation 4.5. The errors with lighting and shading are clearly smaller than the errors without lighting and shading.

Figure 4.9 shows meshes of the different LoD levels of the model. This demonstrates the visual depiction of the results of using our perceptual error metric.

4.4.3 Applying our PoI metric to Image Simplification

Our proposed metric can also be used to compute the PoI of a high resolution image (texture). To compute image PoI, we note that since there is no surface-to-surface error for a 2-D image,

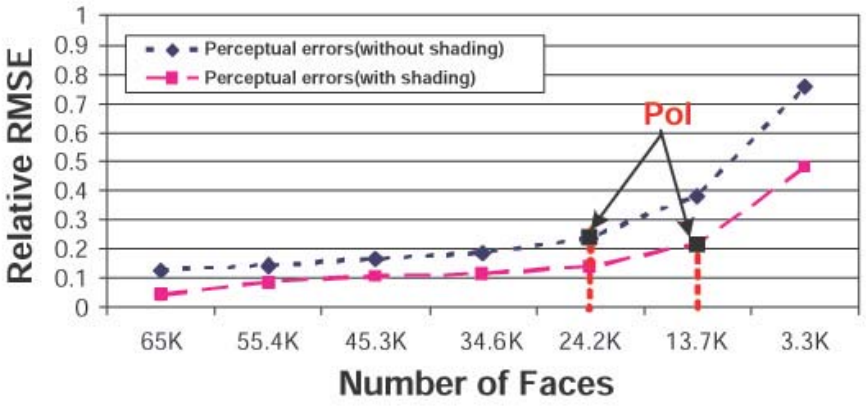


Figure 4.8: Curves with shading and without shading

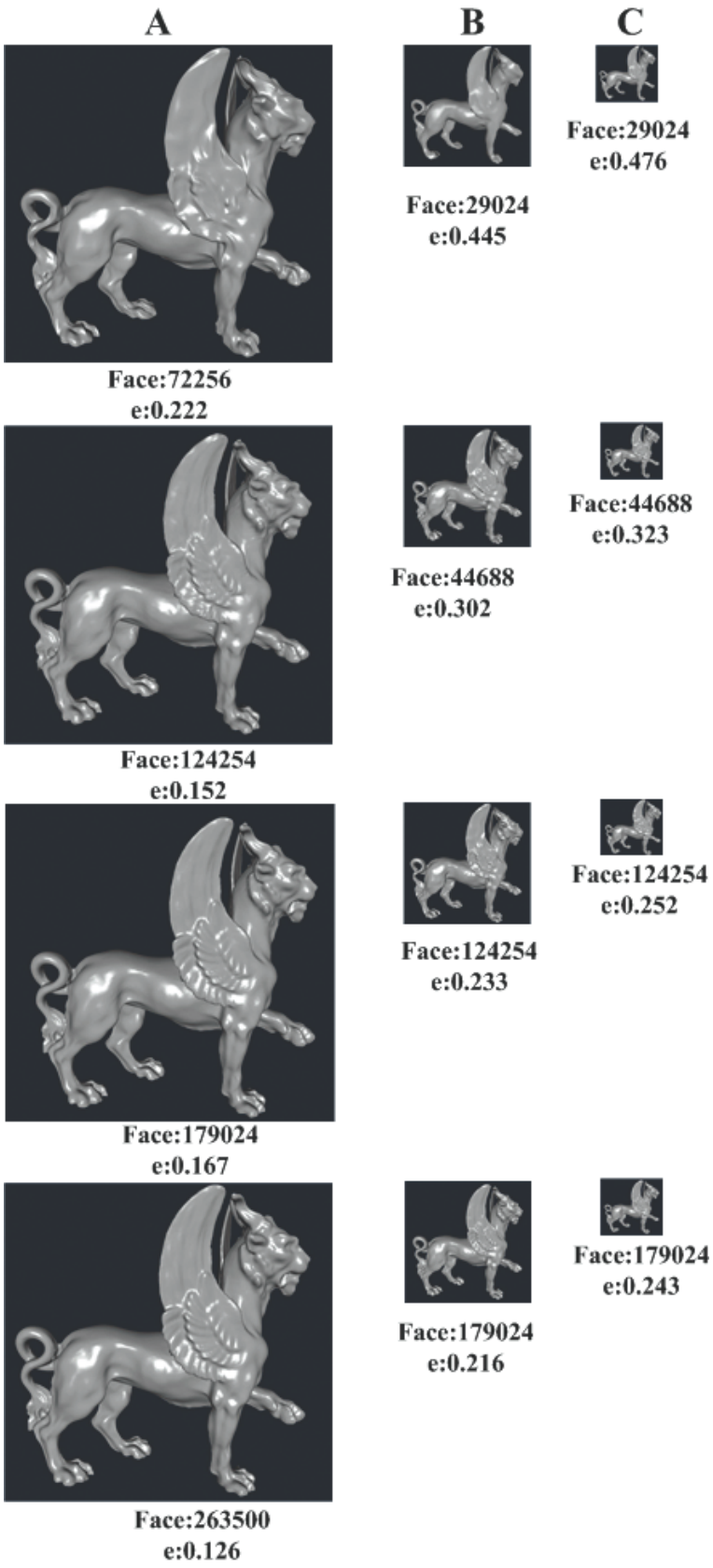


Figure 4.9: File size and Relative RMSE A: 640*720, B:240*320, C: 120*160

we only calculate the screen-space term in equation 4.5, and the E_p and $M_p(S_1, S_2)$ terms. Figure 4.10 shows the image error on different screen sizes.

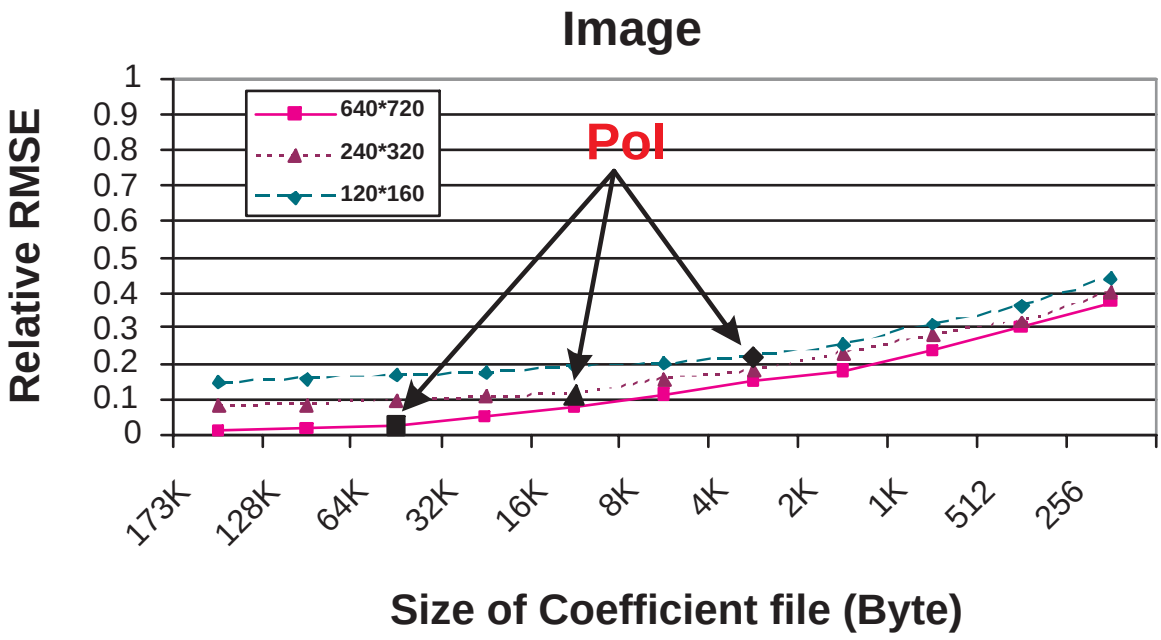


Figure 4.10: Perceptual Error for image on different screen sizes

In Figure 4.10, the black dot indicates the resolution where PoI lies. From Figure 4.10, we see that when the screen size is 640x720, the PoI occurs at a coefficient file of size 64KB, which means that a user perceives very little improvements by using a coefficient file greater than 64KB. Thus, even if a mobile device has adequate resources to receive and render the image at full resolution, we can save valuable resources by sending the 64KB coefficient file. Figure 4.11 shows images generated using different sizes of coefficients files.

4.5 Metric Validation and Analysis

4.5.1 User Studies

Having derived a metric that can be computed to automatically determine the PoI of a given mesh or image, we needed to validate that it works for real users. Specifically, we needed to

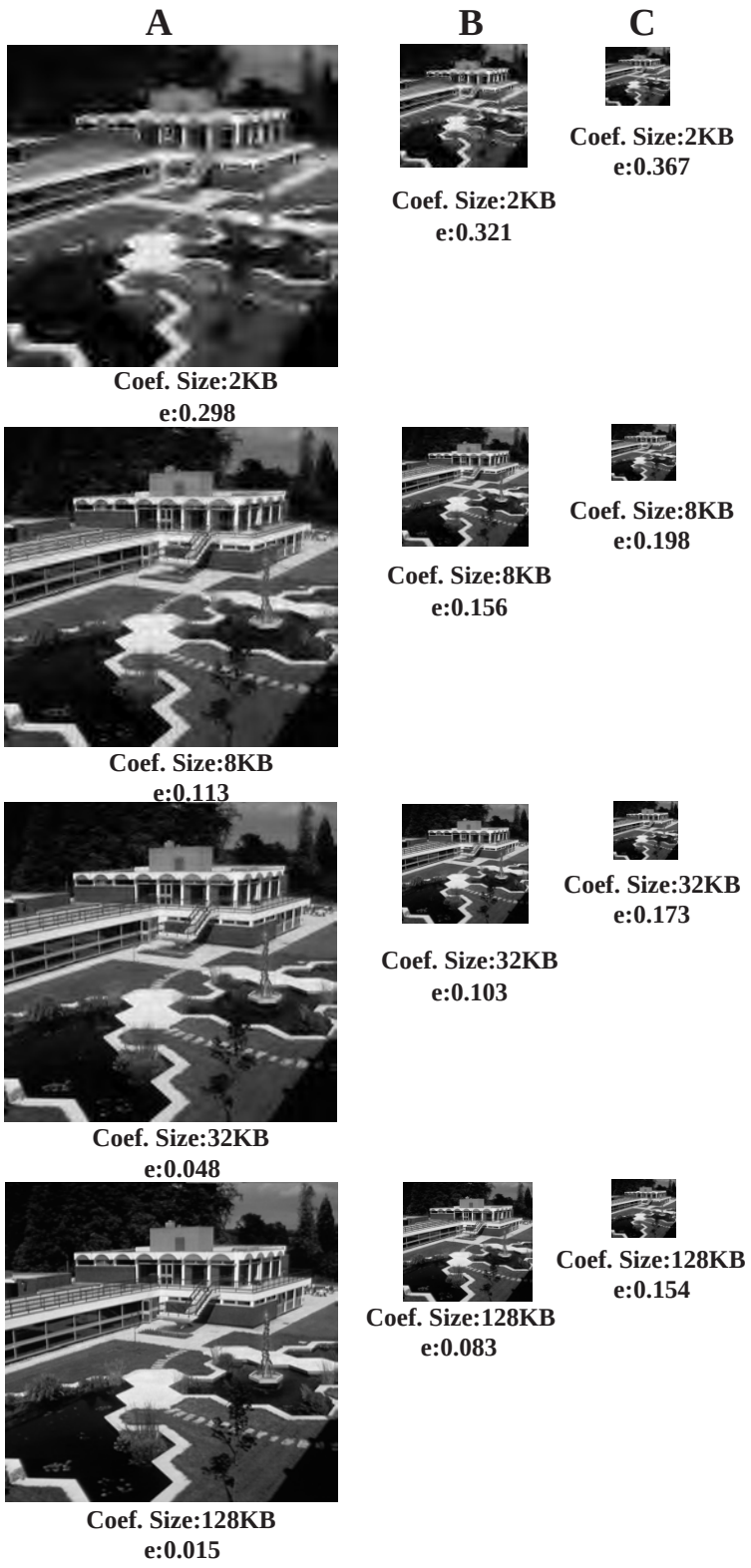


Figure 4.11: Coefficients file size and Relative RMSE A: 640*720, B:240*320, C: 120*160

ascertain that our metric accurately selects the LoD at which users stop perceiving increases in mesh or image resolution. Our approach was to generate a series of mesh and image LoDs and use our metric to determine the PoI LoD. We then asked real users to visually inspect the actual rendered meshes and images that correspond to those LoDs. Our metric worked correctly if it correctly determined the same PoI chosen by real users. Our user studies involved 84 participants.

In our study, several LoDs of a bunny model were rendered at three different screen sizes (laptop:640x720, PDA:240x320 and cellphone:120x160). Figure 4.12 shows one set of bunny images for screen size 240x320 pixels, ordered from highest(left) to lowest (right) resolution. Each LoD level is placed beside the original and shown to the user in pairs. For instance, images 1 and 2, 1 and 3, 1 and 4, and 1 and 5 in figure 4.12 are presented to the user in pairs. For each pair of images, users are required to respond in one of three ways: a) A is more detailed than B; b) A and B are approximately same; c) B is more detailed than A. The permutations of the two mesh models and three different screen sizes generate eighteen different image pairs that we randomly show the user as questions 1-18. For example, in figure 4.14, Q05 presents two images to the user with the option of responding with ”A is more detailed than B”, ”A and B are approximately same” or ”B is more detailed than A” as described above.

For each screen size, as we proceed from high resolution pairs to low resolutions pairs, as long as the user is able to correctly distinguish between pairs of images, the PoI has not been reached. Once the number of incorrect answers (or user answers ’approximately same’) becomes significantly less than the number of correct answers, the lower resolution of the

1 **2** **3** **4** **5** **6** **7**

Figure 4.12: An example of rendered meshes of seven different LoDs in user study

pair is regarded as the PoI that we are looking for. We then compare this experimentally determined PoI with PoI calculated by our metric.

The relative positions of each image pair are also randomized so that the user does not use the image position as a cue to guess which one is more detailed. For instance, if we always placed the high resolution image to the right (image B) and the user happened to notice this, she may always guess that B is more detailed even if she visually cannot see this. To minimize the effect of ambiguities in our phrasing of our questions or problems due to language barriers (English was not the first language of some participants), the users are first shown sample images along with the correct answers.

Figure 4.13 is the screen shot of the survey pages.

Figure 4.14 shows sample results of user study. Each question corresponds to a pair of images at a particular screen size. For instance Q05 refers to images 1 and 6 rendered to a PDA screen size. In Figure 4.6, the resolutions employed in our user studies are shown as black squares. The red line shows where the PoI computed by our metric lies. Comparing the result in figure 4.14 and figure 4.6, we observe that users indeed begin to wrongly distinguish the models or answer incorrectly (The answers of ”almost same” are scored as incorrect) at the PoI. Our calculated error metric is shown with each image to assist the reader in visually mapping calculated error values to an actual image quality.

4.5.2 Error Distribution

Thus far, our analysis has focussed entirely on the relative Root Mean Square Error (RMSE) resulting from the simplification process. The relative RMSE value however does not show what parts of the simplified model or image exhibits the most errors. It would be instructive to investigate the error distribution of the models of a given resolution on different screen sizes. To do this, we perform error calculations per pixel and map them to colors that give a sense of the degree of errors occurring at that pixel. For each pixel in the smaller screen, we calculate the relative RMSE. Then we normalize the relative RMSE so that all errors occur

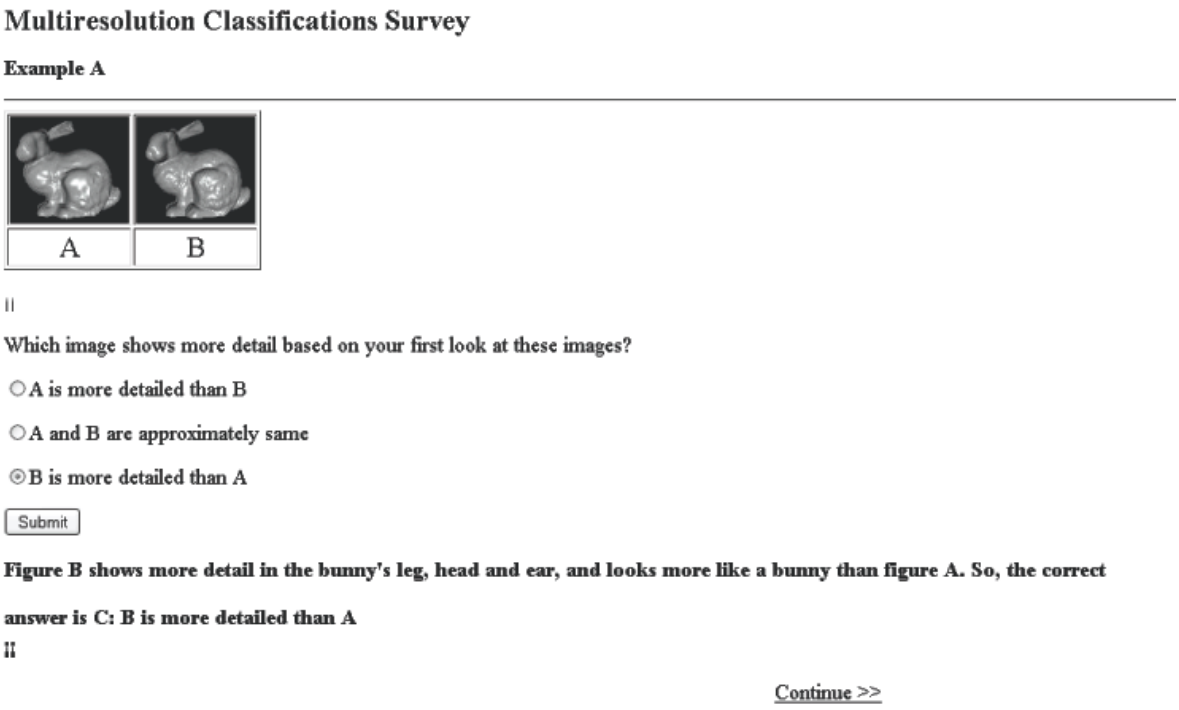


Figure 4.13: Screen shot of survey pages

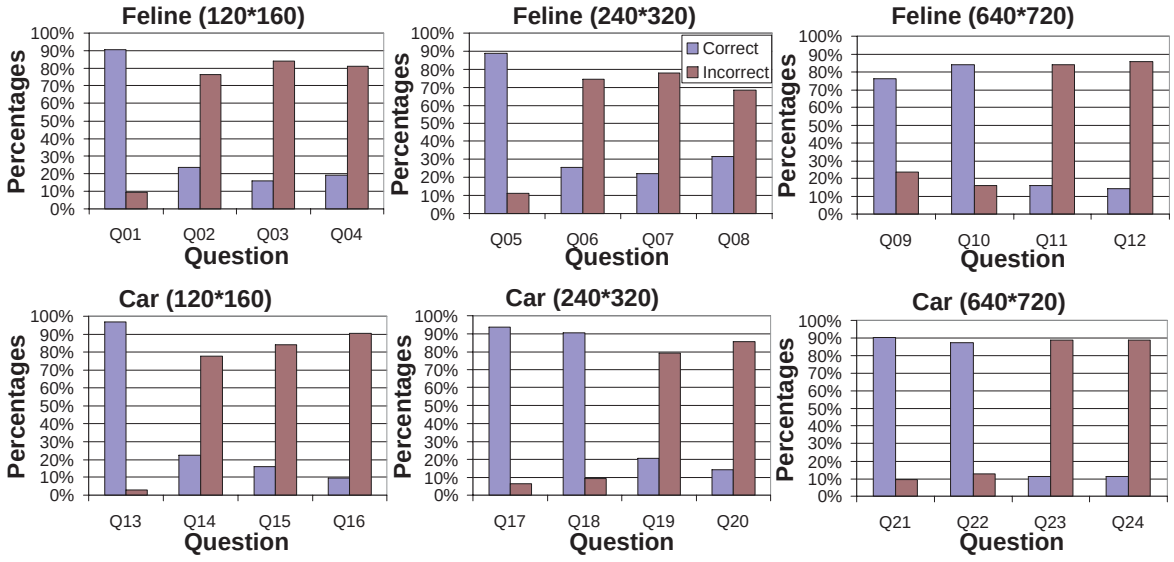


Figure 4.14: Sample results of the user study

in the 0-1 range and map the value of the per-pixel relative RMSE to colors that depict the magnitude of the relative RMSE at that pixel. Suppose relative RMSE is ρ, we use red when $\rho = 1$ (maximum error) and yellow when $\rho = 0$ (minimum error). This gives us equation 4.10 below:

$$(r,g,b) = (255,255,0) * (1-\rho) + (255,0,0) * \rho \tag{4.10}$$

Figure 4.15 shows the different error distributions of different screen sizes. As expected, regions of great undulation (small triangles for the mesh and fast changing RGB values of neighboring pixels for images) such as the face of the feline and wheels of the car, show the most errors when simplified. However, as we compare the same mesh resolution across different screen sizes, we notice that as the screen size reduces, the ranges (peaks and troughs) of the errors are reduced. This is probably due to the averaging or minification process. In contrast, in the larger screen sizes peaks are preserved and regions of error are more easily discerned. Information is lost on smaller screen sizes in this reduction process and the net result is that a smoother distribution of errors occurs and peaks are lost.

4.5.3 Resource Saving Using PoI

Battery energy, CPU cycles, memory and disk space are all resources that are scarce on mobile devices. Using a mesh or image at the PoI instead of its original resolution can improve usage of these resources. We measure encoding, transmission and decoding times, and quantify potential battery energy savings by using a lower resolution mesh. We measure the energy consumption of receiving, decoding and rendering a given mesh resolution on the mobile client by using our tool [10]. PowerSpy is a software tool that tracks the energy consumption of MS Windows applications at the thread and I/O device levels. We calculate the total energy consumption, E, by summing up the energy consumption of the CPU, disk and network cards giving $E = E_{CPU} + E_{Disk} + E_{Network\ Card}$.

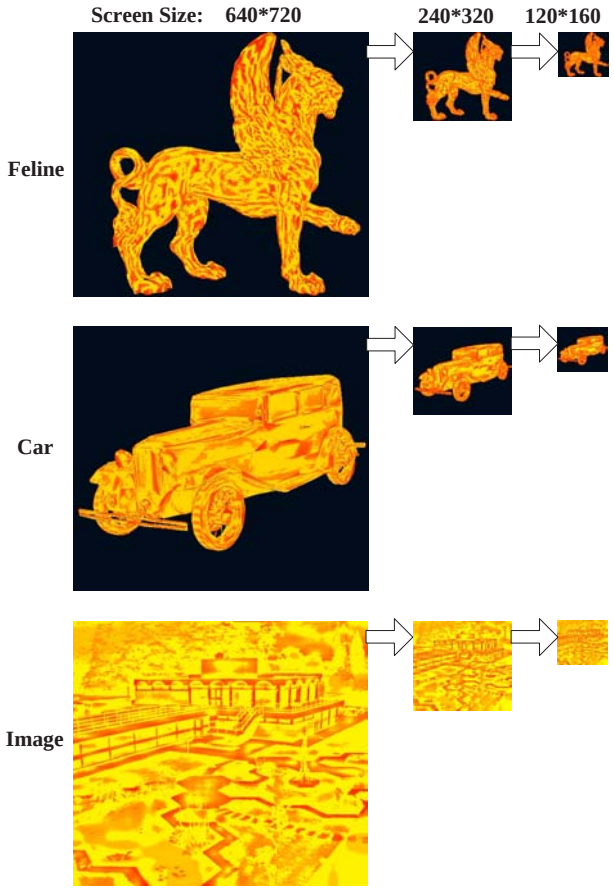


Figure 4.15: Error Distribution on a mesh and image

For a screen size of 640x720, our metric yields a PoI of 13654 faces for a bunny mesh, meaning that there is no significant perceptual difference if the number of faces is greater than 13654. If we use 13654 faces instead of the original mesh, the difference in resource usage is saved. Table 4.16(a) summarizes the saved transmission time, decoder time and energy consumption in the mobile device.

$Faces\ of\ Number$	13654	64951	Saved
$t_{trans.}$	1.23ms	7.03ms	82.5%
$RenderingTime$	463ms	832ms	44.4%
Energy Consumption	**12,865mwh**	**33,298mwh**	**61.4%**

(a) Saved Resources for bunny mesh

$Size\ of\ Coef.\ File$	64KB	173KB	Saved
$t_{trans.}$	47.6ms	120.5ms	60.5%
$RenderingTime$	340ms	530ms	35.8%
Energy Consumption	**7,467mwh**	**12,156mwh**	**38.6%**

(b) Saved Resources for image

Figure 4.16: Resource savings

Thus, using our perceptual metric, in this example we save over 80% of the transmission time, 44% of the decoding time and 61% of the total battery energy. Similar results for an image are tabulated in table 4.16(b). For the image in this example, it is possible to save over 60% of transmission time, 35% of the decoding time and 38% of the total battery energy. The above numbers on savings clearly depend on how large the original mesh (or image) is compared with the PoI. Our numbers are included above mainly for illustration purposes.

It is important to note that in calculating our PoI metric, the mesh is rendered from a single viewpoint. However, as an object is moved, different viewpoints will lead to different screen errors and PoIs. To make our metric view independent, in the server pre-processing step, the PoI can be calculated from multiple view points around the mesh. The PoI's generated from multiple viewpoints can then be averaged, maximum or the minimum used in a conservative scheme.

4.6 Chapter Summary

This chapter presents a wavelets-based multiresolution framework for scalable graphics content transmission and rendering. We present a Point of Imperceptibility (PoI) error metric that accurately picks the lowest acceptable mesh (or image)resolution based on the target mobile device's screen size. We develop versions of our PoI that considers only mesh geometry without considering lighting, as well as an extension that considers the effects of lighting on the perceptibility of distortion. We present LoD selection heuristics based on our proposed metric and analyzed the relative Root Mean Square Error (RMSE) our metric. We perform user studies to validate our metric, employed our metric in a heuristic to save mobile device resources and quanitified resulting resource savings.

Chapter 5

Unequal Error Protection for Wavelet-Based 3D Transmission

This chapter presents the research work for Unequal Error Protection(UEP) for Wavelets-Based Wireless 3D Mesh Transmission. This work has been published in [9] and has been submitted to [1].

5.1 Overview

To minimize transmission times on low-bandwidth network links, several compression [78, 79, 80, 103] techniques have been developed to reduce transmitted mesh sizes. Additionally, the wireless channel is well known to have significantly high error rates. Retransmission of damaged packets or Forward Error Correction (FEC) are two strategies that are frequently used to mitigate wireless channel errors. However, the roundtrip delays caused by retransmissions in network protocols such as TCP/IP and the IEEE 802.11 Wireless LAN protocol appear as latency to users, which sometimes affects the interactivity of networked graphics applications. For such applications, FEC is preferred to retransmissions. FEC schemes add redundant bits to the original meshes before transmission such that minor errors can be corrected by the receiver, hence avoiding retransmissions.

To speed up large mesh transmission over low-bandwidth wireless links, we have proposed a encoding meshes using wavelets prior to transmission in chapter 2, because wavelets achieve over 100x compression ratios and facilitates piece-wise transmission of large meshes. Using wavelets, a server only needs to send a small base mesh along with wavelets coefficients that refine it, saving memory and bandwidth.

As one of our main contributions, we propose a FEC scheme to protect wavelets-encoded meshes from wireless errors. The Hamming code [108] and Reed-Solomon [109] codes are two popular FEC schemes that mitigate error well for most applications. However, FEC schemes that consider the underlying structure of wavelets-encoded content frequently outperform more general schemes that do not. Wavelets-specific FEC techniques for image [110] and video transmission [111] have been proposed, but not for wavelets-encoded meshes. We propose FEC scheme based on the principle of Unequal Error Protection (UEP). In UEP [81], the number of FEC bits alloted to each part of the mesh is proportional to the amount of information it contains: more bits are added to parts with more information. Thus, areas of a mesh such as a human face that has many fine details are allocated more FEC bits than areas such as the back with less details. The rest of the chapter is organized as following: Section 5.2 describes our UEP scheme for wavelets-encoded meshes; Section 5.3 describes our channel model and simulation results; Finally, Sections 5.4 summarizes the chapter.

5.2 Unequal Error Protection of Wavelets-Encoded Meshes

5.2.1 Unequal Error Protection

Approaches to mitigate wireless channel errors packets losses can be network-oriented solution such as retransmissions in TCP, post-processing solutions such as error concealment, or pre-processing solutions such as FEC codes. The roundtrip delay incurred make retransmissions unsuitable for real-time graphics applications. In multicast environments, retransmissions would also flood the sender with acknowledgements and performance could suffer. We

consider the use of FEC. FEC strategies include Equal Error Protection (EEP) and Unequal Error Protection (UEP). EEP methods apply the same FEC code to all parts of the mesh's bit stream and is suitable when the channel has a low packet loss rate. However, at higher packet loss rates, considerable degradation on the decoded model quality may occur because of the high possibility that important parts might be lost. In this case, UEP is more suitable since important parts of the decoded mesh get more assigned more FEC bits. Figure 5.1 shows if the information in base mesh lost, the holes will happen after rendering. But if some coefficients lost, the LoD will decrease after rendering.

In our approach, after applying wavelets decomposition to a mesh, the base mesh as well as wavelets coefficients are assigned an FEC code rate depending on their contribution to the decoded mesh quality. The distribution of these FEC codes is calculated using a statistical distortion measure. Based on this measurement, we determine the number of error-protection codes to be assigned to the base mesh and each level of detail. The FEC codes used in our methods are Reed-Solomon (RS) codes. These error codes are perfect for error protection against bursty packet losses because they are maximum distance separable codes. An (n,k) RS-code encodes k information symbols where each symbol is represented by q bits. These k symbols are encoded into a codeword of n symbols, which is restricted by $n \leq 2^q - 1$. As soon as k symbols are received, all lost symbols can be reconstructed.

5.2.2 UEP in Wavelets-Based Multiresolution

After wavelets decomposition, the base mesh and first few levels of wavelets coefficient tree should be strongly protected to prevent packet loss. We examine several strategies for adding FEC bits to the base mesh and wavelets coefficients. First, we apply Equal Error Protection where an equal number of FEC bits are applied to all parts of the base mesh and to all levels of the wavelets coefficient tree. That is, $S^1 = S^2 = ... = S^{M+1}$, where S^k is the number of FEC bits added to on the k^{th} level of wavelets coefficients. Next, we propose applying Unequal Error Protection where bits in the encoded mesh are classified based on their contributions to

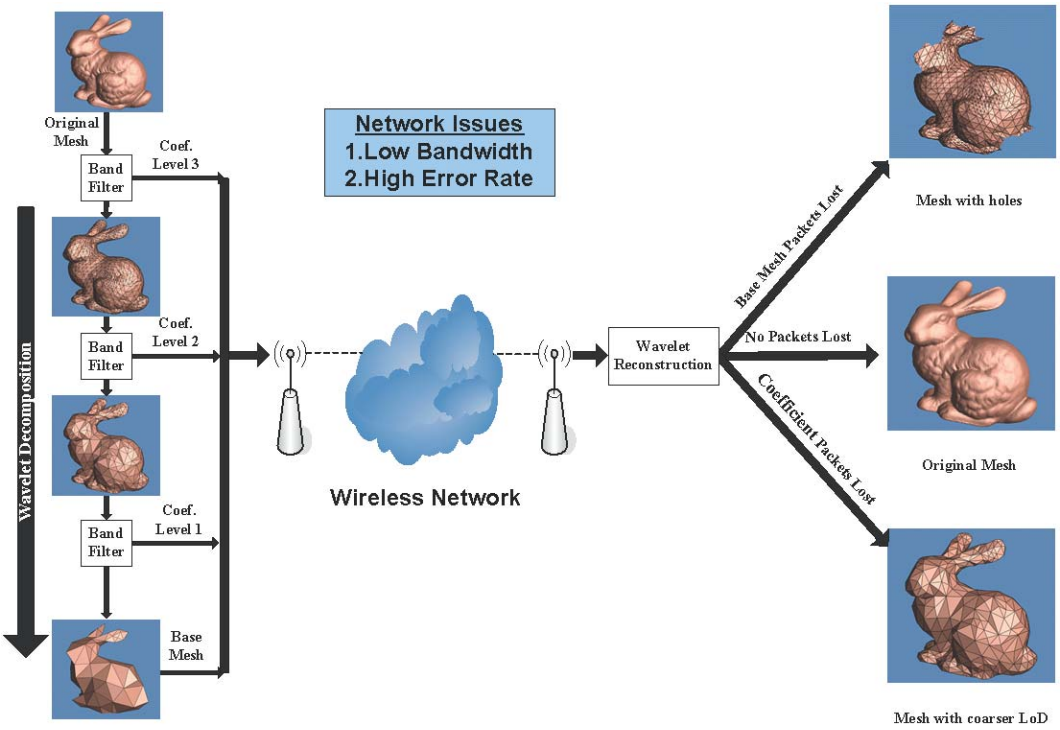


Figure 5.1: The importance of different level

the final look of the reconstructed mesh. Each class is then protected by a number of FEC bits that can provide a certain level of protection against channel losses. In our research, each level of the wavelets coefficient tree and the base mesh, is assigned an FEC code based on amount of distortion that would be introduced into the reconstructed mesh if that portion of the bitstream is lost. Parts of the bitstream that distort the look of the reconstructed mesh most when they are lost are the most important and hence we apply the largest portion of the FEC bit budget. Wavelets coefficients with large absolute values contain the most detail receive more error bit budget, since this level of coefficients contains more information (e.g. fine details such as eyes and nose of a face) compared to other levels. The FEC codes used are the Reed-Solomon (RS) codes. Reed-Solomon codes are block-based error correcting codes with a wide range of applications for error protection against burst packet losses. We also adapt our encoding order of our bitstream to further increase resilience to burst errors. The output bitstream is encoded in blocks of packets, where the data is placed in horizontal packets and then RS is applied across the block of packets vertically. Each block of packets

is protected with a FEC code that is proportional to the importance of the corresponding base mesh or coefficients.

Since all types of error protection add extra bits to the original mesh bitstream prior to transmission, both EEP and UEP incur overheads that reduce the number of actual data bits sent compared with No Error Protection (NEP). However, since reconstruction starts from the base mesh, loss of the base mesh or parts of it are particularly devastating. Essentially, the base mesh as well as coarser wavelets coefficients are more important than detail coefficients. At high packet loss rates, losing the base mesh or coarser wavelets coefficients degrades the decoded mesh quality significantly even if the detail coefficients are received correctly. EEP distributes error correction bits equally to the base mesh, and all levels of detail coefficients.

5.2.3 Distortion Measure

To determine the level of channel coding associated with each level of the wavelets coefficient tree, we need to evaluate the importance of those coefficients. In this section, we develop a distortion metric that evaluates the relative importance of the various levels of a wavelets coefficient tree. After we determine the importance of each level of the wavelets coefficient tree, we can then assign a fraction of the total FEC bits that is proportional to their importance. The main factors integrated into this distortion measure are: 1) The amount of information contained in the wavelets coefficient, 2)the total number of error-protection bits. As figure 5.2 shows, in each LoD, some new coefficients are added to the mesh, which provide more detailed information to the final rendered mesh.

To calculate the importance of each level of the wavelets coefficient tree, we evaluate the distortion that would be present in the final decoded mesh if all the coefficients in that level of the tree were lost. We associate a coefficients distortion quantity, $D_{wLOD}^{(j)}$ with the j^{th} LOD, which is defined as the average distortion (per coefficient) added when all coefficients that are added by this LOD are lost. The $D_{wLOD}^{(j)}$ is given by:

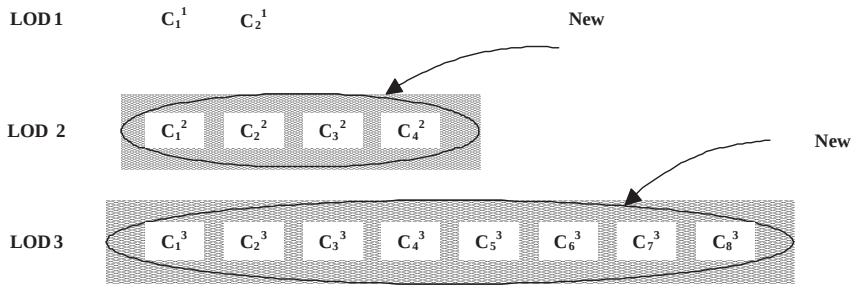


Figure 5.2: Wavelets coefficient tree for a mesh with three LODs. C_i^j is the wavelets coefficient at level j.

$$D_{wLOD}^{(j)} = \frac{1}{N_j} \sum_{i=1}^{N_j} |c_i^j| \tag{5.1}$$

Where N_j is the number of coefficients added by $LOD^{(j)}$. This distortion measure estimates the error between the meshes with the j^{th} LOD and the $(j+1)^{th}$ LOD. We use this distortion measure to calculate the fraction of the total error protection bit budget that is assigned to each level in UEP. In EEP, the available error protection bit-budget can be calculated as follows:

$$S = \sum_{j=1}^{M+1} (n-k) \times q \times B^j \tag{5.2}$$

where q is the codeword size. B^j is the number of codewords in each horizontal packet. In the case of UEP, the bit-budget, S, and the total packet size, n, are provided. Therefore, the RS code rates for all M layers need to be computed. Let α^j be the portion of the total bit-budget to protect j^{th} level of decoded mesh. That is, $\alpha^j = \frac{S^j}{S}$. So the j^{th} level bit-budget is given by:

$$(n-k_j) = \frac{\alpha^j \times S}{q \times B^j} \tag{5.3}$$

From Equation 5.3, we know α^j is the main factor to determine the RS code rate. We set α^j to be equal to the coefficients distortion quantity, $D_{wLOD}^{(j)}$ which was given in Equation 5.1.

In this way, we can calculate RS code $(n-k_j)$ using Equation 5.3 for each part of decoded mesh.

5.2.4 Block-based Encoding:

To further increase the error-resilience of our transmitted meshes, we apply block-based encoding after UEP encoding, before transmission. A simple example of our approach to block-based error correcting is described. Consider a 3D model that has been decomposed into a base mesh and three levels of wavelets coefficients (L_1, L_2 and L_3). Applying RS codes, the resulting packets are shown in Figure 5.3. The base mesh consists of five data packets with five error protection packets. The wavelets coefficients corresponding to level one, L_1, consists of six data packets with four error protection packets. Wavelets coefficient level L_2 consists of eight data packets with two error protection packets and level L_3 consists of ten data packets with no error protection packets. The base mesh and its associated RS packets are transmitted first, followed by the coarse wavelets coefficients, until the finest one. As shown in Figure 5.3, more FEC codes are assigned to the coarser level of coefficients than the finer one. Such an allocation of FEC codes is calculated by a distortion quantity that is described above. At a certain packet loss rate, some of the packets will be lost. Taking an example of three packets for each block being lost. Since the base mesh uses (10,5) error correction codes, when the number of lost packets is not more than five, the client can recover all lost packets. Therefore, in this example, it can recover all three lost packets. For the same reason, all three lost packets in L_1 can be recovered. But the lost packets in L_2 and L_3 can not be recovered by the assigned RS codes. At the client, the base mesh and L_1 level of coefficients have adequate protection but L_2 and L_3 levels of coefficients get lost. Therefore, the more important parts of the mesh are protected, are correctly received by the client and decoded even when the wireless channel loses a significant number of packets.

Figure 5.3: Example of transmitted packets in unequal error protection methods

5.3 Result

In this section, we describe tests that we conducted using meshes to evaluate the performance of our method. In particular, the performance of the UEP, EEP and NEP are compared. First we describe a two-state Markov model known as the G-E model [88] for the wireless channel.

5.3.1 Channel Model

We use a Markov model with only two states to model a wireless channel with high bit error rates [88]. We shall now briefly describe its main characteristics.

G-E models are defined by the distribution of error-free intervals, which are called gaps. The gap is defined as the interval of length $v-1$ packets between two consecutive received error packets. This model is illustrated in figure 5.4 and the probability density function (pdf) $g(v)$ and cumulative distribution function (cdf) of the gaps greater than $v-1$ packets $G(v)$ are defined as equation 5.4 and equation 5.5, respectively.

$$g(v) = \begin{cases} 1 - P_{BG} & , \quad v = 1 \\ P_{BG}(1-P_{GB})^{v-2}P_{GB} & , \quad v > 1 \end{cases} \tag{5.4}$$

$$G(v) = \begin{cases} 1 & , \quad v = 1 \\ P_{BG}(1-P_{GB})^{v-2} & , \quad v > 1 \end{cases} \tag{5.5}$$

Let $R(m,n)$ denote the probability of having $m-1$ packet losses within the $n-1$ packets

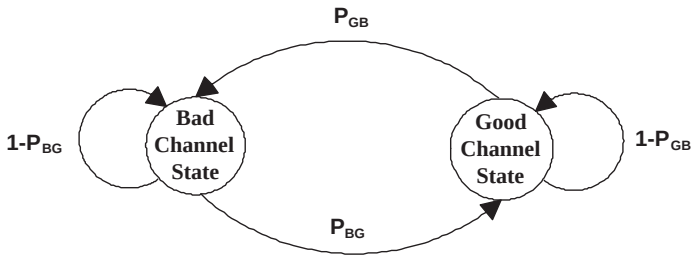


Figure 5.4: G-E two state Markovian Channel Model. P_{GB} is the transition probability from the good state to the bad state while P_{BG} is the transition probability from the bad state to the good state

following a lost packet. Then $R(m,n)$ is given by:

$$R(m,n) = \begin{cases} G(n) & , \quad m = 1 \\ \sum_{v=1}^{n-m+1} g(v)R(m-1,n-v) & , \quad 2 \leq m \leq n \end{cases} \tag{5.6}$$

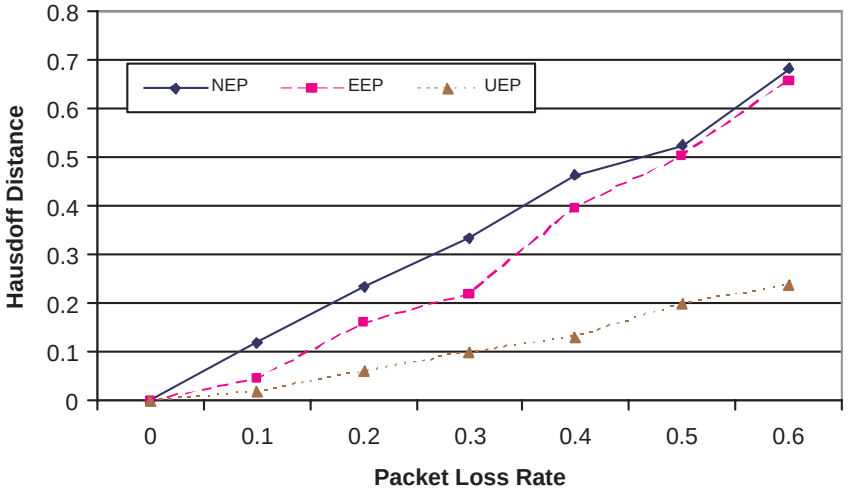
So, the probability of losing m symbols, each of which is of q bits in length, within a block of n symbols can be written as:

$$p(m,n) = \begin{cases} \sum_{v=1}^{n-m+1} P_B g(v)R(m,n-v+1) & , \quad 1 \leq m \leq n \\ 1 - \sum_{m=1}^{n} p(m,n) & , \quad m = 0 \end{cases} \tag{5.7}$$

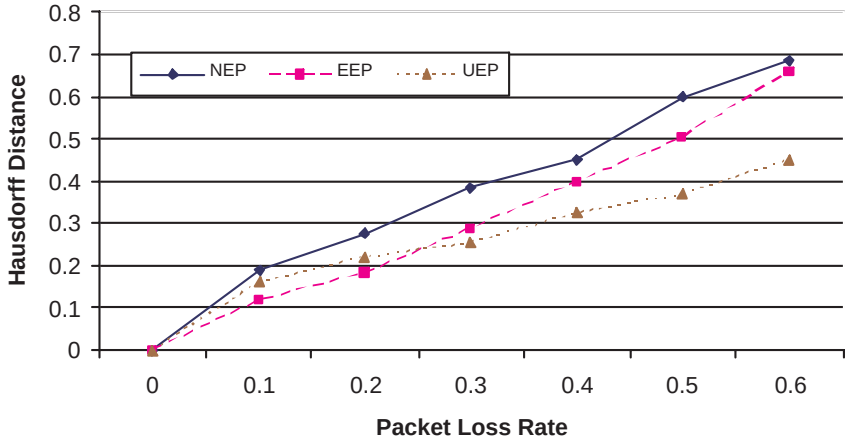
5.3.2 Simulation Results

We applied the proposed unequal error protection (UEP) method on several models and here we report the results for the small bunny mesh. We consider three cases: encoding the original mesh into a base mesh and 5 levels of detail, 10 levels of detail and 15 levels of detail. In general, the more levels of detail we use, the less information each layer contains. We use the Hausdorff distance to measure the amount of distortion in our received mesh. The Hausdorff distance expresses the geometric distance between two surfaces as the maximum of all pointwise distances. In general, more distortion increases the Hausdorff distance.

Figure 5.5 depicts the distortion as a function of the packet loss rate for the small bunny model. Three curves in this figure represent the cases of EEP, UEP, and NEP with level 5. As



a



b

Figure 5.5: Maximum Error(Hausdorff distance) between the transmitted and the decoded mesh when the RS code used for EEP is a: (n,k) = (63,45) and b: (n,k) = (63,51). NEP: no error protection is applied, EEP: equal error protection is applied, and UEP: unequal error protection is applied

can be seen from these curves, for an error-free channel no packets are lost and the distortion in the transmitted mesh is zero. As the packet loss rate increases, the performance of EEP and NEP become closer to each other since neither technique can recover when packets of the base mesh or coarse level of coefficients are lost. However, UEP manages to protect the base mesh and coarse wavelets coefficients by assigning more error-protection bits and therefore

improving the quality of the decoded mesh quality is better compared to other two methods. When the packet loss rate $P_{LR} \geq 0.2$, the base mesh information is lost and only UEP is able to protect the base mesh.

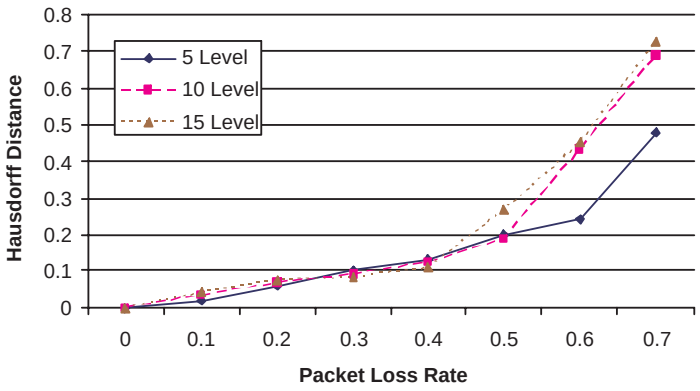


Figure 5.6: Maximum Error(Hausdorff distance) between the transmitted and the decoded mesh when different level of detail (5,10,15) are used with RS code (n,k) = (63,45)

Figure 5.6 shows the distortion as a function of the packet loss rate for the small bunny mesh. Three curves in this figure represent the cases of 5, 10, 15 levels of detail. The figure shows a slow increase in the Hausdorff distance up till a knee point at which the Hausdorff distance (or distortion) increases quickly. Before the knee point, only wavelets coefficients are lost while the base mesh is correctly received. Beyond the knee points the high error rates cause the base mesh to get lost, causing a large increase in distortion (Hausdorff distance). The knee point of the 5-level LoD is larger (more resilient to errors) than that of the 10-level and 15-level LoDs. This is intuitive since as the mesh is encoded into more LoD levels, each level of the wavelets coefficient tree as well as the base mesh all receive fewer error protection bits. Hence, meshes that are encoded into more LoD levels will lose the base mesh information easier than meshes encoded with fewer LoD levels. Thus for a fixed UEP bit budget, we find an inverse relationship between the number of mesh LoDs used and the error resilience of the wavelets-encoded mesh. Before the knee points, the base mesh is received and only wavelets coefficients are lost. As the mesh is encoded into more LoDs, the importance of each level of the wavelets tree level is reduced and the degradation introduced

when wavelets coefficients are lost are also reduced. Therefore, before the knee point, the distortion of the meshes encoded with more LoDs is slightly lower than that of meshes that use fewer LoDs.

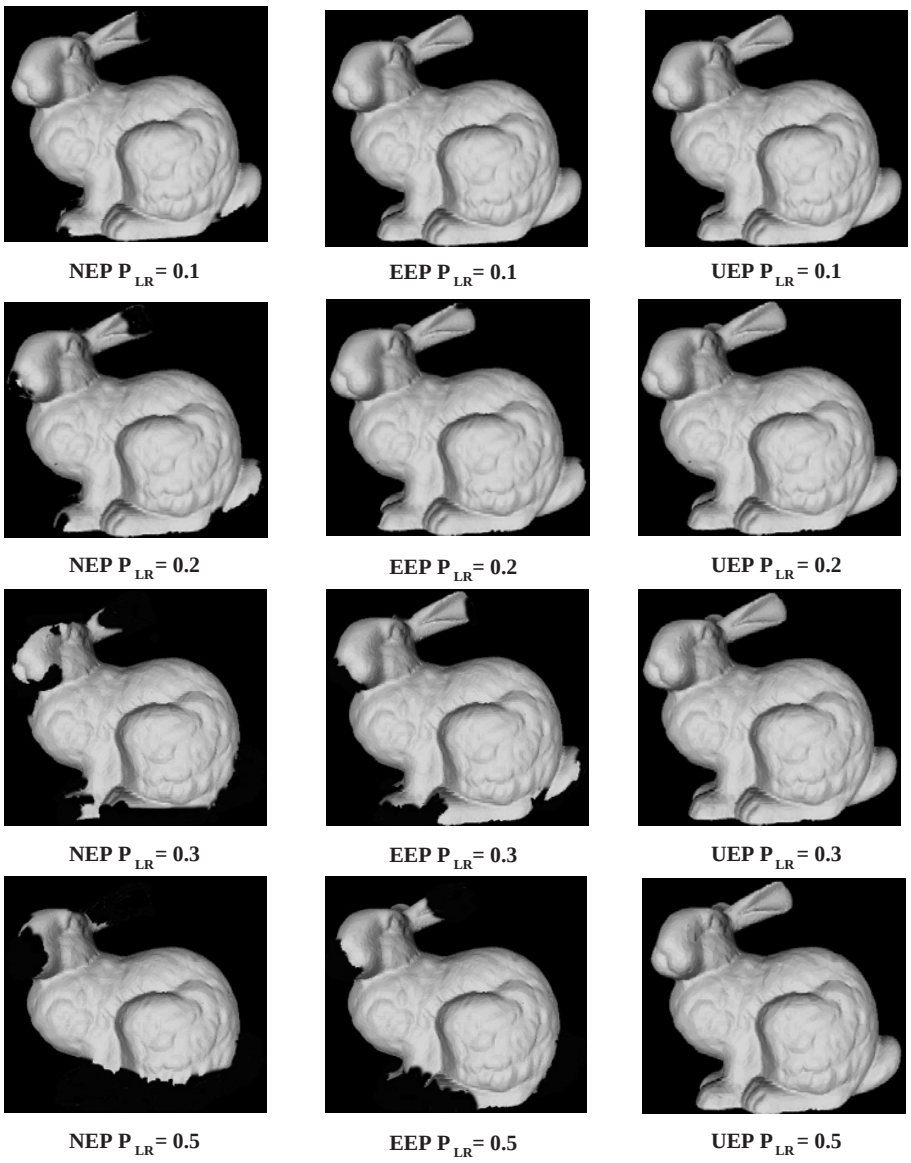


Figure 5.7: Subjective results of applying no error protection (NEP), equal error protection (EEP), and unequal error protection (UEP) methods on the SMALL BUNNY model. The caption under every image gives the error protection method and the packet loss rate of the channel. RS code (n,k) = (63,45)

Objective results have been presented above. We also compare the three methods, NEP, EEP, and UEP, subjectively by looking at images of the final rendered mesh after passing

them through a simulated wireless channel. Figure 5.7 shows the experimental results for the small bunny mesh. The first column on the left shows the decoded mesh in the NEP case for different packet loss rates. Similarly, the second and the third columns show the decoded meshes for EEP and UEP respectively. As shown, UEP maintains a reasonable decoded mesh quality as the packet loss rate increases. We have encoded the mesh into 5 Levels of Detail. As the error rate increases, UEP loses some detail coefficients but the base mesh and coarse coefficients are adequately protected and correctly received. Hence, only minor artifacts can be observed on the UEP as error rates increase. We can thus conclude that using our proposed UEP method on wavelets multiresolution, the quality of the decoded meshes is better as the packet loss rate increases.

5.4 Chapter Summary

This chapter presents Unequal Error Protection (UEP), a Forward Error Correction (FEC) scheme for the error-resilient transmission of meshes that have been encoded using wavelets, to increase decoded mesh quality. Error-protection bits are allocated according to the importance of parts of the wavelets-encoded mesh. The importance of each level is determined by a distortion measure that reflects the information the coefficients contain. Theoretically, the UEP method increases the resilience of wavelets-based mesh transmission to high error rates. By simulating mesh transmission using our proposed scheme on two different channel models, we compare the performance of the proposed UEP, EEP and NEP methods.

Chapter 6

Energy-efficient Adaptive Real-time Rendering Heuristic

This chapter presents the research work for Energy-efficient Adaptive Real-time Rendering (EARR) heuristic. This work has been published in [3, 4].

6.1 Overview

The most limiting resource on a mobile device is its short battery life. While mobile CPU speed, memory and disk space have grown exponentially over the years, battery capacity has only increased 3-fold in the past decade. Consequently, the mobile user is frequently forced to interrupt their mobile graphics experience to recharge dead batteries.

Application-directed energy saving techniques have previously been proposed to reduce the energy usage of non-graphics mobile applications. Our main contribution is the introduction of application-directed energy saving techniques to make mobile graphics applications more energy-efficient. The main idea of our work is that energy can be saved by scheduling less CPU timeslices or lower the CPU's clock speed (Dynamic Voltage and Frequency Scaling (DVFS)) for mobile applications during periods when its requirements are reduced.

In order to vary the CPU timeslices alloted to a mobile application, we need to accurately

predict its workload from frame to frame. Workload prediction is a difficult problem since the workload of real-time graphics applications depends on several time-varying factors, such as user interactivity level, the current Level-of-Detail (LoD) of scene meshes and mip-mapped textures, visibility and distance of scene models, and the complexity of animation and lighting. Without dynamically changing the application's CPU allotment to correspond to its needs, the mobile application's frame rate fluctuates whenever there is a significant change in scene LoD, animation complexity, or other factors that affect its workload. Such spikes above 25-30 Frames Per Second (FPS) drain the mobile device's battery and increased energy consumption by up to 70% in our measurements (see figure 6.1). We propose an accurate method to predict the mobile application's workload and determine what fraction of the CPU's cycles it should be alloted to maintain a frame rate of 25 FPS. As the application's workload changes, we update its CPU allotment at time intervals determined by a windowing scheme that is sensitive to applications with fast-changing workloads and prudent for applications with slow-changing workloads. Our adaptive CPU scheduling scheme dampens frame rate oscillations and saves energy.

Many techniques have been proposed to achieve three desirable qualities of mobile graphics: photorealism, real-time rendering and energy efficiency. For instance, Level-of-Detail (LoD) management allows scenes to be rendered at real-time speeds while maximizing visual realism. Also, intelligent scheduling and application-directed Dynamic Voltage and Frequency Scaling have been proposed to save energy on mobile devices. While these techniques work if applied separately, they can create conflicts when they are integrated into the same graphics framework. Specifically, techniques that improve one attribute can degrade another. For instance, improving image quality requires increasing mesh LoD, which need more CPU cycles and memories accesses which kills (degrades) the mobile devices battery. Essentially, we can think about these three attributes as orthogonal axes. Ideally, we would like to make progress along all three axes. However, in practice, proposed techniques have fundamental limitations that allow them to only make progress along one or two axes but typically not all

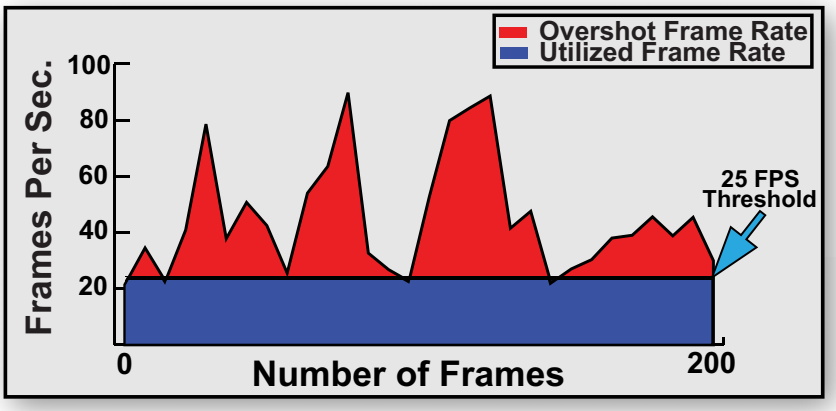


Figure 6.1: Application running at high frame rate

three axes (See table 6.1 for examples).

Since the application's workload changes and should be re-estimated whenever LoDs are switched, we have coupled our CPU scheduler with the application's LoD management scheme. When switching scene LoD, we minimized energy consumption by selecting the lowest LoD at which the user does not see visual artifacts, also known as the Point of Imperceptibility (PoI) [6]. Although our primary goal was to minimize the mobile application's energy consumption, we also ensured that the frame rates and visual quality of the rendered LoD were acceptable. In summary, our integrated EARR (Energy-efficient Adaptive Real-time Rendering) heuristic minimizes energy consumption by i) selecting the lowest LoD that yields acceptable visual realism, ii) scheduling just enough CPU timeslices to maintain real-

Technique	***Realism***	***Rendering Speed***	***Energy Efficiency***
LoD Reduction	⇓	⇑	⇑
Voltage Scaling	⇓	⇓	⇑
Frequency Scaling	⇓	⇓	⇑
CPU Scheduling	⇓	⇓	⇑
Ray tracing	⇑	⇓	⇓
Complex (HDR) lighting	⇑	⇓	⇓
Complex material (BRDF)	⇑	⇓	⇓

Table 6.1: Proposed techniques improve one or two desirable mobile graphics attributes while degrading the third one.

time frame rates (25 FPS). EARR also switches scene LoD to compensate for workload changes caused by animation, lighting, user interactivity and other factors outside our control. To the best of our knowledge, this is the first work to use CPU scheduling to save energy in mobile graphics. Our results on animated test scenes show that CPU scheduling reduced energy consumption by up to 60% while maintaining real time frame rates and acceptable image realism. The rest of the chapter is organized as following: Section 6.2 presents our heuristic architecture and overview of our approach; Section 6.3 presents our windows-based workload predicting model. Section 6.4 presents the CPU scheduling policy; Section 6.5 presents the complete heuristic in detail; Section 6.6 describes our experimental results; Finally, Sections 7.4 summarizes the chapter.

6.2 Our Approach

6.2.1 Heuristic Architecture

Our framework includes components for monitoring application frame rate and the rendered appearance of a selected mesh LoD, as well as a component for allocating CPU resources to our mobile graphics application. Our adaptation algorithm balances desired attributes using these components, which is shown along with our system architecture in figure 6.2. The energy monitor measures system-wide energy consumption.

6.2.2 Overview of EARR Heuristic

Our approach is a generalization of the predictive strategy. We predict the LoDs that will be rendered at the speed threshold of 25 frames per second. Within a real-time application such as a game, LoD is just one of many factors that affect the application's frame rates. Other aspects include lighting, texturing, system animation, artificial intelligence and networking of the application. In fact, in complex real-time graphics application such as a game or flight simulator, it is difficult to accurately model and predict all factors that affect observed frame

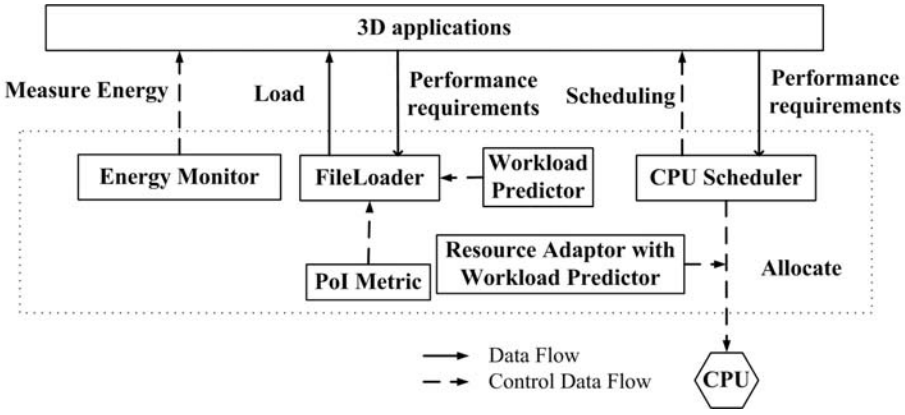


Figure 6.2: Heuristic Architecture

rates including when the user will interact with the scene or to anticipate the animation paths of meshes. We can not hope to consider all of these complex factors that can be computed efficiently. However, using efficient workload predict model, we have developed approximate heuristics that are both efficient to compute and accurate enough to be useful. Our algorithm takes actions such as switching mesh LoD or CPU allocation to compensate for the demands of game components outside its control, such that the frame rate of the entire application remains within the threshold frame rate.

More formally, we define $Energy(O,S,R)$, to be the energy required required to render an instance of a mesh or object *O*, rendered in the scene *S*, with adaptive algorithm *R*. Our approach can be stated as:

$$\begin{aligned} Minimize &: Energy(O,S,R) \\ Subject\ to &: Rendering\ frame\ rate \geq Threshold \quad (6.1) \\ Subject\ to &: Visual\ Realism \geq Threshold \quad (6.2) \end{aligned}$$

This formulation captures the essence of 3D graphics rendering on mobile devices with real-time constraints. Verbally stated, our goal is to reduce mobile device energy consumption as much as possible, while rendering the lowest LoD that meets the PoI (visual realism)

within the target frame rate.

6.3 Workload Predicting Model

6.3.1 Overview

The workload predicting model predicts what fraction of available CPU timeslices should be alloted to our mobile application in order to render a given mesh LoD or scene at our target frame rate of 25 FPS. We derive our predicting model in two parts. The first part predicts the workload of a single mesh object. Since most real world scenes consist of multiple objects, as a next step, we extend our workload predicting model to estimate the workload of complex scenes with multiple objects.

In general, as a given mesh is rendered faster, more CPU timeslices are consumed per unit time, and more battery energy is expended with no improvement in visual realism. Thus, to minimize energy consumption, the goal of the CPU scheduler is to allot *just enough* CPU cycles to finish rendering each frame *just before* its deadline expires. We strived to maintain a frame rate of 25 FPS, which means that each frame should finish rendering within a deadline of 40 milliseconds. Based on this deadline, if the rendering time of each frame using a particular LoD is estimated to be 20 milliseconds when 100% of CPU resources are alloted to our mobile graphics application, then the alloted CPU resources (and rendering speed) can be halved without exceeding the frame's deadline. The optimal (fewest) CPU resources C_{opt} to meet our task's deadline can be expressed as:

$$C_{opt} = \frac{\tau}{r_{max}} \times C_{max} \tag{6.3}$$

where C_{max} is the maximum available allotment of the processor's timeslices, C_{opt} is a reduced allotment of CPU timeslices generated by our algorithm, which just meets the frame's deadline. r_{max} is the rendering time of a mesh with all available processor cycles

alloted to our application and τ is the deadline for the frame. Since our target frame rate is 25 Frames Per Second, we set τ, the deadline for each frame, to 40 ms.

We apply our workload predictor as follows. At runtime, given a frame rendering deadline, τ, we use equation 6.3 to calculate the optimal CPU processor allotment, C_{opt}. We then use our pre-generated statistics to estimate the mesh LoD that corresponds to C_{opt}.

For our workload predictor to be successful, we derive our predicting model in two parts. The first part predicts the workload of a single mesh object. Since most real world scenes consist of multiple objects, as a next step, we extend our workload predicting model to estimate the workload of complex scenes with multiple objects.

6.3.2 Workload Predicting Model for a single Object

Given a scene, we would like to use certain observable features to predict its rendering time. Funkhouser and Sequin [120] previously suggested that the number of triangles in a mesh was a good predictor of its rendering time. To examine how accurately the number of triangles in a mesh predicts of its rendering time, we set up experiments to study how correlated rendering times are with mesh LoD. In a offline calibration pre-process, various meshes (bunny, feline, venus) were rendered at different LoDs and statistics were collected on their rendering times and corresponding processor demand for each LoD. To formally establish the degree of correlation between mesh LoDs and their rendering times, we calculated the first and second order statistics of measured rendering times and triangle counts.

Let x and y be two random variables corresponding to the mesh size(number of triangles) and rendering time, respectively; and let μ_x and σ_x be the mean and standard deviation of the mesh size; and also let μ_y and σ_y be the mean and standard deviation of the rendering time, respectively. Thus the theoretical correlation coefficient ρ_{xy} between x and y is given by:

$$\rho_{xy} = \frac{E[(x-\mu_x)(y-\mu_y)]}{\sigma_x \sigma_y} \tag{6.4}$$

Now assume we have N experimentally measured pairs of x and y values. The correlation coefficient ρ_{xy} may be estimated from these N pairs of data as:

$$r_{xy} = \frac{\sum_{i=1}^{N}(x_i - \overline{x})(y_i - \overline{y})}{[\sum_{i=1}^{N}(x_i - \overline{x})^2 \sum_{i=1}^{N}(y_i - \overline{y})^2]^{1/2}} \tag{6.5}$$

In general, a correlation coefficient of 1.0 is the highest achievable value and implies that given a value of x, the corresponding value of y can be predicted with 100% accuracy. Figure 6.3 shows three meshes (bunny, feline and venus) with calculated correlation coefficients, respectively. These results show strong correlation between mesh LoD and rendering time. In fact, there is a linear relationship between the number of triangles and rendering time, which corroborates corroborate the results of Funkhouser and Sequin [120]. The slope of this linear relationship depends on the mesh features and how powerful the machine on which it is rendered is. Thus, for a particular mesh and rendering machine, the slope and intercept of the linear function can be determined during pre-processing by rendering the same model at n LoDs, and graphing measured rendering time versus the number of triangles. Depending on its features, different meshes produce functions of different slopes. For instance, increasing the LoD of a complex model by 1000 triangles yields a larger increase in its rendering time than if the LoD of a simpler mesh were increased by 1000 triangles. Hence, complex models have steeper slopes than simple models. For example, the feline model is more complex than the bunny mesh, and thus yields a steeper slope.

Finally, using observed data points of rendering times for different LoDs, we can use linear regression to generate a line of best fit. Let s_i and rt_i denote the number of triangles

a. Bunny model (r_{xy}=0.9343) **b. Feline model (r_{xy}=0.8974)** **c. Venus model (r_{xy}=0.9438)**

Figure 6.3: Sample Meshes and their Correlation Coefficients

and rendering time of the *i*th LoD, respectively, with all available CPU cycles alloted to our mobile graphics application. Thus the slope (b) and intercept (a) of the line of best fit are given as:

$$\begin{aligned} \overline{s} &= \frac{\sum_{i=1}^{n} s_i}{n} \\ \overline{rt} &= \frac{\sum_{i=1}^{n} rt_i}{n} \\ b &= \frac{\sum_{i=1}^{n} (s_i - \overline{s}) rt_i}{\sum_{i=1}^{n} (s_i - \overline{s})^2} \\ a &= \overline{rt} - b\overline{s} \end{aligned} \tag{6.6}$$

Figure 6.4 shows a sample best fit line. This line of best fit is used in our workload predictor. To charactize the overall accuracy of our workload predictor, the relative error between actual measured rendering times and predicted rendering times produced by our workload predictor, was calculated for various LoDs. Figure 6.5 is a plot of calculated relative error corresponding to different LoDs. The figure shows that our workload predictor is reasonably accurate where all relative errors are less than 5%.

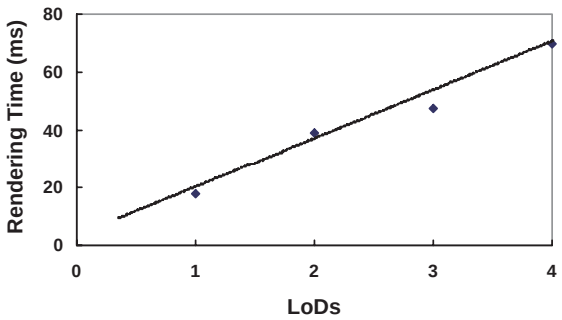


Figure 6.4: Sample Best Fit Line

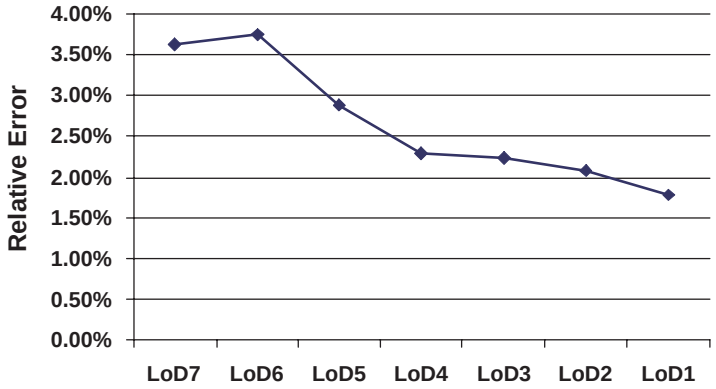


Figure 6.5: Relative Error between actual and estimated rendering times

6.3.3 Workload Predicting Model for Multiple Objects

Note that in a real game or application, there are typically many objects at various LoDs. Our proposed method should also predict the workload of multiple objects in a game or applications.

In complex scene with multiple objects, the workload for rendering the scene depends on the visibility of objects in the scene, which can vary over time as objects and the camera move. Tens of thousands of polygons might be simultaneously visible from some observer viewpoints, whereas just a few can be seen from others. Thus, the rendering effort for a dynamic scene is proportional to the triangles that are visible. We used an *eye-to-object* visibility algorithm described in [126] to determine a set of potentially visible objects to be rendered in each frame. Thus, the workloads of all visible objects (as determined in section 6.3.2), are then linearly combined to generate the workload of the complex scene.

Next, we considered changes in the application's workload over time. Since application workload changes only slightly from one frame to the next (milliseconds), the workload of successive frames are highly correlated. Thus, we use the current frame's workload to predict the workloads of next *n* frames. We define the *window size* as the number of frames in the future *n* for which current frame's workload is used as an estimate. The choice of *n* affects the performance of our algorithm. If *n* is too small, then we need to updated workload value

too often and it will increase the computation overhead; If it n is too large, then the variance between the predicted and actual workload will be high, and the variance could be too high to be accepted. Therefore, in our predicting model, this window size (n) is updated adaptively at run-time. Figure 6.6 shows how the window size is updated in our predicting model, which is inspired by the Transmission Control Protocol (TCP) in networking. It starts from 2. At the end of window size time point, we check the workload error. If it is smaller than the threshold, then the window size is doubled or increased by 1. Normally, there will not be much workload change within 8 frames. Therefore, if the window size is less than 8, then we double window size, otherwise increase window size by 1. If the workload error is larger than threshold, which means the workload predicting value is not accurate, we reset the window size to 2 and update the workload predicting value with current actual workload. Figure 6.7 shows the working flow.

The adaptive workload predictor is then used to estimate the workload of each frame at full processor speed, so that we can get the fraction of available CPU timeslices required to render a frame at our target frame rate. We tested it with two scenes provided by the Benchmark for Animated RayTracing(BART) [121], The results are shown in figure 6.8. As we can see, the relative errors are both bounded in 0.18.

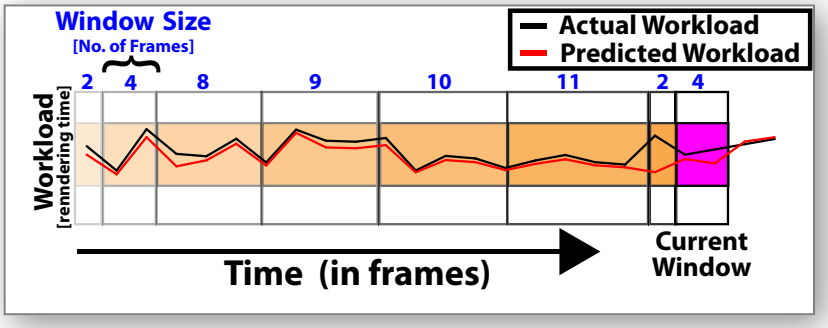


Figure 6.6: Window Size Updating

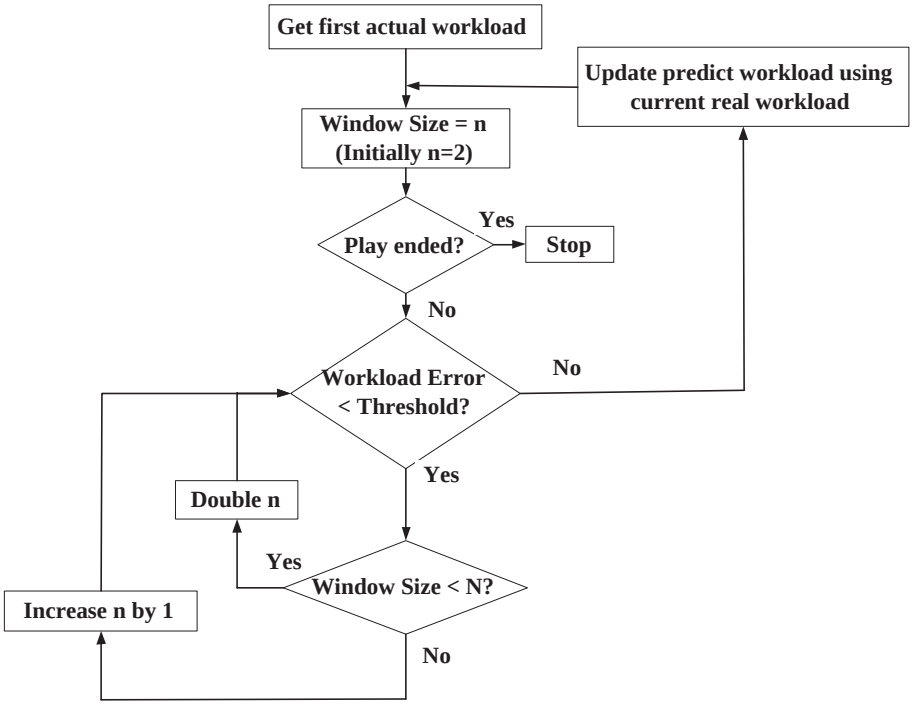


Figure 6.7: Flow Chat for Workload Predicting Model. In our work load predicting model, N=8

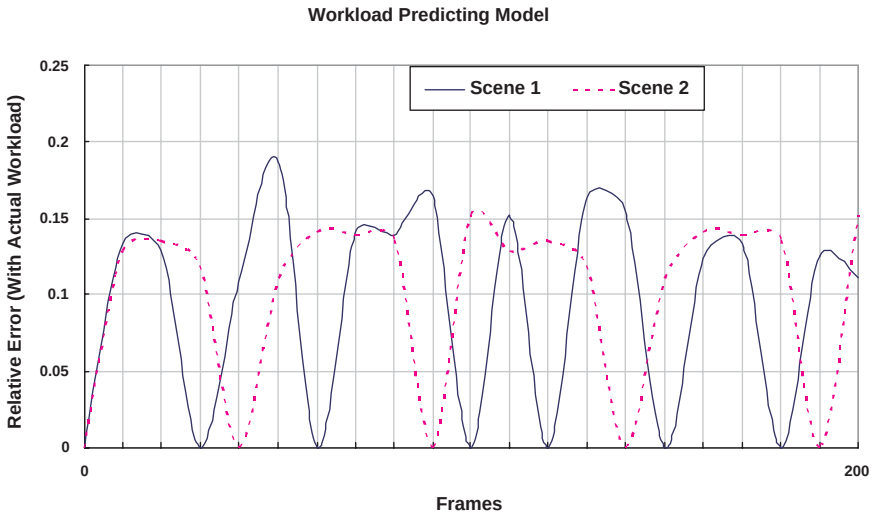


Figure 6.8: Workload Predicting Model

6.4 CPU scheduler

To conserve battery energy, our CPU scheduler runs a three-phase algorithm. The phases of the scheduler algorithm are workload estimation, estimating processor availability and determining processor resource allocation. More detail is now given on each of these steps. (1) *Estimated workload:* In this step, our workload predictor in section 6.3 is used to estimate how many CPU timeslices running the mobile graphics application will consume. (2) *Estimating processor availability:* Since the system may be running other applications or performing system house-keeping functions, the amount of CPU cycles available to our mobile graphics application varies over time. In this step, the amount of CPU resources currently available for applications is estimated. (3) *Determine processor resource allocation:* The last step chooses what fraction of available CPU resources are alloted based on the predicted workload and processor availability. For instance, if the predicted workload is only one third of the CPU resources available, then the mobile graphics application can save energy by using one third of available CPU resources. Likewise, if the predicted workload exceeds available CPU cycles, all available CPU cycles are allocated to the mobile graphics application and a lower mesh is selected to maintain a frame rate of 25 FPS.

We shall now formalize our CPU scheduling algorithm. For each real-time task T, let us denote its start time by t_s and its deadline as t_d. Let C_{max} denote the maximum fraction of CPU timeslices that are currently available for running applications. It is important to note that without the intervention of our scheduling algorithm, all tasks will run with 100% allocations of all available CPU timeslices, C_{max}. The number of processor timeslices required by T will be denoted by p. We note that the execution time of the task T is inverse proportional to p. In summary, a feasible schedule of the task guarantees that the task T receives at least a fraction, A, of the maximumum available CPU cycles such that it receives $A * C_{max}$ CPU cycles before its deadline, where $A \leq 1$.

Given the application workload p, maximum processor availability C_{max} and interactivity deadline t_d, as shown in figure 6.9, our policies to allocate processor resources fall into two

distinct cases that are now described.

Case 1: If $C_{max} < p$, then the application's demand for CPU timeslices exceeds CPU availability. In this case, the CPU schedule has allocated 100% of all available CPU resources to the task and cannot meet the task's deadline while using the current mesh LoD. Our scheduling algorithm shall allot all available CPU timeslices to the task and additionally reduces mesh LoD to lower the offered workload p.

Case 2: If $t_s + p < t_d$, the task can complete before its deadline. If all available CPU resources are alloted to this task, the rendering speed achieved is larger than 25 frames per second. In this case, the algorithm reduces the fraction of CPU timeslices alloted such that the demanded workload p is just adequate to complete the task before its deadline. The percentage of CPU resources alloted should be:

$$A = \frac{p}{min(C_{max}, t_d - t_s)} \tag{6.7}$$

In the equation 6.7, the deadline $t_d - t_s$ is known. In our case, we choose $t_d - t_s$ as 40 ms, p is determined by using our workload predictor. The maximum CPU resources currently available, C_{max} can be monitored by our resource adaptor.

Given an estimated $\hat{p}$ of the demanded workload and the maximum processor availability, $\hat{C_{max}}$, the optimal CPU resource allocation is computed as:

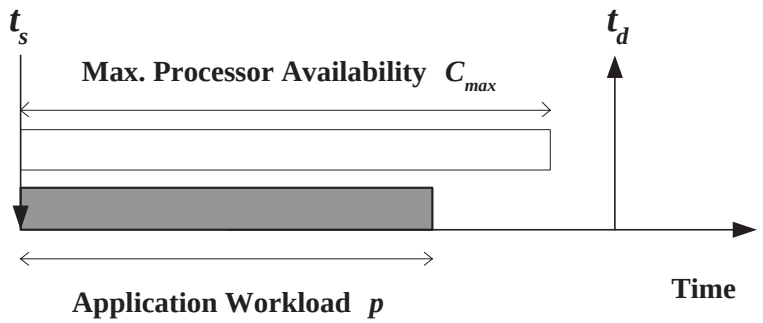


Figure 6.9: Symbols Illustration

$$C_{opt} = \begin{cases} C_{max} & : \quad \hat{C_{max}} < \hat{p} \\ min(C_{max} \times \frac{\hat{p}}{min(\hat{C_{max}}, t_d - t_s)}, C_{max}) & : \quad otherwise \end{cases} \tag{6.8}$$

6.5 EARR Heuristic

Building on our workload predictor and CPU scheduling policy, we now describe our complete optimization algorithm to balance application frame rate, visual realism and energy consumption constraints. Our algorithm monitors predicted frame rate and the rendered appearance of meshes and takes corrective action such as switching mesh LoD or changing the CPU resource allocation, when frame rate or LoD changes considerably.

Our optimization algorithm works as follows. At the start of the algorithm, the LoD of meshes corresponding to their PoI is selected for rendering. As the mesh moves during an animation, the algorithm reallocates CPU resources using the CPU scheduling policy discussed in section 6.4 and the workload predicting model discussed in section 6.3. If the predicted frame rate becomes less than 25 FPS, the algorithm chooses a lower LoD that increases application frame rate to 25 FPS. The optimal CPU allotment that minimizes energy consumption without affecting frame rate is then computed. The algorithm chooses the PoI LoD of the mesh for rendering when the adequate CPU resource can be alloted to render meshes at our speed threshold of 25 FPS.

There are three cases to which our heuristic is required to adjust the application parameters, each require different action. If we let d denote the current LoD of a mesh and d_p denote its PoI LoD. Let f denote the frame rate at which that mesh is currently being rendered. Essentially, there are three cases that our algorithm reacts to:

Case 1, predicted frame rate drops such that $f_i < 25$, current LoD i = minimum LoD possible, and 100% of CPU cycles already alloted to this task: In such a case, since we are at the limits of the factors under our control (minimizing LoD and maximizing CPU cycles), we conclude that it is impossible to meet the rendering speed threshold of 25 FPS. Essentially,

the resources of mobile devices are not enough to render the mesh and animation and we cannot rectify the situation. In such a scenario, we simply choose the minimum possible LoD and set the CPU cycles to a maximum and achieve the highest frame rate possible with this setting (best effort).

Case 2, predicted frame rate drops such that $f_i < 25$*, current LoD i = PoI,* d_p*:* In such a case, the algorithm will allocate more CPU resources to increase the rendering frame rate. If the rendering speed is still less than 25 FPS, the algorithm will choose a lower LoD level that can be rendered at 25 FPS and allocate the optimal fraction of CPU cycles, C_{opt} accordingly. We note that in this case, to achieve 25 FPS, we are forced to use an LoD below the mesh PoI, which introduces simplification artifacts.

Case 3, predicted frame rate increases such that $f_i >> 25$*, current LoD i = PoI,* d_p*:* the algorithm continues to use the PoI LoD but tries to save energy by reducing the percentage of CPU timeslices scheduled for our application to the minimum required to maintain a frame rate of 25 FPS. Figure 6.10 is the flow chart of our algorithm and the complete pseudocode of the algorithm is shown in algorithm 1.

6.6 Experiment and Results

6.6.1 Experiment

We extensively evaluated the performance of our proposed algorithm both a laptop and PDA. The laptop used was a Gateway 3040GZ laptop equipped with an Intel Celeron 1.5GHz processor and 512MB RAM. The laptop's operating system is Linux. The PDA is a HP iPAQ Pocket PC h4300 with a 400 MHz intel XScale processor and 64MB RAM. The operating system of the PDA is windows CE. We repeated all experiments eight times, eliminated the minimum and maximum values before averaging all other values. We animated a mesh bunny along a pre-determined animation path in a scene provided by the Benchmark for Animated RayTracing (BART) [121]. The test animation path was chosen because it is representative of

Algorithm 1 Balancing Heuristic in Animation

```
1: Choose the PoI {Rendering the possible lowest LoD without perceivable difference}
2: if Mesh Move then
3:     if (f_predicted < 25) and (d_p is the lowest LoD) then
4:         Break
5:     else
6:         if d_p is not the lowest LoD then
7:             Choose the suitable lower LoD within current CPU resource constraint by predicting model.
8:             if Can not find such kind of d_i then
9:                 Break
10:            end if
11:        end if
12:    end if
13:    if (f_predicted > 25) then
14:        Do CPU scheduling using our CPU scheduling policy to maintain rendering speed almost 25 FPS
15:    end if
16:    if (f_predicted < 25 in some point) then
17:        Increase allocated CPU resource to the maximum available CPU resource
18:        if still f_predicted < 25 then
19:            choose the suitable lower LoD within current CPU resource constraint by predicting model.
20:            choose PoI to render until CPU resource is enough to maintain frame rate of 25
21:        end if
22:    end if
23: end if
24: Choose the LoD nearest to PoI when it reaches the destination
```

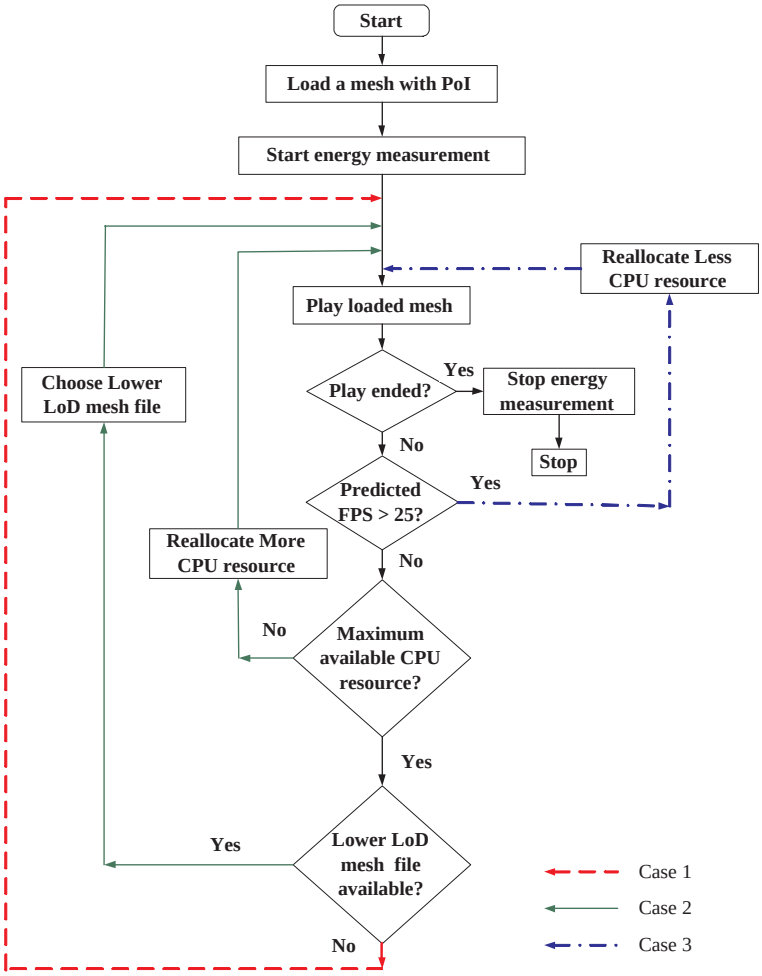


Figure 6.10: Algorithm Flow Chat

typical behavior of real applications. We ran a three sets of experiments using the bunny mesh animated along a sample path in the museum scene, applying three levels of adaptations:

- **Simple (No LoD switching, no CPU scheduling:** The bunny model is rendered at the highest LoD all the time. No LoD changes are made throughout the application's running time and no dynamic CPU scheduling for energy conservation is done. The measured performance of this level of adaptation provides a baseline for establishing how much performance improves with our adaptations.

- **LoD Selection (LoD switching, no CPU scheduling):** The bunny model is rendered, switching mesh LoD as necessary either to react to significant frame rate deviations from 25 FPS, or to react to significant deviations in mesh appearance from acceptable

visual realism (PoI). However, no dynamic CPU scheduling for energy conservation is employed in this case.

- **Our Complete Optimization (LoD selection with CPU scheduling):** LoD Selection is done to satisfy achieve a frame rate of 25 FPS and also to satisfy the visual realism constraint. Additionally, the CPU scheduling policy described in Section 6.4 is also applied. Essentially, this is our complete algorithm to balance visual realism, frame rate and energy conservation.

We now present more details about our experiments. First, we generated a series of mesh LoDs and found the LoD corresponding to the PoI of each mesh. Figure 6.11 shows the PoI curves of bunny model with five LoDs in Laptop. LoD1 is the coarsest one and LoD5 is the finest one. From the figure, we see that LoD2 is the PoI (knee point). Above LoD2, there no perceivable difference as mesh LoD increases. So to satisfy our visual realism constraint, we initially choose to render LoD2.

Next, we calibrated our workload predictor for the bunny mesh. The rendering times for three different LoDs were measured. These measured values were then used to generate a line of best fit that predicts rendering time with error rates of less than than 5% as described in section 6.3.

Our goal was to minimize energy consumption of the CPU excluding peripherals and other system components. Thus, to track how well our algorithm worked, we needed to

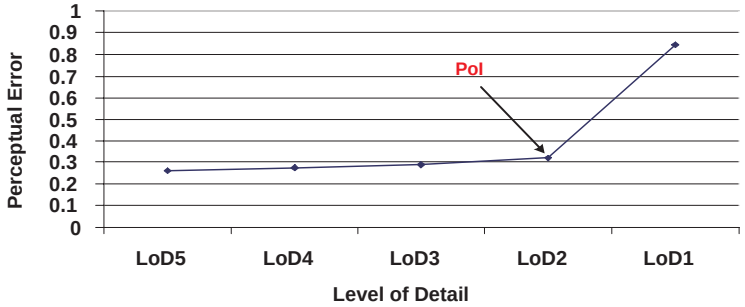


Figure 6.11: PoI for Bunny Model

measure the energy consumption of the CPU alone. Measuring the exact energy consumption of the CPU alone is a fairly hard problem. We use a subtractive technique for estimating CPU energy consumption. First, we measured the power consumption of the entire laptop while running our test application. We then measured the base power consumption of the laptop while running just the operating system in idle mode. Finally, we subtracted this base idle power from measured application energy values. In our experiment, the base power consumed by the laptop in idle mode is 7.19W.

6.6.2 Discussion

During our experiments, we set 20 check points along the animation path of the mesh. Figure 6.12 is a plot of measured frame rates at these check points along the test path with different algorithms tested. Three different plots are used to compare the a) Simple rendering; b) LoD selection and c) optimization algorithms described in section 6.6.

In the experiments called *simple*, the mesh is always rendered at the highest LoD. In such a case, the rendering speed is low, as figure 6.12.a shown. The straight dashed line is the target minimum frame rate of 25 FPS. Without appropriate LoD selection in the simple experiment, the target frame rate of 25 FPS cannot be achieved.

In the expriments called *LoD selection* algorithm, LoD selection is performed but no CPU scheduling is done to conserve energy. In this case, the mesh does not show visual artifacts due to LoD reduction and the application frame rate is always above 25 FPS. However, since no CPU scheduling is done, 100% of all available CPU cycles are always alloted (C_{max}) to the application, and at many points during the application's lifetime, the LoD selected can be rendered much faster than (overshoots) 25 FPS. Figure 6.12.b shows an example. At frame 30 and frame 150, since the frame rate drops, we choose the lower LoD to render and the frame rate goes up. However, this lower LoD will show some visual artifacts since it is below the PoI LoD. At frame 50 and frame 120, since the available CPU resource is enough maintain a frame rate greater than 25 FPS, we choose render the PoI LoD since visually there is little

noticeable difference between PoI and original LoD.

In contrast, in addition to performing LoD selection, *our complete optimization algorithm* reduces alloted CPU resources when the frame rate is far above 25 FPS to save the energy. As a result, the frame rate generated using our optimization algorithm is much more uniform with less fluctuations, as shown in figure 6.12.c. As in the LoD selection algorithm, at frame 30 and frame 150, the frame rate drops. Our complete optimization algorithm first tries to increase alloted CPU timeslices while using the PoI LoD. Since the frame rate continues to drop, the optimization algorithm selects a lower LoD and runs the CPU scheduler algorithm, which reduces the CPU resources alloted to 40% of the maximum available. An application frame rate of 25 FPS is maintained while energy is saved.

Figure 6.13 shows screenshots of our test applications. In figure 6.13.a, the simple algorithm is used with the bunny at its original LoD. The achieved frame rate is only 4.43 FPS. In figure 6.13.b, our complete optimization algorithm is used with the bunny at the PoI LoD. Visually, there is no noticeable difference between the original and PoI LoD. However, the PoI LoD can be rendered at up to 27.01 FPS. In figure 6.13.c, since the target frame rate can not maintained even when all available CPU resource are allocated to the application, a lower LoD is chosen. This LoD is lower than the PoI and introduces some visual artifacts. However, the target frame rate of 25 FPS is maintained. Figure 6.14 shows screenshot of our test applications on a PDA.

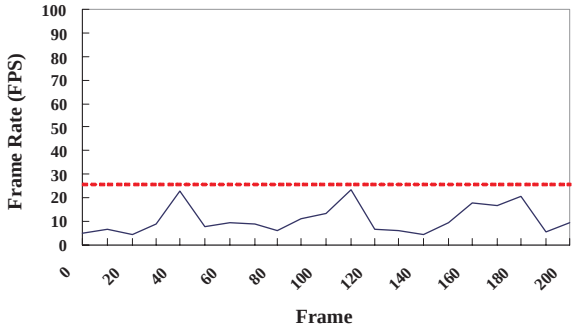
Algorithm	Before(mwh)	After(mwh)	Saved
Simple	9690	9690	0.00%
LoD Selection	9690	7035	27.4%
Our Optimizations	**9690**	**3653**	**62.3**%

Table 6.2: Energy savings

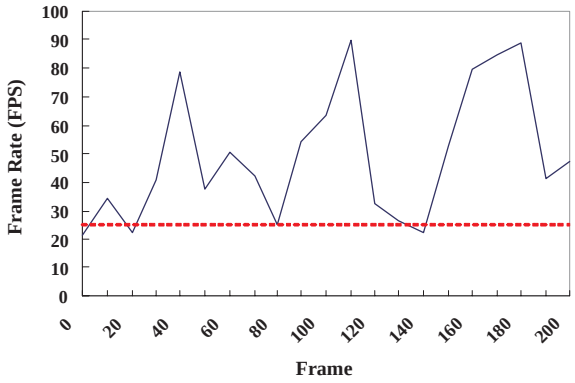
Table 6.2 summarizes the energy saved before and after employing the simple, LoD selection and optimization algorithms. The LoD selection algorithm saves 27.4% of the energy, while our complete Optimization algorithm saves around 62.3% of the energy consumption.

6.7 Chapter Summary

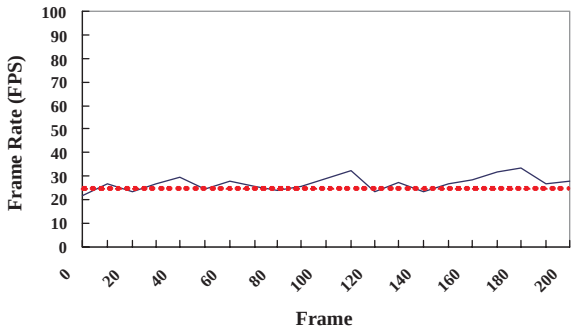
This chapter presents our heuristic to balance energy consumption, rendering speed and image quality. In summary, our integrated EARR (Energy-efficient Adaptive Real-time Rendering) heuristic minimizes energy consumption by i) selecting the lowest LoD that yields acceptable visual realism, ii) scheduling just enough CPU timeslices to maintain real-time frame rates (25 FPS). EARR also switches scene LoD to compensate for workload changes caused by animation, lighting, user interactivity and other factors outside our control. To the best of our knowledge, this is the first work to use CPU scheduling to save energy in mobile graphics. Our results on animated test scenes show that CPU scheduling reduced energy consumption by up to 60% while maintaining real time frame rates and acceptable image realism.



a) Simple

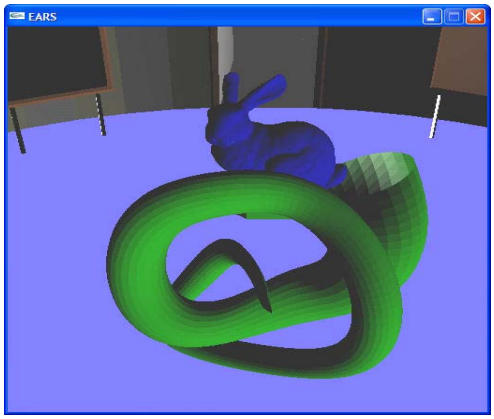


b) LoD Selection

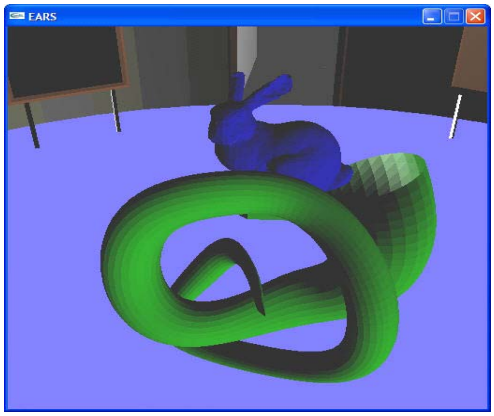


c) Optimization

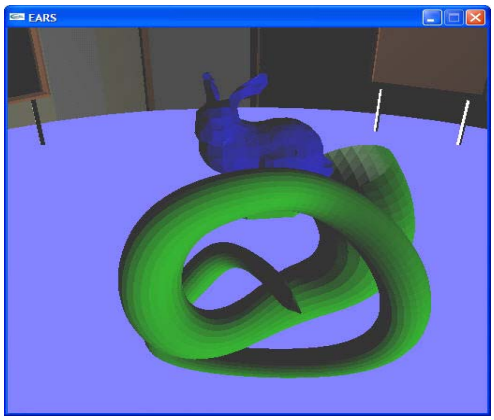
Figure 6.12: Frame Rates at check points along animation path



a) Simple Algorithm (4.43 FPS)



b) Optimization algorithm (27.01 FPS)



c) Optimization algorithm (40.77 FPS)

Figure 6.13: Screenshots on laptop using a) simple algorithm with bunny at original LoD; b) Our Optimization algorithm with bunny at the PoI LOD, and c) optimization algorithm with bunny at LoD lower than PoI.

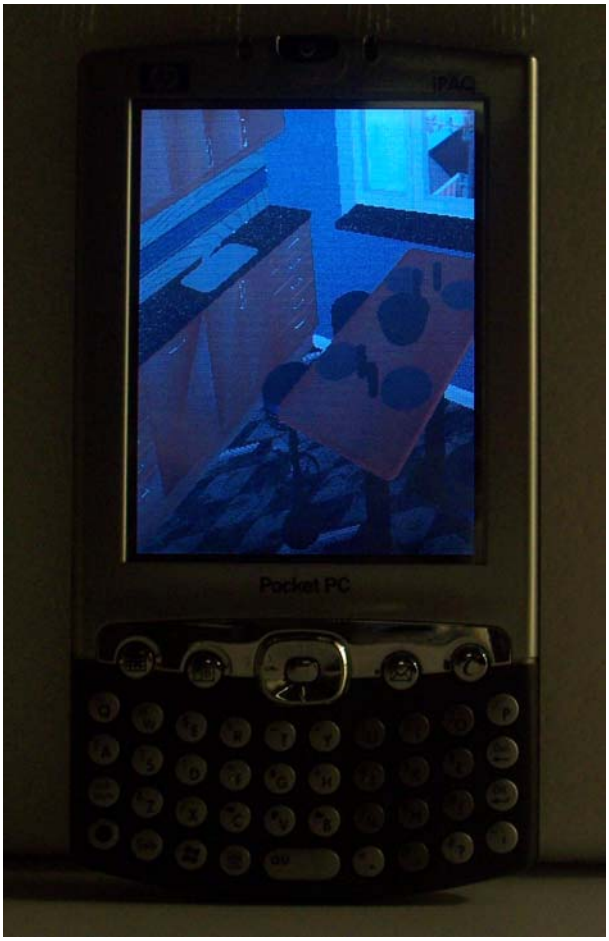


Figure 6.14: Screenshot on a HP iPaq Pocket PC

Chapter 7

Energy-efficient 3D Streaming

This chapter describes a wavelets-based multiresolution mesh streaming technique in UbiWave that utilizes PoI perceptual error metric 4, unequal error protection coding scheme 5 and energy-efficient adaptive real-time rendering heuristic 6. This work is being submitted to [2].

7.1 Overview

In this section, we will outline our proposed technique for 3D streaming in UbiWave.

Normally 3D objects are stored and maintained by either a central or distributed servers. A client sends requests to the server for model retrieval, and the requested models are transmitted accordingly by available communication channels from the server to the client. This scenario is typical when using wireless PDA or cell phone as a tool for Internet access. Since the storage of these mobile device tends to be very limited so that it is difficult to store a lot of 3D data locally. And the size of high resolution 3D data causes long download times in low bandwidth wireless channels, making it is difficult to maintain real-time rendering speeds. UbiWave uses wavelets as the uniform representation for 3D content, which forms the basis to 3D streaming from servers to clients, making rendering 3D data without a complete download.

Figure 7.1 depicts the proposed mesh streaming technique in UbiWave.

To maintain real-time rendering of 3D graphics model, UbiWave decompose 3D meshes into a base mesh and a coefficient tree that is stored in the server so that only base mesh and a certain level of coefficient in coefficient tree need to be encoded and transmitted. Once a mobile device successfully establishes a connection to a server, the parameters of mobile device and network , such as the resources of mobile device and network conditions will be sent to the server. The server determines the PoI of 3D meshes and sends back the data accordingly. The received data is stored at the mobile device side for rendering. Mobile devices use energy-efficient adaptive real-time rendering heuristic to guide rendering so that real-time rendering speed is maintained with minimum energy consumption, while not degrading image quality on mobile devices. Although mobile device has enough resources to maintain real-time rendering speeds, if the network condition is poor, the mobile device still need to wait to receive the whole data so that the real-time rendering speed can not be maintained. A Level of Detail selection algorithm in the server is needed to avoid wasted reception energy consumption in mobile device.

7.2 3D Streaming in UbiWave

From the mobile devices' perspective, the most important qualities of a mesh streaming technique are battery energy, rendering speed and visual quality. our EARR heuristic in the mobile device will balance these three factors.

From the server's perspective, it is preferable if the LoD that is just adequate for each type of mobile device is sent through wireless network with as little data lost as possible. Mesh streaming has two stages: the selection of LoD of the meshes (as determined by the specifications of mobile device) and the efficient transmission of selected data. The first stage involves our PoI perceptual error metrics, while the second stage involves an optimal transmission strategy, unequal error protection coding scheme.

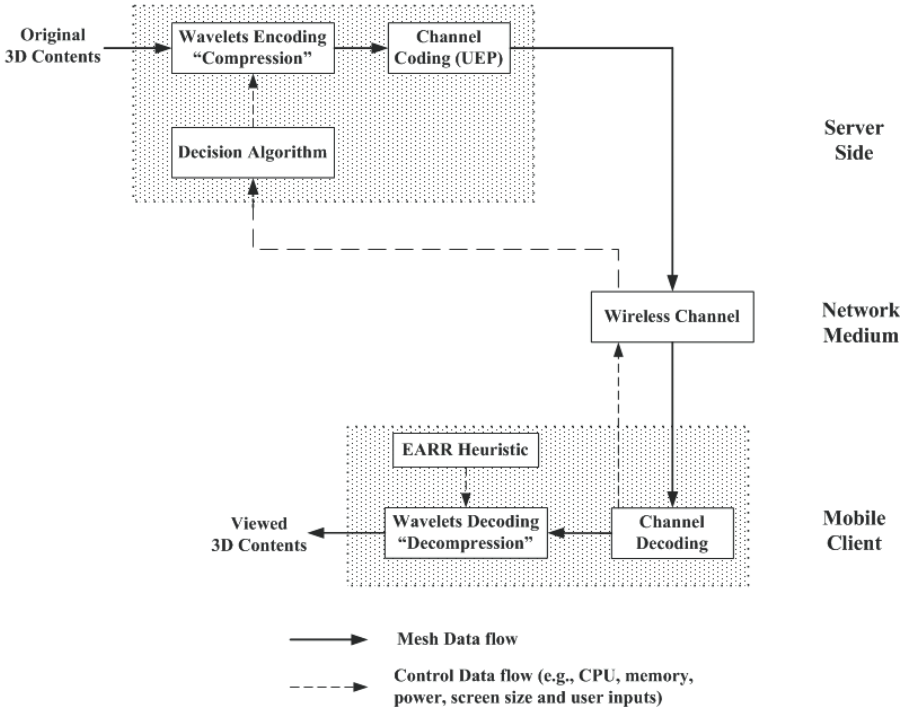


Figure 7.1: The proposed mesh streaming technique in UbiWave

The proposed mesh streaming in UbiWave consists of the following three steps which described in the following subsection.

7.2.1 Stream Generation

Streaming data of the model were generated offline in a preprocess stage in the server. The streaming data has two features: (1) the availability of finer granularity, which can provide a more flexible data organizing structure during transmission; and (2) the remarkable reduction of the size of the base mesh and refinement data, which can dramatically decrease the transmission time.

Figure 7.2 shows the transmission time for meshes and coefficient files of bunny model with wireless network speed 11Mbps. It can be observed that the time required to transfer coefficient files is significantly less than the transfer time for the actual mesh. This demonstrates that the use of wavelets to encode meshes can save transmission time and network bandwidth.

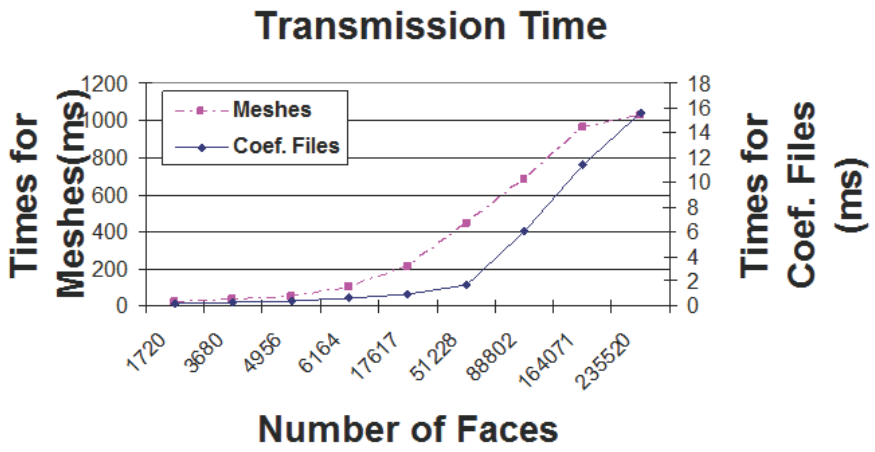


Figure 7.2: Transmission Time of Bunny model

Figure 7.3 shows the transmission time for images and coefficient files with wireless network speed 11Mbps. Again, it can be observed that the time required to transfer coefficient files is significant less than the transfer time for the actual images. This demonstrates that the use of wavelets to encode images can save transmission time and network bandwidth.

In UbiWave, wavelets transform decomposes the 3D mesh into base mesh (structural data) and coefficients (geometric data). The coefficients (geometric data) are then decomposed into different levels. Each level of coefficients is related to one level of detail mesh. After preprocessing, the 3D data is stored as structural and geometric levels. Note that the pre-processing needs to be performed only once offline for a given 3D data stream. All 3D content are initially stored at a server, and mobile devices obtain them through a streaming process from the server.

7.2.2 Server Decision Algorithm

As mentioned in section 7.1,mesh streaming has two stages: the selection of LoD of the meshes and the efficient transmission of selected data. In this section, we describe these two stages in server decision algorithm in detail.

(1) Level-of-Detail Selection

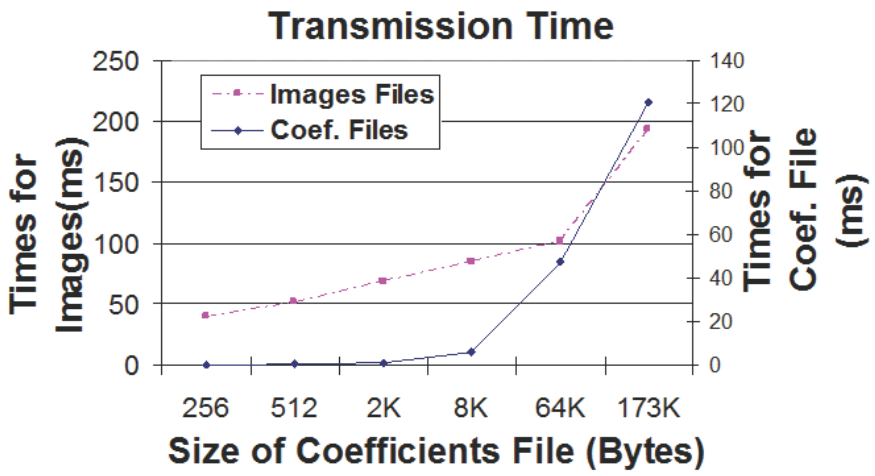


Figure 7.3: Transmission Time for images

The Level of Detail of each mesh is determined on the basis of the three factors: (1) Human perceptual error in different mobile devices; (2) Configurations of mobile device, such as display size, CPU resource and battery energy; (3) Network conditions, such as bandwidth and package loss rates.

The flow of data in our system is illustrated in figure 7.1. The Level-of-Detail Selection algorithm can be illustrated as three basic steps, as shown in figure 7.4:

1. Once a mobile device establishes a connection to a server, the server will immediately transmit the base mesh to the mobile device and the configurations of the mobile device are periodically sent to the server. The transmitted configurations information from mobile device is used to determine the level of coefficients to be streamed, which includes the resolution of displayer of mobile device and the current available resources.

2. After the server receives the configurations information of a mobile device, it can calculate the PoI for the mobile device and record the LoD, d_{sent}, which has been sent to the mobile device, starting with base mesh. This information is organized in the following format in the server: [Device ID][Model ID][PoI][Level of Detail]

3. The server monitor the channel information when the configuration information of mo-

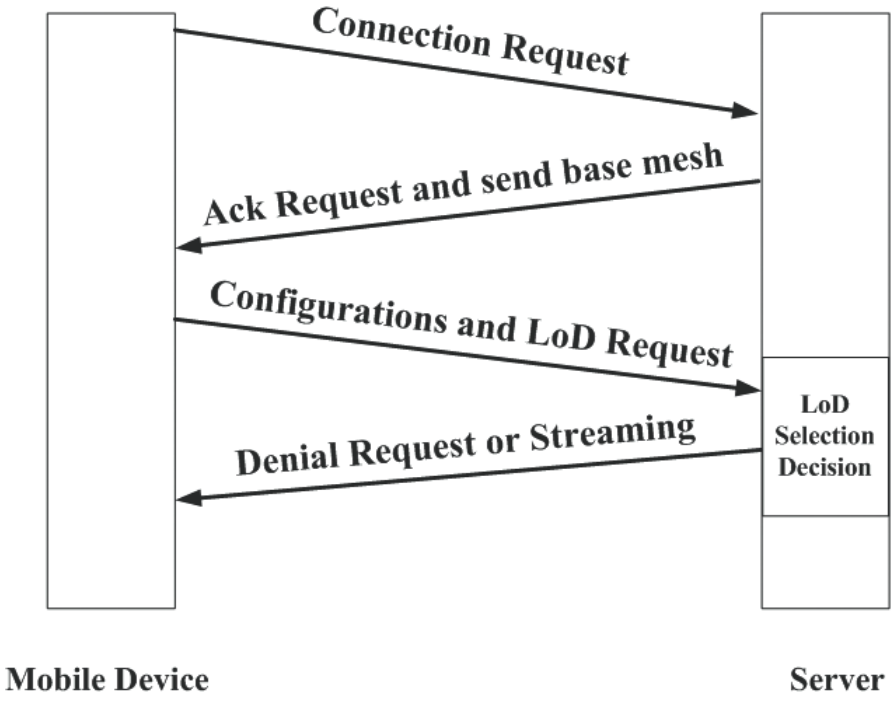


Figure 7.4: The Communication Process of Level of Detail Selection algorithm

bile device is received by the server. In mobile devices, it predicts the real-time rendering time, rt_i for possible acceptable LoD i, d_i within the current available resources and sends them to the server. The server calculates the transmission time, $t_{trans.}$ for the mesh data of LoD i, d_i. Then we have three cases:

(1) if d_i is lower than d_{sent}, there is no need to stream it.

(2) if d_i is higher than d_{sent}, but the transmission time, $t_{trans.}$ is larger than the real-time rendering time, rt_i, there is pointless to stream it to the mobile device, since mesh data can not be transmitted to the mobile device on time.

(3) if d_i is higher than d_{sent}, and the transmission time, $t_{trans.}$ is smaller than the real-time rendering time, rt_i, the difference will be streamed to the mobile device.

Figure 7.5 is the flow chart of the Level of Detail selection algorithm.

Note that there is no significant visual error for the Level of Detail above the PoI for the specific model and mobile device. So the highest Level of Detail for a model sent to a specific

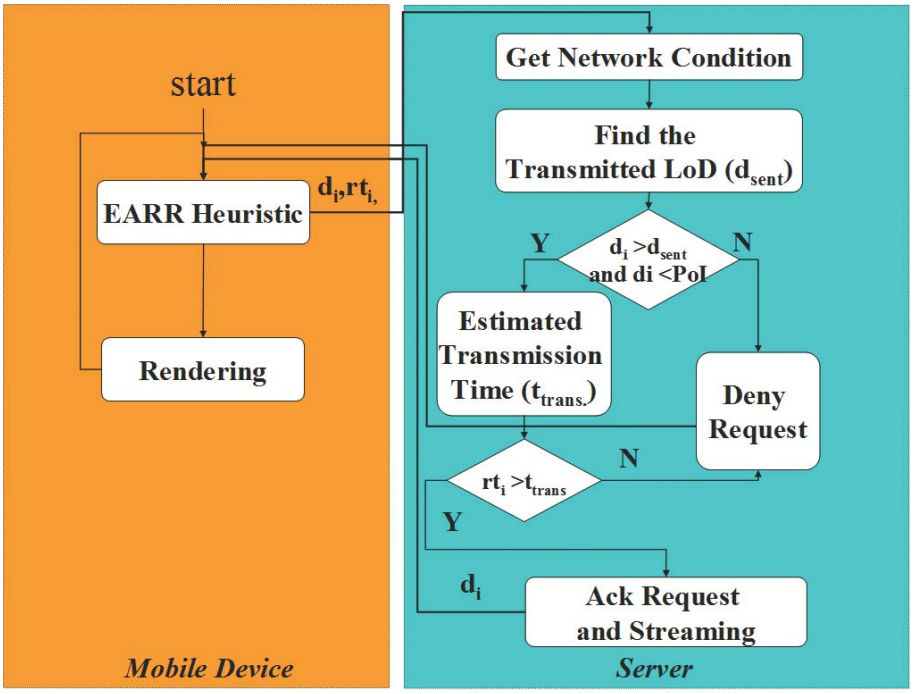


Figure 7.5: Flow chart of Level of Detail selection algorithm

mobile device is its PoI.

(2) Efficient Data Transmission

We then use the UEP coding scheme to protect data from being corrupted. First, to guarantee the same connectivity of the decoded mesh as the original mesh, we assign more FEC bits on the base mesh. Next, we consider the importance of the coefficients and assign more FEC bits. Since the loss of the coefficient data only affects the quality of the decoded mesh and will not make it crash, we can assign less FEC bits to them. The detailed discuss about UEP coding scheme is in Chapter 5.

7.2.3 Rendering

Once the connection between mobile device and server is established, the server sends the base mesh to the mobile device. Normally the size of base mesh is small enough for most of the mobile device to render. Then the mobile device predicts the possible acceptable LoD using EARR heuristic discussed in chapter 6 and sends the request to the server. If

the requested LoD satisfies the Level of detail selector requirement, the server streams the additional requested coefficients to the mobile device. Since the available resources may change, when the coefficients arrive the mobile device, the mobile device decides whether to render the new LoD or not based on EARR heuristic.

7.3 Results

The bunny model in the kitchen scene is transmitted over low bandwidth, high error rate wireless channel. We compare the performance without mesh streaming decision technique in terms of rendering speed, image quality and energy consumption. Table 7.1 summarizes rendering speed, image quality and energy consumption in both wireless network channel.

	Without our streaming technique	With our streaming technique
Rendering Speed	31fps	27fps
$Image\ Quality(faces)$	7328	7328
Energy Consumption	**16432 mwh**	**10387 mwh**

Table 7.1: Performance Comparison

From this tables, we know that the rendering speed and image quality are almost the same since our EARR heuristic will maintain the real-time rendering speed around 25fps. There are two advantages of our approach:

1. **Streaming Latency:** With our streaming technique, the server will not send the requested data to mobile device and deny the request when the network conditions are not good, although the mobile device has enough resources to render the 3D models. The mobile device do not need to wait for the data sending from the server. Our streaming technique achieves a better streaming latency.

2. **Energy Efficiency:** Without our proposed mesh streaming, the server keep sending mesh data with higher LoD. Because of the additional transmission time the mobile

device can not maintain real-time rendering speed, our EARR heuristic will lower the LoD rendered in mobile device automatically. The received data from server with higher LoD is not useful in mobile device. Therefore the reception energy is wasted. With our proposed mesh streaming technique, if the transmission time is longer than the real-time rendering time, the server will deny the request from the mobile device, and the mobile device will not waste energy receiving data with higher LoD from the server. From the above table, the energy could be saved by 36.8%.

7.4 Chapter Summary

This chapter presents our wavelets-based energy-efficient streaming technique in UbiWave. Our streaming technique includes three steps: 1) Streaming Generation; 2) Server Decision Algorithm and 3) Rendering in mobile devices. Our streaming technique is useful in wireless network with low bandwidth. It reduces the wasted energy for data transmission. Our experiment results show that Level-of-Detail selection in our steaming technique achieves better streaming latency and saves energy consumption up to 36.8% in low bandwidth wireless networks.

Chapter 8

Future Work

This chapter presents some possible future work that can be extended from this dissertation.

- Even though our Point of Imperceptibility (PoI) error metric works well for meshes, we could make our metric view independent. We propose calculating our PoI metric for each object from multiple view points around the object, and then combines these values. This approach is similar to the image-driven simplification approach of Lindstrom and Turk [72]. We intend to investigate the behavior of the average, minimum and maximum of the PoI calculated from these different views.

- Texture is another factor which affects on human perception. The future work should consider texture mapping and how it affects on human perception and make the Point of Imperceptibility (PoI) more accurate.

- We analysis the performance of Unequal Error Protection (UEP) scheme and compare the performance with Equal Error Protection (EEP) and None Error Protection (NEP). Comparing the performance of the proposed UEP shceme when applied to wavelets-encoded meshes to UEP on Compressed Progressive Meshes could also be considered.

- We also could investigate the benefits of zero-tree coding. In zero-tree coding, coefficients with values greater than some appropriate threshold value are kept and low-

valued coefficients (little information) are replaced by zero.

- Currently, we only did simulations on simple G-E two state Markovian Channel Model. A more complicated channel model, like noise channel model could be applied in the simulations.

- Improve energy saving by integrating Dynamic Voltage Scaling (DVS) and Dynamic Frequency Scaling (DFS). DVS and DFS are popular to be used in graphics hardware. We expect our heuristic will yield further savings after integrating DVS or DFS.

- Improve PoI by integrating eye's gaze pattern. Eye's gaze pattern is another important factor affecting human visual perception. With cues about the eye's gaze pattern, we can increase the LoD of objects that user focuses on while reducing the LoD of objects outside of the focus area. In this way, even more rendering costs can be saved..

- Accurately measuring CPU energy usage. We currently estimate CPU energy usage using a subtractive technique described in section 6.6, which can be improved in accuracy. We plan to develop more accurate methods to more accurately measure CPU energy consumption on mobile devices.

Chapter 9

Conclusions

This dissertation presents an UbiWave, an end-to-end framework using wavelets solution for improving mobile graphics application performance in mobile device by balancing energy consumption, rendering speed and image quality. The solution includes four parts: 1) a perceptual error metric to guide mobile graphics scenes at the lowest LoD at which users do not perceive distortion due to simplification (PoI); 2) a novel Forward Error Correction scheme based on Unequal Error Protection (UEP); 3) an Energy-efficient Adaptive Real-time Rendering (EARR) Heuristic to balance energy consumption, rendering speed and image quality and 4) an Energy-efficient 3D streaming technique. With PoI, UEP, EARR and streaming technique, the performance of mobile graphics applications in wireless networks can be significantly improved in terms of energy consumption, rendering speed and image quality. This chapter summarizes the major contributions and draws conclusions from the dissertation research.

Mobile displays have a wide range of resolutions that affect the scene Level-of-Detail (LoD) that users can perceive: smaller displays show less detail, therefore lower resolution meshes and textures are acceptable. Mobile devices frequently have limited battery energy, low memory and disk space. To minimize wasting limited system resources, in Chapter 4, we try to render mobile graphics scenes at the lowest LoD at which users do not perceive

distortion due to simplification. We call this LoD the Point of Imperceptibility (PoI). Increasing the mesh or texture resolution beyond the PoI wastes valuable system resources without increasing perceivable visual realism. The PoI depends on several factors including screen size, scene geometry and lighting levels. We propose a perceptual metric that can easily be evaluated to identify the LoD corresponding to a target mobile display's PoI and accounts for object geometry, lighting and shading. Previous work either focussed on the deviation of a simplified mesh's surface from the original high-resolution mesh or were perceptual metrics that needed to be experimentally calibrated in order to parameterize the Contrast Sensitivity Function (CSF). Neither approach directly computes changes in the PoI due to target screen resolution. Our perceptual metric generates a screen-dependent Pareto distribution with a knee point that corresponds to the PoI. We employ wavelets for simplification, which gives direct access to the mesh undulation frequency that we then use to parametrize the CSF curve. A comprehensive user study has been performed to validate our results.

To speed up large mesh transmission over low-bandwidth wireless links, we use a wavelets-based technique that aggressively compresses large meshes and enables progressive (piecewise) transmission. Using wavelets, a server only needs to send the full connectivity information of a small base mesh along with wavelets coefficients that refine it, saving memory and bandwidth. To mitigate packet losses caused by high wireless error rates, in chapter 5, we propose a novel Forward Error Correction (FEC) scheme based on Unequal Error Protection (UEP). UEP adds more error correction bits to regions of the mesh that have more details. Previous work applied UEP to CPM. Our work uses UEP to make wavelets-encoded meshes more resilient to wireless errors. Experimental results shows that our proposed UEP scheme is more error-resilient than No Error Protection (NEP) and Equal Error Protection (EEP) as the packet loss rate increases. Challenges for future research are also discussed. Our scheme can be integrated into future mobile devices and shall be useful in application areas such as military simulators on mobile devices.

To use the benefit of PoI provided by Chapter 4, Chapter 6 presents an energy-efficient

adaptive real-time rendering heuristic to balance energy consumption, rendering speed and image quality. Graphics rendering on mobile devices is severely restricted by available battery energy. In real-time graphics applications continual changes in user interactivity, the LoD, visibility and distance of scene objects, complexity of lighting and animation, and many other factors cause its frame rate to fluctuate. Such frame rate spikes waste precious battery energy. We use a PoI error metric to accurately pick the lowest acceptable mesh resolution considering the screen size. Our proposed heuristic uses a workload predictor to adaptively predict frame rendering times and a dynamic CPU scheduler to save energy in a mobile 3D application and render the scene at a target rate of 25 FPS.

To improve the real-time rendering performance in wireless network, a 3D streaming technique is presented in Chapter 7. Since the wireless network has low bandwidth, some of the mesh data can not be transmitted on time, although the mobile device has enough resources to render graphics models, thus waste the reception energy of mobile device. We proposed an energy-efficient 3D streaming in UbiWave. It estimates the transmission time prior to transmission and decides whether it could be transmitted to the mobile device on time or not. If not, the server denies the requests from mobile device. With this technique, we achieve a better streaming latency and save the reception energy consumption of mobile device.

Based on the summary of this dissertation, the following conclusion can be drawn:

- Our Point of Imperceptibility (PoI) error metric accurately picks the lowest acceptable mesh (or image) resolution based on the target mobile device's screen size, which is validated by our user studies. By using our perceptual metric, it can save up to 61% of the total battery energy.

- Our Unequal Error Protect Scheme allocates more Forward Error Correction (FEC) to the important parts of the decoded mesh. The performance is better than Equal Error Protection and No Error Protection.

- Windows based workload predictor can predictor workload adaptively. The relative errors are usually bounded in 0.2.

- Our integrated Energy-efficient Adaptive Real-time Rendering heuristic reduces energy consumption by up to 60% while maintaining acceptable image quality at real-time frame rate of 25 FPS.

- Energy-efficient 3D streaming enables scalable rendering in mobile device with better streaming latency and reduces energy consumption by up to 36% while maintaining the real-time frame rate.

In conclusion, This dissertation presents an UbiWave, an end-to-end scalable framework using wavelets solution for improving mobile graphics application performance in mobile device by balancing energy consumption, rendering speed and image quality. Using analytical mathematical models, experiments and user studies, this dissertation shows that our 3D mobile graphics solution, consisting of PoI, UEP, EARR and Energy-efficient 3D Streaming technique can effectively improve the 3D mobile graphics application performance in terms of energy consumption, rendering speed and image quality on a variety of mobile devices and wireless network conditions.

Bibliography

Our Peer-Reviewed Submitted Mobile Graphics Papers

[1] F Wu, E Agu and C Lindsay, Unequal Error Protection (UEP) for wavelet-based Wireless Mesh Transmission, Submitted to Percom 2009, Galvaston, TX. March 2009. 53

[2] F Wu, E Agu and C Lindsay, An Energy-efficient 3D Streaming Scheme in Real-Time Mobile Graphics Applications, Submitted to MobileWare 2009, Berlin, Germany, April, 2009. 93

Our Papers on Mobile Graphics Papers

[3] F Wu, E Agu and C Lindsay, Adaptive CPU Scheduling to Conserve Energy in Real-Time Mobile Graphics Applications, ISVC 2008, Las Vegas, NV. December 2008. 67

[4] F Wu, E Agu and C Lindsay, Adaptive CPU Scheduling to Conserve Energy in Real-Time Mobile Graphics Applications, Pacific Graphics 2008, Tokyo, Japan. October 2008. 67

[5] F Wu, E Agu and C Lindsay, Pareto-Based Perceptual Metric for Imperceptible simplification on mobile displays, Technical Report WPI-CS-TR-07-01, Computer Science Department, Worcester Polytechnic Institute, May 2007. 27

[6] F Wu, E Agu and C Lindsay, Pareto-Based Perceptual Metric for Imperceptible simplification on mobile displays, Eurographics 2007, Prague, Czech Republic, September 2007. 27, 69

[7] F Wu, E Agu and M Ward, Multiresolution Graphics on Ubiquitous Displays using Wavelets, International Journal of Virtual Reality, Vol. 5, Number 3, 2006. *(acceptance rate: 12/522)* . 27

[8] F Wu, E Agu and M Ward, UbiWave: Ubiquitous Multiresolution Graphics using Wavelets, In Proc. Of 16th International Conference Artificial Reality and Telexistence(ICAT06), Hangzhou, China, November 2006. *(Best Paper Awarded)* . 27

[9] F Wu and E Agu, Unequal Error Protection for Wavelet-Based Wireless Mesh Transmission, ACM SIGGRAPH, Boston, MA, 2006(Poster). 53

[10] Kutty Banerjee, Fan Wu and Emmanuel Agu, Estimating Mobile Memory Requirements and Rendering Time for Remote Execution of the Graphics Pipeline, in Proc Eurographics 2005. 48

Introductory/General Papers

[11] Boier-Martin I. M., Adaptive Graphics, IEEE Computer Graphics and Applications, 2,1 (Jan-Feb 2003), 6-10. 19

[12] B T Phong, Illumination for Computer Generated Images, Communications of the ACM, Vol. 18(6): 311-317, June 1975. 2

[13] N. Aspert, D. Santa-cruz, and T. Ebrahimi. Mesh:measuring errors between surfaces using the hausdoff distance. *Proc. IEEE Int'l Conf. on Multimedia and Expo*, pages 705–708, 2002. 21, 32

[14] P. Cignini, C. Rocchini, and R. Scopigno. Metro: measuring error on simplified surfaces. *Computer Graphics Forum*, pages 167–174, 1998. 21

[15] H. Hoppe. Progressive meshes. *Proc ACM SIGGRAPH*, pages 99–108, 1996. 25, 31

Wavelets

[16] A Graps, A friendly guide to wavelets, IEEE Computational Science and Engineering, Summer 1995, vol. 2, num. 2, published by the IEEE Computer Society 9

[17] C. Christopoulos, A. Skodras, and T. Ebrahimi. The jpeg2000 still image coding system: an overview. *IEEE Trans. on Consumer Electronics Vol. 46, Issue 4*, pages 1103–1127, 2000. 9, 12

[18] J N Bradley, C M Brislawn, The wavelet/scalar quantization compression standard for digital fingerprint images, in Proc. IEEE International Symposium on Circuits and Systems(ISCAS), 1994. 10

[19] T. D.Derose, M. Lounsbery, and J. Warren. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Trans. on Graphics 16*, pages 34–73, 1997. 10, 11

[20] O. Devillers, D. Levin, and O. Remez. Progressive compression of arbitrary triangular meshes. *IEEE Visualization 99*, pages 67–72, 1999. 31

[21] N. Dyn, D. Levin, and J. A. Gregory. A butterfly subdivision scheme for surface interpolation with tension control. *ACM Trans. on Graphics*, pages 160–169, 1990. 11

[22] A. Webster et al. An objective video quality assessment system based on human perception. *Proc. SPIE Human Vision, Visual Processing, and Digital Display TV*, pages 15–26, 1993. 22

[23] C. J. van Den, B. Lambrecht and O. Verscheure, Perceptual quality measure using a spatio-temporal model of the human visual system., in Proc. SPIE, p. 450-461, 1996. 22

[24] A. Finklestein. Multiresolution application in computer graphics curves, images and video. *PhD diss., U. Washington*, 1996. 30

[25] A. Khoakovsky, P. Schroder, and W. Sweldens. Progressive geometry compression. *Proc. of SIGGRAPH 2000*, pages 271–278, 2000. 25

[26] P. G. Lemarie and Y. Meyer. Ondelettes et bases hilbertiennes. *Rev. Mat. Iberoamericana, vol 2*, pages 1–18, 1986. 12

[27] C. Loop. Smooth subdivision surfaces based on triangles. *Master's thesis. Dept. of Math., Univ. Utah*, 1987. 11

[28] M. Lounsbery. Multiresolution analysis for surfaces of arbitrary topological type. *PhD diss., U. Washington*, 1994. 9, 10, 11, 30

[29] Y. Pan, I. Cheng, and A. Basu. Quality metric for approximating subjective evaluation of 3d objects. *IEEE Trans. on Multimedia, Vol. 7, No. 2*, pages 269–279, 2005. 22

[30] L. A. F. Park, K. Ramamohanarao, and M. Palaniswami. A novel document retrieval method using the discrete wavelet transform. *Trans. on Graphics (TOG), Vol. 23 Issue 3*, 2005. 10

[31] W. Sweldens. The lifting scheme: A custom-design construction of biothogonal wavelets. *Applied and Computational Harmonic Analysis,vol.3*, pages 186–200, 1996. 11, 12

[32] N. Tack, G. Lafruit, F. Catthoor, and R. Lauwereins. Pareto based optimization of multiresolution geometry for real time rendering. *Proc. ACM Web 3D*, pages 19–27, 2005. 31, 33

[33] S. Valette and R. Prost. Wavelet-based progressive compression scheme for triangle meshes: Wavemesh. *IEEE Trans. Vis. and Computer Graphics, vol. 10, no. 2*, 2003. 12

[34] S. Valette and R. Prost. Multiresolution analysis of irregular surface meshes. *IEEE Trans. Visual. Comput. Graphics 10*, pages 113–122, 2004. 11, 12, 25

[35] S. Winkler. A perceptuall distortion metric fo rdigital color video. *Proc. SPIE Human Vision and Electronic Imaging*, pages 23–29, 1999. 22

[36] P. Schroder and W. Sweldens,Spherical wavelets: Efficiently representing functions on the sphere, in Proc. ACM SIGGRAPH Computer Graphics 29 (1995), no. Annual Conference Series, 1995, p 161-172. 9, 13

[37] P. Lalonde, Representations and uses of light distribution functions, Ph.D thesis, The University of British Columbia, 1997. 13

[38] P. Clarberg, W. Jarosz, T. Akenine-Moller and H. W. Jensen, Wavelet Importance Sampling: Efficiently Evaluating Products of Complex Functions, in Proc. of ACM SIGGRAPH 2005. p. 1166-1175. 10

Captured Content Repositories

[39] Reflectance Data, Cornell University Program of Computer Graphics, http://www.graphics.cornell.edu/online/measurements/reflectance/index.html July 2006 3, 5

[40] Paul Debevec's Light Probe Image Gallery, http://www.debevec.org/Probes/ July 2006 3, 5

[41] Bidirectional Texture Function Database Bonn, http://btf.cs.uni-bonn.de/index.html July 2006 3, 5

[42] Stanford 3D Scanning Repository, http://graphics.stanford.edu/data/3Dscanrep/, July 2006 3, 5

[43] Georgia Institute of Technology Large Geometric Models Archive, http://www-static.cc.gatech.edu/projects/large_models/ July 2006 3, 5

[44] M Levoy, The Digital Michelangelo Project, in Proc. Second International Conference on 3D Digital Imaging and Modeling, Oct. 1999. p. 2-11. 3

[45] The Digital Michelangelo Project Archive, http://graphics.stanford.edu/data/mich/ 3

Related Graphics Systems

[46] Macintyre B and Feiner S, A Distributed 3D Library, in Proc. 25th Annual Conference on Computer Graphics and Interactive Techniques (1998), ACM Press, pp. 361-370. 20

[47] Martin I. M., ARTE: An Adaptive Rendering and Transmission Environment for 3D Graphics, in Proc. 8th ACM International Conference on Multimedia (2000), ACM Press, pp. 413-415. 19

[48] Yang C-K, Chiueh T, An Integrated Pipeline of Decompression, Simplification and Rendering for Irregular Volume Data 20

[49] Lamberti F, Zunino C, Sanna A, Fiume A and Maniezzo M, An Accelerated Remote Graphics Architecture for PDAs, in Proc. ACM Web3D 2003, pp. 55-61 20

[50] Zunino C, Lamberti F, Sanna A and Montrucchio B, A Wireless Architecture for Performance Monitoring and Visualization on PDA Devices, in Proc. SCI 02 Proceedings, vol. XV, pp. 143-148 20

[51] Paelke V, Reimann C and Rosenbach W, A Visualization Design Repository for Mobile Devices, in Proc. AFRIGRAPH 2003, pp 57-62. 20

[52] Pandzic I, Facial Animation Framework for the Web and Mobile Platforms, in Proc. ACM Web3D 2002, pp. 27-34 20

[53] Schmalstieg D, The Remote Rendering Pipeline, PhD Dissertation, Vienna University of Technology, 1997 20

Geometric Simplification

[54] Cignoni P, Montani C, and Scopigno R, A Comparison of Mesh Simplification Algorithms, Computers and Graphics 22, 1 (1998), 37-54.

[55] Luebke, D.P, A Developer's Survey of Polygonal Simplification Algorithms. IEEE Computer Graphics and Applications, May/June 2000, pp 24-34 21

[56] Cignoni P, Montani C and Scopigno R, Metro: Measuring Error on Simplified Surfaces, Computer Graphics Forum, 1998, pp. 167-174 21

[57] Garland M, QSlim 2.0, University of Illinois at Urbana-Champaign, UIUC Computer Graphics Lab., 1999. 21

[58] Cohen J, Varshney A, D. Manocha Turk G, Weber H, Agarwal P, Brooks F and right W, Simplification Envelopes, in Proc. ACM SIGGRAP 1996. 21

[59] Cohen J, Olano M, Manocha D, Appearance Preserving Simplification, in Proc. ACM SIGGRAPH 1998, pp. 115-122. 20, 21

21

[60] Gotsman C, Gumbold S, Kobbelt L, Simplification and Compression of 3D Meshes, in Tutorials on Multi-Resolution in Geometric Modeling (Munich Summer School Lecture Notes), Iske A, Quak E and Floater M (Eds), Springer-Verlag, 2002 21

[61] Garland M and Heckbert P, Surface Simplification using Quadric Error Metrics, in Proc. ACM SIGGRAPH 1997, pp. 209-216 20, 21

[62] Garland M and Heckbert P, Simplifying Surfaces with Color and Texture using Quadric Error Metrics, in Proc. IEEE Vis. 1998, pp. 264-270. 21

[63] CAMPBELL F. W., ROBSON J. G.: An application of fourier analysis to the visibility of contrast gratings. *Journal of Physiology, 187 (1968)*. 21, 33

[64] Gueziec A, Surface simplification with variable tolerance, in Proc. of Second Annual International Symposium On Medical Robotics and Computer Assisted Surgery. 1995, pp. 132-139. 20, 21

[65] Gueziec A, Locally toleranced surface simplification, in IEEE Transactions on Visualization and Computer Graphics. Vol.5(2), 1999, pp. 168-189 20, 21

[66] Ronfard R and Rossignac J, Full-range approximation of triangulated polyhedra, Computer Graphics Forum. Vol.15(3), 1996, pp. 67-76 20, 21

[67] Bajaj C and Schikore D, Error-bounded reduction of trianges meshes with multivariate date, SPIE. Vol.2656, 1996, pp. 34-45 20, 21

[68] N. Williams, D. Luebke, J. D. Cohen, M. Kelley, and B. Schubert. Perceptually guided simplification of lit, textured meshes. *Proc. Symp. Interactive 3D Graphics*, pages 113–121, 2003. 21, 22

[69] WANG X., SILVA F., HEIDEMANN J.: Demo abstract: Follow-me application—active visitor guidance system. In *Proceedings of the 2nd ACM SenSys Conference* (Baltimore, Maryland, USA, 2004), ACM. 29

[70] M. Reddy. Perceptrally modulated level of detail for virtual environments. *PhD diss., Univ. Edinburgh*, 1997. 21, 22, 32

[71] M. Reddy. Perceptrally optimized 3D graphics. *IEEE Comp. Graph. and Appl.*, 21(5) 2001, pp. 68-75 21

[72] P. Lindstrom and G. Turk. Image-driven simplification. *ACM Trans Graphics 19, 3*, pages 204–241, 2000. 19, 21, 103

[73] D. Luebke and B. Hallen. Perceptually driven simplification. for interactive rendering. *Proc. Eurographics Rendering Workshop*, pages 7–18, 2001. 21, 22, 29, 32

[74] J. Mannos and D. Sakrison. The Effects of a Visual Fidelity Criterion on the Encoding of Images. *IEEE Transactions on Information Theory*, pages 525–535, Vol. 20, No 4, 1974. 38

Image-based Simplification

[75] Decoret X, Durand F, Sillion F and Dorsey J, Billboard Clouds, in Technical Report RR-4485, 2002 19

[76] Decoret X, Durand F, Sillion F and Dorsey J, Billboard Clouds for Extreme Model Simplification, in Proc. ACM SIGGRAPH 2003 19

[77] Oliveira M and Bishop G, Image-based Objects, in Proc. Symposium Interactive 3D Graphics (SI3D) 1999, pp. 191-198. 19

Efficient Transmission

[78] Deering M, Geometry compression, *Proc. SIGGRAPH'95*, pp. PP. 13–20, Aug. 1995. 53

[79] Rossignac J, "Edgebreaker: connectivity compression for triangle meshes," *IEEE Trans. Vis. and comput. Graphics, Vol. 5, no. 1*, pp. pp 47–61, Jan. 1999. 53

[80] Chow M, "Optimized geometry compression for real-time rendering," *Proc. IEEE Visualization'97, Phoenix, AZ*, pp. pp 347–354, Oct. 1997. 53

[81] G. Al-Regib, Y. Altunbasak, and J. Rossignac, "An unequal error protection method for progressively transmitted 3-d models," *IEEE Transactions on Multimedia, vol. 7, no. 4*, pp. 766–776, 2005. 23, 54

[82] A. E. Mohr, E. A. Riskin, and R. E. Ladner, "Unequal loss protection: Graceful degradation of image quality over packet erasure channels through forward error correction," *IEEE Journal on Selected Areas in Communications, vol. 18, no. 6*, pp. 819–828, 2000. 22

[83] S. Bischoff and L. Kobbelt, "Toward robust broadcasting of geometry data," *Comput. and Graph., vol.26, no.5*, pp.665–675, 2002. 22

[84] C.L. Bajaj, S. Cutchin, V. Pascucci and G. Zhuang, "Error Resilient Transmission of Compressed VRML," TICAM, The Univ. Texas at Austin, Austin, TX, Tech. Rep., 1998. 23

[85] Z. Yan, S. Kumar and C. Kuo, "Error resilient coding of 3-D graphic models via adaptive mesh segmentation," *IEEE Trans. Circ. Syst. Video Tech., vol.11, no.7*, pp.860–873, July 2001. 23

[86] R. Pajarola and J. Rossignac, "Compressed pregressive meshes," *IEEE Transactions on Visualization and Computer Graphics, vol. 6, no. 1*, pp. 79–93, 2000. 23

[87] W. J. Schroeder., "Decimation of triangle meshes." *in Proc. ACM SIGGRAPH*, pp. 65–70, 1992. 10, 30, 31

[88] C. Pimentel and I. Blake, "Modeling burst channels using partitioned fritchman's markov models," *IEEE Trans Veh Tech., vol. 47, no. 3*, pp.885–899, 1998. 60

[89] Chen B-Y and Nishita T, Multiresolution Streaming Mesh with Shape Preserving and QoS-like Controlling, in Proc. Web3D 2002, pp. 35-42 19

[90] Chim J et al, Multi-Resolution Model Transmission in Distributed Virtual Environments, in Proc. ACM Symp. Virtual Reality Software and Technology, May 2002, pp. 25-34 19

[91] Southern R, Perkins S, Steyn B, Muller A, Marais P and Blake E, A Stateless Client for Progressive View-Dependent Transmission, In Proc. Web3D 2001 Symposium, pp. 43 - 49 19

[92] Fogel E, Cohen-Or D, Revital I, and Zvi T, A Web Architecture for Progressive Delivery of 3D Content, in Proc. ACM Web3D 2001, pp. 35-41 19

[93] Hoppe H, Progressive Meshes, in Proc. 24th annual conference on Computer Graphics and Interactive Techniques (1997), ACM Press/Addison-Wesley Publishing Co. pp. 189-198 19

[94] Hoppe H, Efficient Implementation of Progressive Meshes, Computers and Graphics, 22.1, 1998, pp. 27-36 19

[95] Hoppe H, View-Dependent Refinement of Progressive Meshes, in Proc. ACM SIGGRAPH 1997, pp. 189-198 19

[96] Deering M, Geometry Compression, in Proc. ACM SIGGRAPH 1995, pp. 13-20 19

[97] Alliez P and Desbraun M, Progressive Compression for Lossless Transmission of Triangle Meshes, SIGGRAPH 2001, pp. 195-202 19

[98] Cohen-Or D, Levin D, and Remez O, Progressive Compression of Arbitrary Triangle Meshes, in Proc. IEEE vis 1999, pp. 67-72. 19

[99] Chow M, Optimized Geometry Compression for Realtime Rendering, in Proc. IEEE Vis 1997, pp. 347-354 19

[100] Gumbold S and Straber W, Real Time Compression of Triangle Mesh Connectivity, in Proc. ACM SIGGRAPH 1998, pp. 133-140. 19

[101] Rossignac J, EdgeBreaker: Connectivity Compression for Triangle Meshes, IEEE Trans. on Vis and Computer Graphics, 1999, pp. 47-61. 19

[102] Taubin G and Rossignac J, Geometric Compression through Topological Surgery, ACM Trans on Graphics, 17.2, 1998, pp. 84-115 19

[103] Touma C and Gotsman C, Triangle Mesh Compression, in Proc. Graphics Interface, pp. 26-34. 19, 53

[104] G. Al-Regib, Y. Altunbasak, and J. Rossignac, An unequal error protection method for progressively compressed 3-D models, IEEE Int. Conf. on Acoustics Speech and Signal Processing, vol. 2, pp. 2041-2044, Orlando, FL, May 2002 23

[105] G. Al Regib, Y. Altunbasak, and R. M. Mersereau, A joint source and channel coding based bit allocation for progressively compressed 3-D mesh models, in IEEE Transactions on Circuits and Systems for Video Technology. 23

[106] G. Al-Regib and Y. Altunbasak. An unequal error protection method for packet loss resilient 3d mesh transmission,.*IEEE INFOCOM*, '02. 23

[107] F A Tobagi, R Binder and B Leiner, Packet Radio and Satellite Networks, IEEE Communications, p. 24-40, Nov. 1984. 22

[108] R W Hamming, Error Detecting and Error Correcting Codes, Bell System Technical Journal, vol. 29 p. 147-160, April 1950. 23, 54

[109] I S Reed and G Solomon, Polynomial Codes over Certain Finite Fields, Journal of the Society for Industrial and Applied Mathematics, June 1960. 23, 54

[110] P C Cosman, J K Rogers, P G Sherwood and K Zeger, Combined Forward Error Control and Packetized Zero Tree Wavelet Encoding for Transmission of Images over Time-Varying Channels, IEEE Trans. Image Processing, 9(6):982-993,June 2000. 23, 54

[111] K Sohn, C Lee, J Ryou and W Jang, Error-resilient Zerotree Wavelet Video Coding, in SPIE Journal of Optical Engineering, Nov 2001, p. 2480-2488. 23, 54

[112] Transmission Control Protocol (TCP), Request For Comment (RFC) 793, Internet Engineering Task Force (IETF), September 1981. 22

[113] IEEE 802.11 Wireless LAN Standard, 2001. 22

Energy-efficient Adaptive Real-time Rendering Heuristic

[114] : Mobile Games Industry Worth US $11.2 Billion by 2010. *http://www.3g.co.uk/PR/May2005/1459.htm* (2005) 2

[115] CHOI K., DANTU K., CHENG. W AND PEDRAM. M: Frame-based dynamic voltage and frequency scaling for a mpeg decoder. *Proc. of the IEEE/ACM CAD'02* (2002), 732–737. 24

[116] YUAN W. AND NAHRSTEDT K.: Practical voltage scaling for mobile multimedia device. *Proc. of ACM MM'04* (2004), 924–931. 24

[117] FLINN J., DE LARA E., SATYANARAYANAN M., WALLACH D., AND ZWAENEPOEL W.: Reducing the energy usage of office applications. *Proc. of Middleware'01* (2001). 24

[118] TAMAI M., SUN T., YASUMOTO K., SHIBATA N., AND ITO M.: Energy-aware video streaming with qos control for portable computing devices. *Proc. of ACM NOSSDAV'04* (2004), 68–73. 24

[119] LIU X., SHENOY P.AND CORNER M.: Chameleon: Application level power management with performance isolation. *Proc. of ACM MM'05* (2005). 24

[120] FUNKHOUSER T. AND SEQUIN C.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. *Proc. of ACM SIGGRAPH'93* (1993), 247–254. 24, 73, 74

[121] LEXT J., ASSARSSON U. AND MOLLER T.: A Benchmark for Animated Ray Tracing. *IEEE Computer Graphics and Applications, Volume 21, Issue 2* (2001), 22–31. 77, 82

[122] GOBBETI E. AND BOUVIER E.: Time-Critical Multiresolution Scene Rendering. *Proc. of IEEE Visualizatoin* (1999), 123–130. 24

[123] WINMMER M. AND WONKA P.: Rendering time estimation for Real-Time Rendering. *Proc. of the Eurographics Symposium on Rendering* (2003), 118–129. 24

[124] TACK N., MORAN F., LAFRUIT G. AND LAUWEREINS R.: 3D Rendering Time Modeling and Control for Mobile Terminals. *Proc. of ACM Web3D Synposium* (2004), 109–117. 24

[125] ROHLF J. AND HELMAN J.: IRIS Perfromer: A High Performance Multiprocessing Toolkit for Real-Time 3D Graphics. *Proc. of ACM SIGGRAPH'94* (1994). 381–395 24

[126] TELLER S.: Visibility Computations in Densely Occluded Polyhedral Environments. *Ph.D. thesis* (1992). 76

Wireless Networking for Graphics Applications

Point-Based Graphics

[127] Sain M, Pajarola R, Mercade A, Susin A, A Simple Approach for Point-Based Object Capturing and Rendering, IEEE Computer Graphics and Applications, July/August 2004, pp. 24-33. 19

[128] Duguet F and Drettakis G, Flexible Point-Based Rendering on Mobile Devices, IEEE Computer Graphics and Applications, July/August 2004, pp. 57-63. 19

[129] Duguet F and Drettakis G, Flexible Point-Based Rendering on Mobile Devices, INRIA Research Report 4833. 19

[130] Botsch M, Wiratanaya A, Kobbelt L, Efficient High Quality Rendering of Point Sampled Geometry, in Proc. Eurographics Workshop on Rendering 2002, Springer Verlag. 19

[131] Chen B and Nguyen M, Pop: A Hybrid Point and Polygon Rendering System for Large Data, in proc. IEEE Visualization 2001 19

[132] Cohen J, Aliaga D, Zhang W, Hybrid Simplification: Combining Multi-Resolution Polygon and Point Rendering, in Proc. IEEE Visualization 2001 19

[133] Grossman J P and Dally W J, Point Sample Rendering, in Proc. Eurographics Workshop on Rendering 1998, Springer Verlag. 19

[134] Rusinkiewicz S and Levoy M, Qsplat: A Multiresolution Point Rendering System for Large Meshes, in Proc. ACM SIGGRAPH 2000, pp. 343-352. 19, 25

[135] Yang S, Kim C and Kuo C, A progressive view-dependent technique for interactive 3D mesh transmission, in IEEE Trans. Circuits and Systems for Video Technology. 2004. 25

[136] Kim J, Lee S and Kobbelt L, View-dependent streaming of progressive meshes, in IEEE Trans. Circuits and Systems for Video Technology. 2004. 26

Spherical Harmonics

[137] Lindsay, C., Agu, E., Wavelength dependent Rendering Using Spherical Harmonics. in Proc. Eurographics 2005 19

[138] Sloan, P., Kautz, J., Snyder, J., Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments, ACM Transactions on Graphics, pg 527-536, 2002. 19

[139] Ramamoorhthi, R., Hanrahan, P., Frequency Space Environment Map Rendering. in Proc. ACMSIGGRAPH 2002, pg 517-526. 19

[140] Ramamoorthi, R., Hanrahan, P., An Efficient Representation for Irradiance Environment Maps. in Proc. ACM SIGGRAPH 2001, pg 497-500. 19

[141] Kautz, J., Sloan, P., Snyder, J., Fast Arbitrary BRDF Shading for Low-Frequency Lighting Using Spherical Harmonics. in Proc. 13th Eurographics workshop on Rendering, pg 291-296, 2002. 19