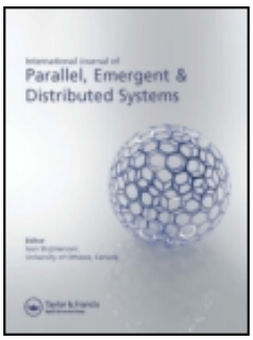

International Journal of Parallel, Emergent and Distributed Systems

ISSN: 1744-5760 (Print) 1744-5779 (Online) Journal homepage: http://www.tandfonline.com/loi/gpaa20

A middleware architecture for mobile 3D graphics

EMMANUEL AGU , KUTTY BANERJEE , SHIRISH NILEKAR , OLEG REKUTIN & DIANE KRAMER

To cite this article: EMMANUEL AGU , KUTTY BANERJEE , SHIRISH NILEKAR , OLEG REKUTIN & DIANE KRAMER (2006) A middleware architecture for mobile 3D graphics, International Journal of Parallel, Emergent and Distributed Systems, 21:3, 183-197

To link to this article: http://dx.doi.org/10.1080/17445760500355884

The International Journal of Parallel, Emergent and Distributed Systems,
Vol. 21, No. 3, June 2006, 183–197

Taylor & Francis
Taylor & Francis Group

A middleware architecture for mobile 3D graphics

EMMANUEL AGU*, KUTTY BANERJEE, SHIRISH NILEKAR, OLEG REKUTIN and DIANE KRAMER

Department of Computer Science, Worcester Polytechnic Institute, Worcester, MA 01609, USA

(Received 6 December 2004; in final form 1 July 2005)

Mobile graphics, which involves running networked computer graphics applications on mobile devices across wireless networks, is a fast growing segment of the networks and graphics industries. Running networked graphics applications in mobile environments faces a fundamental conflict; graphics applications require large amounts of memory, CPU cycles, battery power and disk space, while mobile devices and wireless channels tend to be limited in these resources. In order to mitigate mobile environment issues, some form of adaptation based on a client device's capabilities, prevailing wireless network conditions, characteristics of the graphics application and user preference, is necessary. In this paper, we describe the mobile adaptive distributed graphics framework (MADGRAF), a graphics-aware middleware architecture that makes it feasible to run complex 3D graphics applications on low-end mobile devices over wireless networks. In MADGRAF, a server can perform mobile device-optimized pre-processing of complex graphics scenes in order to speed up run time rendering, scale high-resolution meshes using polygon or image-based simplification, progressively transmit compressed graphics files, conceal transmission errors by including redundant bits or perform remote execution, all tailored to the client's capabilities. MADGRAF exposes our mobile adaptive distributed graphics language (MADGL), an API that facilitates the programming and management of networked 3D graphics in mobile environments.

Keywords: Middleware architecture; Mobile 3D graphics; CPU cycles; Memory constraints; Adaptive graphics; Remote execution

1. Architectural firm mobile 3D graphics scenario

The Ulo corporation is a multi-national architectural firm with clients and workers in 50 countries across five continents. Ulo maintains a large database of 3D architectural drawings of various types of buildings. Some of Ulo's clients are located remote areas with limited internet access and low bandwidth links. Additionally, to accommodate workers at various levels, Ulo has found it useful to equip its workers with PDAs, laptops and cell phones with graphics capability. Different teams of architects work on different projects which 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

*Corresponding author. Email: emmanuel@cs.wpi.edu

ISSN 1744-5760 print/ISSN 1744-5779 online
http://www.tandf.co.uk/journals
DOI: 10.1080/17445760500355884

seek more feedback, towards a mutually agreeable design. In cases where the client resides in a remote location, the architects can select a disconnect operation mode, in which case the relevant drawings are pre-downloaded (hoarded) onto the client, before disconnection. On reconnection, the hoarded files are reconciled with the database.

2. Introduction

Mobile graphics, which involves running networked computer graphics applications on mobile devices across wireless networks, is a fast growing segment of the networks and graphics industries. In on-site consulting situations (such as the Ulo scenario above), an interior designer or architect can show clients preliminary designs and seek feedback towards a mutually agreeable final solution. Field service technicians repairing complex equipment such as copiers and automobiles can retrieve and playback synthetically generated animations of repair manuals, focussing on relevant sections from new viewpoints. In electronic commerce, major corporations are already using interactive 3D catalogs to demonstrate product features, enabling rich interaction and virtual product examination.

Running networked graphics applications in mobile environments faces a fundamental conflict; graphics applications require large amounts of memory, CPU cycles, battery power and disk space, while mobile devices and wireless channels tend to be limited in resources. Specific issues to be dealt with include mobile device limitations, wireless channel, mobility and network infrastructure issues [3]. *Mobile device limitations* in memory, CPU power, disk space, screen size and battery life make it difficult to load and manipulate very high resolution geometric models or run sophisticated rendering algorithms that are necessary for visual photorealism. Highly interactive distributed applications cannot be supported due to *wireless channel issues* such as low transmission bandwidths and high, variable bit error rates (BER) that lead to prohibitively high, variable model download times. For wide area networks with wireless hops, *network infrastructure issues* such as variable latency and low bandwidth caused by slow network routers, switches and backbones further exacerbate networking. *Mobility issues* such as disconnection and hand-off require additional algorithms to manage the changing network topology, network address resolution and guarantee continuous service. Finally, any progress made in algorithms that run on the graphical processing unit (GPUs) of graphics cards has limited use since most mobile devices are still not equipped with 3D graphics accelerators.

In order to mitigate the above issues, some form of adaptation based on a client device's capabilities and prevailing wireless network conditions is necessary. *Application-aware adaptation* [14] in mobile computing dictates that any trade-offs made should exploit the specific nature of each type of mobile application. The mobile graphics industry is already embracing the adaptation theme for mobile graphics in directions that include the development of new efficient 2D vector graphics standards such as scalable vector graphics (SVG), reduced command-set languages and versions of graphics libraries such as OpenGL ES, wireless gaming engines, and low-power GPUs. However, most of these directions are niche solutions and more general solutions for mobile 3D graphics have been slower to appear.

The heart of the wireless graphics problem lies in the scalability of the currently adopted polygonal mesh representations, which show greater realism as the number of faces in a mesh increases. Today's state-of-the art meshes that are automatically produced by 3D scanning

can easily consist of billions of faces (or hundreds of gigabytes (GB)), each of which have to be rendered using a number of algorithmic steps, leading to an explosion in rendering complexity. Manipulating, processing and displaying such large models can be extremely unwieldy.

Research in mobile graphics tries to exploit the characteristics of graphics applications in adapting to resource constraints in mobile environments. Mobile graphics applications have varying degrees of interactivity, latency tolerance, refresh frame rate and scene complexity. Thus, different mobile applications will benefit from different forms of adaptation. For instance, a ray tracer performs thousands of floating point computations per pixel to determine the final pixel color, exhibits very low interactivity and would benefit from remote execution to offload expensive computations to a high-end server. A mobile gaming application on the other hand would requires low latency and would benefit more from a reduction in the resolution of geometric meshes, images and foreground graphics content to preserve interactivity.

We present the mobile adaptive distributed graphics framework (MADGRAF) [1], a middleware architecture for interactive 3D graphics in mobile environments. MADGRAF exposes the mobile adaptive distributed graphics language (MADGL), which facilitates the programming and management of networked 3D graphics in mobile environments. Specific research directions being investigated as middleware components in MADGRAF include device-optimized pre-processing of complex graphics scenes in order to speed up run time rendering, automatically scaling down the resolutions of complex scenes based on target mobile device capabilities [15], the offloading of draining rendering steps to more powerful servers (remote execution and cyber-foraging) [3], efficient transmission via compression [18], progressive (batch) transmission and rendering of large models [19], concealing the effects of wireless errors by including redundant bits [20], and the use of graphics-aware network protocols. We are also investigating machine learning decision algorithms in our centralized decision module at the MADGRAF server. By closely tracking resource usage at the client as well as graphics application performance, necessary trade-offs can be made to improve overall performance.

3. MADGRAF

The generic functions of a graphics library such as OpenGL and DirectX include scene modeling, import of predefined scenes, scene viewing and rendering. It is instructive to note that MADGRAF and MADGL do not duplicate these functions, but compliment them by focussing on aspects such as the import, adaptation and rendering of remotely stored graphics files, remote execution and geometric compression, which are of direct benefit to a mobile device. Our proposed research thrusts are being developed using MADGRAF as a foundation.

3.1 MADGRAF middleware

It is envisioned that MADGRAF will service a diverse spectrum of mobile graphics applications ranging from ray tracers to mobile games, in different ways. A set of middleware components can be combined in various ways for different applications, exploiting any

trade-offs that make sense for a given application. Therein lies the power of the MADGRAF framework. The main middleware components in MADGRAF are:

- *Geometric simplification*: algorithms convert an input high-resolution mesh into a lower resolution version (less polygons) of the original mesh [15]. Figure 1 shows several representations of the same mesh that are suitable for different mobile clients.
- *Image based simplification*: convert an input high resolution mesh into a sequence of high resolution images. The main advantage of converting to images is that even at high resolutions, images (or textures) have lower memory (storage) requirements than meshes. However, images permit less interaction than meshes [17].
- *Remote execution*: allows a portion of a computation task, or rendering in the case of graphics, to be performed on a remote server. In our many envisaged usage scenarios, the original high-resolution meshes will be stored on the server and remote execution could simply involve having the server start a portion of the rendering process before transmitting partially rendered results back to the client.
- *Geometry compression*: reduces the size of the geometric mesh prior to transmission [18]. Unlike ASCII compression (such as gzip on Unix systems), geometry compression exploits the connectivity of input meshes and is thus more efficient for meshes.

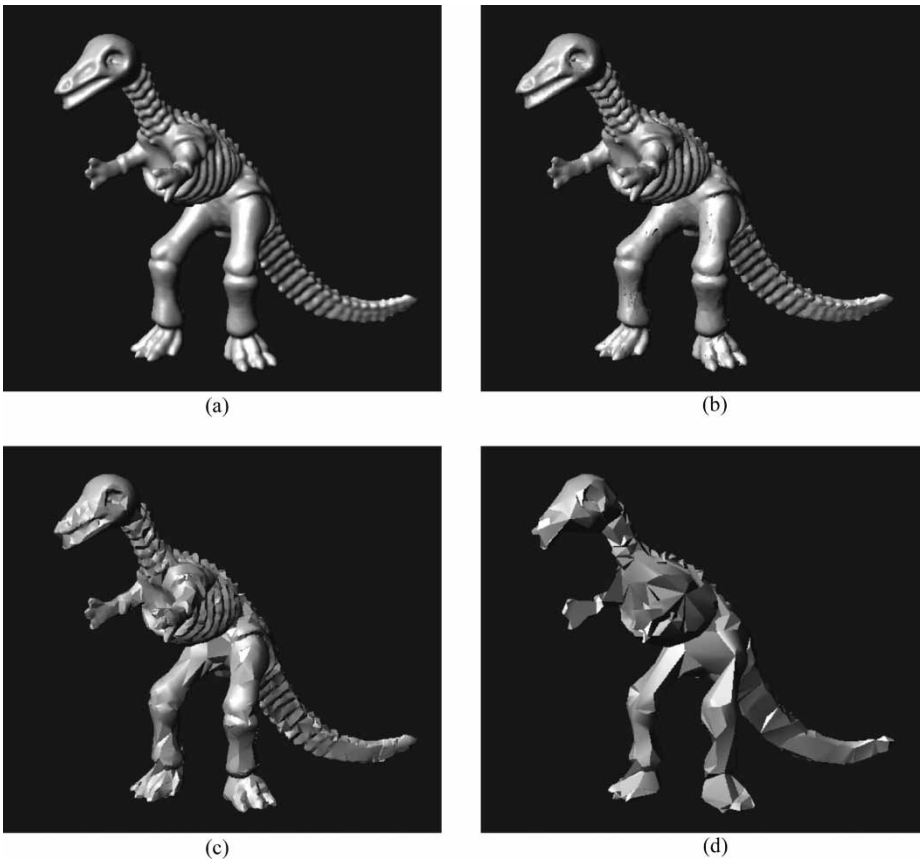

Figure 1. Various resolutions of a mesh suitable for different mobile clients.

- *Progressive meshes*: large high-resolution graphics mesh files can take a long time to download over most wireless networks, which generally limits interactivity. In certain situations, a quick rough preview image that could be improved iteratively would be adequate. For instance, an experienced interior designer who is trying to match a blue leather sofa to a virtual model of a room would probably reach a quick decision just by seeing a rough rendered image of the sofa in place. Progressive transmission [19] is a hierarchical data structure which encodes a large mesh such that a small base mesh is transmitted and rendered at first, followed by batches of mesh parts which could be added iteratively to increase the resolution of the rendered image [19].
- *Unequal error protection (UEP)*: is a forward error correction (FEC) technique that adds redundant bits to a mesh prior to transmission such that a client can repair meshes that are damaged due to transmission errors [20], and sometimes eliminate retransmissions. Since the progressive mesh data structure is hierachical, each layer contains meta information about the other layers. UEP recognizes this hierarchy and adds more redundancy to more important packets.
- *Graphics-aware wireless network protocols*: highly interactive graphics applications such as games are highly sensitive to network latency [21]. We have shown that latencies as low as 150 ms affect the performance of users in the first person shooter (FPS) class of games. However, our earlier studies investigate only latency patterns due to infrastructure bottlenecks arising from geographically distant multi-user game servers on the Internet. Multi-path fading and the hidden terminal problem [5] on wireless networks can lead to significant latencies. We would like to characterize the nature of latencies due to the wireless channel and its effect on graphics application users.

3.2 Examples of MADGRAF middleware usage

In the Ulo architectural firm scenario in Section 1, since the Ulo architects still require a significant amount of interaction with the building models, the models must be retrieved in a geometric mesh format. Since rendering the original mesh at full resolution may not be possible at interactive rates, *geometric simplification* based on the client's hardware configuration could be applied. The decision module at the MADGRAF server would determine what the output simplification ratio should be. The MADGRAF server also determines if it is necessary to perform part of the rendering of retrieved buildings for the client (remote execution), and also what level of compression or efficient transmission is necessary based on the bandwidth and prevailing error conditions on the wireless channel.

Since users of *interactive 3D shopping guides* expect high quality images to show off product features and can tolerate limited interactivity, image-based simplification of meshes is ideal. In a *mobile gaming* application, near and foreground characters require high interactivity and geometric simplification is preferable. Image-based simplification can be used for far and background images. In a *ubiquitous synthetic posters* application, the quality of rendered output from ray tracing engines is highly dependent on the resolution of input meshes and as little simplification as possible should be done. Additionally, this application has very low interactivity, making remote execution a natural choice. Table 1 summarizes the nature of mobile applications and related MADGRAF adaptations. In all the scenarios, although we highlight a main enabling MADGRAF adaptation, other adaptations may be

Table 1. Mobile graphics applications.

Application	*Interactivity*	*Latency*	*Scene complexity*	*Frame rate*	*Main MADGRAF adaptation*
Architectural firm	Medium	High	Very high	Low	Geometric simplification
Online shopping	Medium	Medium	Low	Medium	Image-based simplification
Realtime 3D game	High	Very low	Low	High	Geometric and image-based simplification
Remote ray tracer	Low	High	High	Low	Remote execution

used additionally. For instance, compression, unequal error protection or progressive transmission may be applied as well.

3.3 Overview of MADGRAF operation

In this section, an overview of MADGRAF's operation is now given. Figure 2 shows the main components of the MADGRAF architecture. The key server components for adaptation are the *Adaptive Graphics Engine*, the *Adaptive Transmission Engine* and the *IntelliGraph* (a centralized intelligent decision engine). The main client-side components are the *Environment Monitor* and the *Profile Generator*.

When a MADGRAF client program requests a 3D graphics file that is stored on a MADGRAF server, the client's Profile Generator retrieves information on the mobile host configuration and capabilities along with the current quality of the wireless communication channel from the Environment Monitor, and then builds the client profile, which it sends to the server along with the originally requested file. At the MADGRAF server, the request and client profile are forwarded to the intelliGraph, which decides if scaling is required and what optimizations and conversions should be done for the mobile client. Adaptation takes place in four main ways: (1) *adaptive graphics* such as geometric mesh simplification and image-based simplification, that pre-process input scene files; (2) *efficient transmission* techniques such as geometry compression, error concealment and progressive transmission [19] that address wireless network transmission bottlenecks; (3) *architectural* and distributed systems techniques such as remote execution, distributed rendering [1], and parallel rendering which

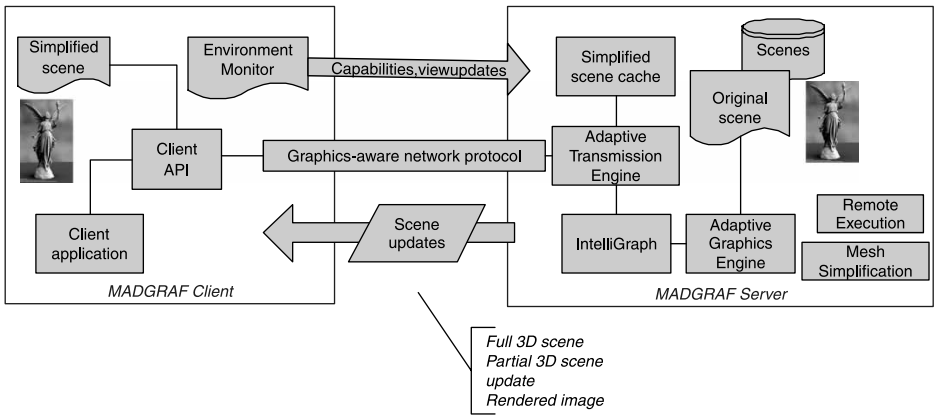

Figure 2. MADGRAF components.

attempt to borrow rendering resources from more powerful network servers; and (4) *intelligent* caching which retains the results of expensive computations for re-use in the future. These adaptations and the decision process forms the core of our short term goals and will be expounded later as specific projects in Section 5.

The IntelliGraph on the MADGRAF server manages the entire adaptation process, and utilizes the Adaptive Graphics Engine to generate files, which are best suited for use by a given mobile client, and the Adaptive Transmission Engine to transmit these files efficiently over a wireless network. The Adaptive Graphics Engine contains algorithms for geometric conversions such as scaling, polygon simplification, view-dependent optimizations and image-based techniques, while the Adaptive Transmission Engine contains efficient transmission algorithms such as progressive transmission, compression, and graphics-aware network protocols. Since a lot of the inter-conversions are computationally expensive and could potentially drain server resources, the client's profile and previously requested files are cached in the profile cache, while the resultant optimized geometric models are cached in the scene cache and may be re-used in subsequent requests.

The Environment Monitor gathers information and statistics about the MADGRAF client's operating environment, which are then used to build the client's profile, and is a key component for mobile host and communication channel adaptation. The information collected include *static attributes* such as the hardware specification (CPU, memory, hard disk and screen size) of the mobile host, *dynamic attributes* of the operating environment (channel error rate, download times, latency and system battery power) and *depreciating resources* such as battery power. The client profile may also include application requirements on speed and interactivity, quality of service (QoS) parameters and location information that may be useful for some optimizations.

4. IntelliGraph

Section 3.1 has described the different types of adaptations and optimizations that are possible in order to adapt graphics content for a mobile client. Usually a combination of these optimizations is applied and each combination of adaptations yields different rendering speeds and perceived distortion at the client. IntelliGraph is a centralized control center at the MADGRAF server, which decides what set of adaptations are necessary: specifically, what set of efficient transmission techniques are applied in the Adaptive Transmission Engine and what set of adaptive graphics techniques are applied in the Adaptive Graphics Engine. The secondary issues are what models and client profiles are cached at the server and the client, what stages of the graphics pipeline are mapped to the server and what stages are mapped to the client. Finally, also, what degree of parallelism and mapping to a multi-process, beowulf cluster or multi-processor machine. The IntelliGraph in the MADGRAF server uses information which it receives about the client's capabilities to make adaptation decisions in four passes: adaptive graphics, architectural efficient transmission and intelligent caching decisions.

4.1 Adaptive graphics decision

In general, combinations of graphics adaptations lead to a degraded synthetic image, which can be rendered at cheaper computational costs at the mobile client. A brute force search for

an optimal sequence of adaptations would, for a large number of possible optimizations, map to the classic knapsack problem in computer science algorithms. In order to compare and evaluate the effectiveness of any given set of adaptations, it is necessary to establish the degradation-computational savings relationship for that set of optimizations. Evaluation can be done using either *quantitative metrics* such as the number of polygons in the scene and refresh rate, or by using *perceptual metrics*, which give a measure of the perceptual impact on a user. Quantitative metrics tend to be easier to derive but perceptual metrics that require detailed user studies have more meaning, since the response of the user is the desired end result. Also, perceived levels of detail are not linearly correlated with quantitative metrics. For instance, a user may by unable to tell the difference between a model with 100,000 polygons and a simplified version with 50,000 polygons.

4.1.1 Quantitative metrics. Pasman [16] has derived the distortion introduced by two key types of degradation, polygon simplification and image-based rendering. For instance, for view independent simplification, the distortion D introduced by simplifying a given model to N polygons is shown to be given by

$$D = \frac{2k \arctan(r/d)}{N} \tag{1}$$

These results are important since they allow the IntelliGraph to compute an estimate of the distortion introduced by running a certain simplification or image-based optimization on a scene. Thus, it is possible to compare whole categories of adaptive graphics by computing a common metric, distortion. It is instructive to note that geometric simplification and image-based rendering are the two most popular categories of techniques used in resource-constrained graphics.

4.1.2 Perceptual metrics. Although calculating distortion is a first step towards determining how well a certain optimization works, an additional link needs to be made between user perception and the various forms of adaptive graphics, since in the end, the final consumer of the results of these trade-offs is the human perceptual system. Making this link will require detailed user experiments in the future; however, at this point, we shall formulate the perceptual decision problem at the IntelliGraph. For this we adapt the decision-theoretic approach proposed in [7]. The exact relationship between sets of adaptations and the visual perceptory system remains unclear. Although we use the methods in [7] it is also instructive to mention that several alternative bodies of work on what the most relevant visual factors are exist in visual cognitive psychology, but there is no concensus and it is somewhat unclear as to how these factors are to be used.

Humans typically scan rendered scenes in one of two modes, namely, *pre-attentive vision* and *attentive vision*. Pre-attentive vision continually scans large areas at a time in parallel, noting major or basic changes in a static of moving scene. Features efficiently detected by pre-attentive vision include overall color, size, luminance, motion, object shape and orientation. Attentive visual processes refer to the more serial, resource-constrained processes, which are required in order to recognize details about objects and relationships in scenes. In general, degradations in the quality of rendered scenes affect a human being's attentive senses and may be more prominent if they lead to discontinuities in color, texture or

motion. Since rendering artifacts are most noticeable in regions where a human being is focussing attention, degradation techniques (such as polygon simplification) exploit this and in general more resources (more detail) may be assigned to regions in which a user pays more attention. View-dependent techniques are a good example of this.

Modeling attention-based rendering can be divided into three main tasks: (1) establishing a relationship between perceived degradation in image quality and levels of adaptive algorithms; (2) deriving the probabilities that a user focusses attention on particular objects; and (3) combining the costs of degradation of individual pieces, objects or faces (or sprites) which make up the rendered image.

First, we define a perceptual cost function C^p (R_k, F_i) that gives a measure of the perceptual cost associated with each face, F_i in a scene as a function of adaptive rendering action R_k. A minimal perceptual cost C^p means more photorealistic images and vice versa. The R represents the combination or vector of graphics adaptations such as polygon simplification, which are applied to the rendered faces. The faces in the scene can then be combined by summation to give a total perceptual cost C^p as

$$C^p = \sum_i C^p(R_k, F_i) \tag{2}$$

It is important to note that the summation used above is somewhat naive since some interaction between multiple faces may lead to artefacts or pop-out effects. Next, we extend our perceptual cost function by including another variable, A, which captures the attentional focus on a face to get C^p (R_k, F_i, A). We also introduce $p(A^{F_i}|E)$, the probability that a user focuses on face F_i, given some evidence E. Evidence includes information about the scene to be rendered and inferences, which can be made from the general nature of the graphics task. Assuming a *continuous attentional model* in which attention is a scalar random variable for which minimum attention is at zero and maximal attention at one, the expected perceptual cost becomes

$$E[C^p] = \sum_i \int_{x=0}^{1} p(A^{F_i} = x|E) C^p(R_k, S_i, x) \mathrm{d}x \tag{3}$$

Since users naturally attend to *contiguous objects*, $p(A^{F_{ij}}|A^{O_j}, E)$ is the probability of attending to a face, F_{ij} conditioned on the viewer attending to a set of interrelated elements which are perceived as a contiguous object, O_j. The expected perceptual cost in equation (6) can be re-written as

$$E[C^p] = \sum_i \int_{x=0}^{1} p(A^{F_{ij}}|A^{O_j}, E) C^p(R_k, S_i, x) \mathrm{d}x \tag{4}$$

The probabilistic models of attention above can be constructed using information including the nature and context of the graphics task (e.g. game or ray tracer), direct mining from the input scene file, and meta-data embedded by the input scene creators. So far, the expected costs have been derived for static scenes. For dynamic scenes, the perceptual cost is modeled by considering the last n frames observed at time t as $C^p(t - n)$. In general, the more frames over which artefacts persist, the more noticeable they would be. For a dynamic scene the probability of attention becomes $p(A^{F_{ij}}|A^{O_j}, E, C^p(t - n))$.

The IntelliGraph seeks to minimize the expected perceptual rendering cost. An exhaustive search on all combinations of adaptive algorithms maps to an expensive knapsack problem

and is infeasible. However, classes of adaptation may be compared. More importantly, the ratio in computational gains from a certain class of adaptation can be compared to the perceptual loss. In most adaptations, the user accepts a certain level of error or distortion in the image. The *marginal computational cost* $\Delta C^c(R_r,R_a,F_i)$ is the difference in cost between rendering the original scene and rendering an adapted version and can be written as

$$\Delta C^c(R_r, R_a, F_i) = C^c(R_r, F_i) - C^c(R_w, F_i) \tag{5}$$

The perceptual cost contributed to the overall expected perceptual cost of the whole image by the adapted scene is given as

$$\Delta C^p(R_r, R_a, F_i) = \sum_i \int_{x=0}^{1} p(A^{F_i} = x|E) C^p(R_k, S_i, x) \mathrm{d}x \tag{6}$$

The expected *perceptual refinement rate*, $\Phi(S_i)$ is the ratio of the incremental gain in quality and the computational cost by using an adapted scene and can be expressed as

$$\Phi(S_i) = \frac{C^p(R_r, R_a, F_i)}{C^c(R_r, R_a, F_i)} \tag{7}$$

The IntelliGraph searches for a combination of adaptations which optimize the value of the perceptual refinement rate.

4.2 Architectural adaptation decision

In the architectural decision, the IntelliGraph decides where to locate stages of the graphics pipeline based on the capabilities of the mobile device, the capacity of the wireless channel and the degree of resources and parallelism, which the server can support. Different configurations and mappings of our distributed architecture will lead to different network communications overheads for the different stages. In order to evaluate the gains from various distribution scenarios, we need to quantify the computation-communication overheads inherent in each stage of the graphics pipeline and hence each possible distribution. Specifically, the storage and computation requirements of each stage and the inter-stage communication overhead are evaluated.

A simple analysis of each stage of the graphics pipeline now follows. The pipeline takes as input a vertex $(P_x,P_y,P_z,1)$ expressed in homogeneous coordinates. The modeling and view transforms, lighting and projection stages involve multiplication of the vertex with a 4×4 matrix in homogenous coordinates of the form

$$\begin{pmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{pmatrix} \tag{8}$$

The per-stage, per-vertex computation and storage overheads are roughly equivalent to the matrix storage and multiplication overheads. So that for a graphics scene with N vertices, a first order approximation leads to a stage computation overhead of PM and storage requirement of $P + M$ and the communication overhead incurred is that of transferring

intermediate matrix *PM*. The IntelliGraph evaluates the above overheads and decides where to locate the graphics pipeline stages.

4.3 Efficient transmission decision

Remotely located input graphics scene and mesh files, as well as intermediate results in our distributed architecture, require transmission. The combination of efficient transmission techniques such as encoding, compression and graphics-aware protocols selected is a complex decision. With transmission, assuming that the server has few limits on resources, the main overhead involved is the load of the techniques and time taken at the client. For instance, compression at the server requires decompression at the client. Hence, higher compression ratios will lead to more resources required for decompression. Typically, these client overheads are minor and in general, the most compact and efficient transmission techniques can be used, greatly reducing the need for optimality in this portion of the adaptive decision process. The available bandwidth is also a big factor in deciding what set of transmission techniques are used. For instance, for low bandwidth links like GSM, high compression ratios are important. Finally, some redundancy may be introduced into the meshes before transmission for the purposes of error concealment. The most suitable error concealment strategy is decided in the efficient transmission decision, based on prevailing wireless channel conditions.

4.4 Intelligent caching decision

When a server receives a client's request for a geometric model it computes the optimal representation of the model for the client. This process may be time and resource consuming. For instance, we found that it can take up to 10 min to simplify a 500,000 vertex input model by 98% for a cell phone. Running interactive graphics applications under these conditions would be difficult. As an optimization, the server may generate a discrete number of representations of all stored mesh models as a pre-process. All incoming client requests are mapped to the cached representations, which is closest to its actual calculated optimal representation. Also, the process of computing a client's optimal representation is also time-consuming and the same client may repeatedly request the same files. It is also useful to store the mapping of a client, its configuration and the computed representation and simply look that up on repeated requests. The specific number and increments of representations calculated is done during an Intelligent caching decision.

5. MADGRAF research projects

Using MADGRAF as a foundation, we now define the research thrusts, which we are currently pursuing in mobile 3D graphics. While the results of each thrust will generally lead to the development of a module of the MADGRAF system, the results should be general enough to apply to a wide range of mobile graphics systems. Due to space constraints, we simply outline these research thrusts, which are expounded further in [1]:

- *Budget-based graphics transcoders for heterogeneous mobile clients*: which develop metrics for the pre-processing of input meshes, subject to a *budget* on available memory, CPU or other mobile device resource.

- *Remote execution for mobile graphics*: in which based on prevailing conditions, each of the various stages of the 3D graphics pipeline can be flexibly mapped for either a client or surrogate server using our novel *pipeline-splitting* mechanism.
- *Efficient transmission of graphics content and wireless networking*: where we combine progressive transmission, geometric compression, unequal error protection (UEP) and graphics aware network protocol in order to satisfy application-dependent restrictions on latency, interactivity, image resolution and other suitable performance criteria.
- *Real-time performance monitoring and tracking*: in which we instrument the mobile device in order to constantly receive feedback for the demand and supply of its system resources and wireless channel condition.
- *Machine learning-based intelligence using history-based linear prediction*: in which by observing the resource usage of certain graphics models and optimizations, we develop heuristics for the automatic selection and employment of our middleware modules for the benefit of the mobile device.
- *MADGL API definition*: in which we are defining an intuitive powerful programmable interface to our system to increase flexible usage and rapid prototyping.
- *MADGRAF server optimization*: in which we are investigating scalability issues on the server based on workload analysis. We are also implementing *out-of-core* algorithms which are paging techniques used in the pre-processing of extremely large models that cannot fit into the server's memory.

6. Current MADGRAF prototype

Our initial investigations showed that very little research on adapting 3D graphics applications in mobile environments has been done. To get a concrete sense of the issues involved, as well as to fuel our research efforts, we chose to develop MADGRAF while prototyping. A minimal prototype has been developed. We shall now describe briefly our accomplishments thus far.

6.1 Accomplishments thus far

- *Minimal working prototype*: MADGL and MADGRAF were developed initially using an object-oriented design methodology with final implementation in Java, which due to its platform independence, allows MADGRAF to run on various mobile devices with little modification. A few third party tools were also used in developing MADGRAF, either as offline pre-processing tools (3D model converters) or as third-party components (polygon simplification, VRML viewers), which have been integrated into the prototype to reduce development time. The high-resolution polygonal meshes used models were retrieved from the Stanford 3D Scanning mesh repository [4]. Figure 3 is a screen shot of our MADGRAF prototype.
- *Automatic constraints detection*: Since we wanted to make MADGL and MADGRAF easy to use, we chose not to burden the client with specification of its resource constraints, configuration and network conditions, which are used by the server's IntelliGraph in decision-making. Constraints detection is done automatically once a MADGRAF client requests a graphics file. As such, it was necessary to develop

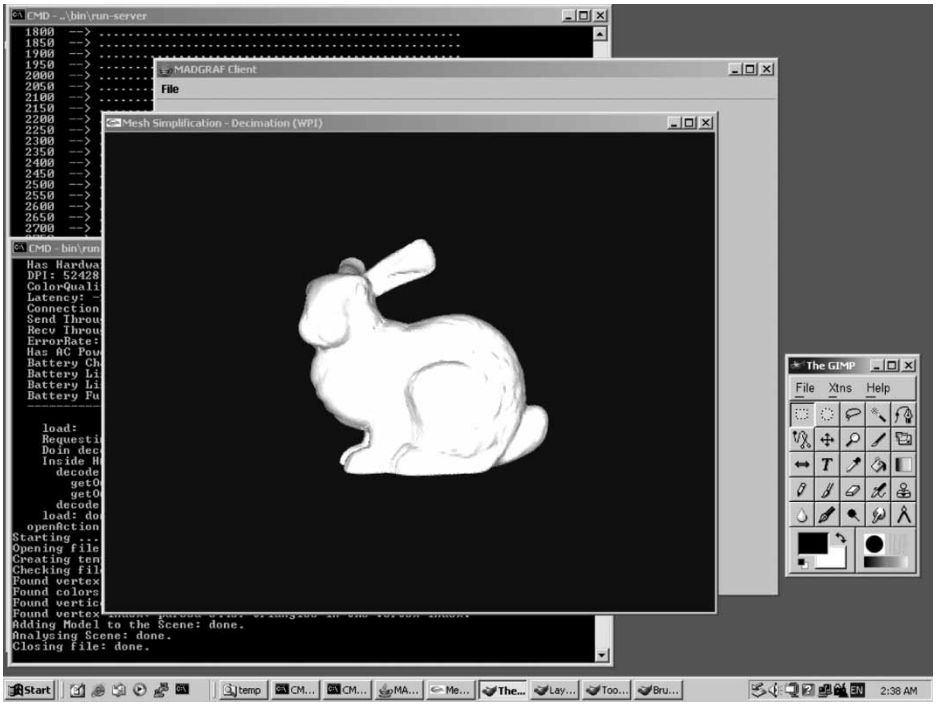

Figure 3. MADGRAF client screen shot.

platform-dependent modules for detecting client configurations and periodically sample prevailing device and network conditions. The MADGRAF constraints/environment detection module was developed using native code since it involves accessing specific operating system variables, which was not possible in pure Java due to Java's platform independent design. On a Win32 client, system constraints were accessed by querying the windows registry.

- *Energy profiler*: One key success is the development of *PowerSpy* [9], a software-only power meter for determining power usage on a Windows mobile client, to enable the server to make energy-efficient decisions.
- *Pipeline splitting for remote execution validated*: We instrumented a version of Mesa, a popular software implementation of the OpenGL graphics library, with networking code to create Remote Mesa (RMesa). Our preliminary tests show that our idea of remote execution for mobile graphics using fine-grained pipeline splitting has merits and are expounded in [8].
- *Preliminary system performance measurements*: Our working prototype allowed us to get a rough sense of how long the retrieval, transmission, simplification and rendering of meshes of a given resolution would take. These early measurements reinforced some of our initial assertions, and are informing our current directions. Table 2 is a sample of measurements on our current prototype. Since graphics applications are so different in nature, we are developing a wide range of test applications and perform detailed measurements as well as perceptual experiments to evaluate how much distortion is experienced by the human visual system.

Table 2. MADGRAF prototype performance measurements.

Percent reduction	*No caching*			*With caching*		
	Reduction time (ms)	*GZip time (ms)*	*Total time (ms)*	*Reduction time (ms)*	*GZip time (ms)*	*Total time (ms)*
05	136729	5943	142672	3825	4560	8385
20	340383	4950	345334	3819	4031	7850
30	455495	4376	459871	3809	3473	7282
50	653118	3152	656270	3857	2436	6293
80	802992	1225	804217	3836	1136	4972
99	925709	26	925735	3845	50	3895

7. Related work

Some of the adaptation concepts used in MADGRAF such as polygonal mesh simplification, image-based simplification [17], geometry compression [18] and progressive transmission [19] have been proposed separately in the literature. However, combining them in an integrated system for mobile environments is novel. Specifically, the application of mobile environment constraints in determining which sets of adaptations to use and to what extent, is a key contribution of the MADGRAF project. Augmented reality systems [12] integrate a number of these techniques but are immersive, while MADGRAF is non-immersive. Web-based graphics systems, [13], also deal with latency and bandwidth constraints on network performance.

A few related complete systems are also worth mentioning. The ARTE system [2] implements primarily adaptive graphics techniques such as polygon simplification and LOD techniques, but does not use remoted execution, parallelism, or any distributed rendering. ARTE also uses a simple decision model, which does not address specific device or wireless channel constraints. Repo3D [11] is a distributed graphics library, which primarily presents an object-oriented framework for the distribution of input graphics models, but does not address issues of resource-constraints.

8. Conclusion and future work

We have presented the MADGRAF, a middleware architecture for mobile 3D graphics. The MADGRAF prototype is serving as a proof of concept to validate our concepts. Our research thrusts and key accomplishments thus far have all been presented.

In order to keep our goals realistic, we have limited the middleware components described in this paper. However, several ideas for long-term research are already being considered. Other research themes that we would like to explore longer term include pre-fetching and hoarding techniques that allow offline operation in a disconnected mode, cyber-foraging [3] which allows a mobile graphics client to "lease" the services of new servers in new locations as it moves, and parallelization of server-side algorithms to enable rendering on clusters. Other research thrusts include using MADGRAF to scale graphics output to heterogenous displays in ubiquitous environments and point-based graphics for mobile devices.

Acknowledgements

Funding for this work was provided in part by the National Science Foundation grant number 0303592.

References

[1] Agu, E., 2003, *MADGRAF: A Distributed Architecture for Rendering Interactive 3D Graphics Applications in Mobile Environments*, Technical Report (Worcester: Worcester Polytechnic Institute).
[2] Boier-Martin, I.M., 2003, Adaptive graphics. *IEEE Computer Graphics and Applications*. **2, 1** (January–February), 6–10.
[3] Satyanarayanan, M., 2001, Pervasive computing: vision and challenges. *IEEE Personal Communications*.
[4] Stanford 3D Scanning Repository, http://graphics.stanford.edu/data/3Dscanrep/, November 2003
[5] Pahlavan, K. and Krishnamurthy, P., 2002, *Priniciples of Wireless Networks: a Unified Approach* (Englewood Cliffs, NJ: Prentice-Hall).
[6] Narayanan, D., Flinn, J. and Satyanarayanan, M., 2000, Using history to improve mobile application adaptation. *Proceedings of Third Workshop on Mobile Computing Systems and Applications (WMCSA)*.
[7] Horvitz, E. and Lengyel, J., 1997, Perception, attention, and resources: a decision-theoretic approach to graphics rendering. *Proceedings of UAI '97*. 238–249.
[8] Banerjee, K. and Agu, E., 2005, Remote execution for 3D graphics on mobile devices. *Proceedings of IEEE Wirelesscom*.
[9] Banerjee, K. and Agu, E., PowerSpy: fine-grained software energy profiling for mobile devices. *Proceedings of IEEE Wirelesscom*.
[10] Flinn, J. and Satyanarayanan, M., 1999, PowerScope: a tool for profiling the energy usage of mobile applications. *Proceedings of second IEEE Workshop on Mobile Computing Systems and Applications, New Orleans LA*.
[11] Macintyre, B. and Feiner, S., 1998, A distributed 3D library. *Proceedings of 25th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY: ACM Press), pp. 361–370.
[12] Gleue, T. and Dahne, P., 2002, Design and implementation of a mobile device for outdoor augmented reality in ARCHEOGUIDE project. *Proceedings of ACM Web3D*, pp. 161–168.
[13] Lau, R.W.H., Li, F., Kunii, T.L., Guo, B., Zhang, B., Magnenat-Thalmann, N., Kshirsagar, S., Thalmann, D. and Gutierrez, M., Emerging web graphics standards and technologies. *IEEE Computer Graphics and Applications*.
[14] Noble, B., Satyanarayanan, M., Narayanan, D., Tilton, J.E., Flinn, J. and Walker, K., Agile application-aware adaptation for mobility. *Proceedings of ACM Symposium on Operating Systems Principles*, Saint Malo, France, pp. 276–287.
[15] Luebke, D.P., 2000, A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications.* (May/June), 24–34.
[16] Pasman, W. and Jansen, F.W., 2003, Comparing simplification and image-based techniques for 3D clientserver rendering systems. *IEEE Transactions on Visualization and Graphics*. **9**(2), 226–240.
[17] Decoret, X., Durand, F., Sillion, F. and Dorsey, J., 2003, Billboard clouds for extreme model simplification. *Proceedings of ACM SIGGRAPH*.
[18] Deering, M., 1995, Geometry compression. *Proceedings of ACM SIGGRAPH*, pp. 13–20.
[19] Hoppe, H., 1997, Progressive meshes. *Proceedings of 24th Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY/Reading, MA: ACM Press/Addison-Wesley Publishing Co.), pp. 189–198.
[20] Al-Regib, G. and Altunbasak, Y., 2002, An unequal error protection method for packet loss resilient 3-D mesh transmission. *IEEE INFOCOM*. New York, NY **2**(June), 743–752.
[21] Sheldon, N., Girard, E., Borg, S., Claypool, M. and Agu, E., 2003, The effect of latency on user performance in Warcraft III. *Proceedings of ACM NetGames*.