Remote Execution for 3D Graphics on Mobile Devices

Kutty S Banerjee Emmanuel Agu
Worcester Polytechnic Institute
Computer Science Department
100 Institute Road
Worcester, MA 01609,USA*

Abstract

Mobile devices have limited processing power and wireless networks have limited bandwidth. A modern photorealistic graphics application is resource-hungry, consumes large amounts of cpu cycles, memory and network bandwidth if networked. Moreover running them on mobile devices may also diminish their battery power in the process. The majority of graphics computations are floating point operations which can run significantly slower on mobile devices which do not have floating point units or 3D graphics accelerators. Proposed solutions such as input mesh simplification are lossy and reduce photorealism. Remote execution, wherein part or entire rendering process is offloaded to a powerful surrogate server, is an attractive solution. We propose pipeline-splitting, a paradigm whereby 15 sub-stages of the graphics pipeline are isolated and instrumented with networking code such that they can run on either a mobile client or a surrogate server. To validate our concepts, we instrument Mesa3D, a popular implementation of the OpenGL graphics to support pipeline-splitting, creating Remote Mesa (RMesa). We explore various mappings of the graphics pipeline to the client and server while monitoring key performance metrics such as total rendering time, power consumption on the client and network usage and establish conditions under which remote execution is an optimal solution.Our results show that even with the incurred roundtrip delay, our remote execution framework can improve rendering performance by up to 10 times when rendering a moderate-sized graphics mesh file.

1. Introduction

Mobile clients such as PDAs, laptops, wrist watches, cell phones are rapidly emerging in the consumer market and an increasing number of graphics applications are being developed for them. However, current hardware technology limits the processing power, memory and disk space on these mobile devices and wireless network bandwidth can be scarce and unreliable.

A modern graphics application is resource-hungry and can consume large amounts of cpu cycles and memory and network bandwidth. Besides taking a very long time, running graphics applications on mobile devices may also diminish their battery power in the process.

Mesh simplification, and 3D compression have all been proposed to improve the performance of highly interactive graphics components such as the foreground characters in flight simulators and computer games. However, these techniques involve some form of degradation which compromise the photorealism of rendered images.

Remote execution of portions or the entire 3D graphics pipeline is a better fit for photorealistic rendering where the quality of the image increases with the number of faces or polygons in the input mesh model. However, key challenges exist. 3D graphics libraries are generally deployed as closely-coupled parts which reside on a single client machine, which makes distribution challenging. Also, user interaction with a remotely executing stage may incur a penalty of a roundtrip delay.

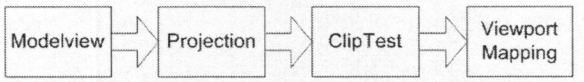
We propose pipeline-splitting, a novel paradigm in which the stages of the 3D graphics pipeline are isolated and networked such that each stage can be mapped to either the client or server. Thus, a weak mobile client can use a powerful surrogate server to execute parts or whole of low interactivity, compute-intensive graphics applications with the goal of increasing the overall execution speed and extending the battery life of the mobile host.

To validate pipeline-splitting, as part of our Mobile Adaptive Distributed Graphics Framework (MADGRAF) [2], we create Remote Mesa (R-Mesa) by instrumenting Mesa, an implementation of the OpenGL graphics API to support our novel pipeline splitting mechanism. We then run an OpenGL-based VRML browser as our test application and establish client-server mappings of the graph-

*Funding was provided in part by the NSF grant number 0303592

0-7803-9305-8/05/$20.00 ©2005 IEEE

(a) Individual stages of the graphics pipeline. Software renderers execute themsequentially. The Geometry stage operations comprise mostly floating point operations



(b) Geometry stage has further sub stages that are executed sequentially.

Figure 1. Graphics Pipeline Overview

ics pipeline stages which perform well, as well as conditions under which remote execution is an optimal solution for different input mesh model sizes. In addition to traditional performance metrics such as total rendering time, we extensively monitor the impact of various stage mappings on mobile client resources including battery and network bandwidth usage. We carry out our performance tests on both laptops (with floating point units) and PDAs (without floating point units) to investigate the impact of the absence of hardware floating point support. Our results show that even with the incurred roundtrip delay, PDAs can benefit from remotely executing floating point intensive operations of the graphics pipeline remotely.

2. The 3D Graphics Pipeline

Rendering (drawing) in computer graphics is typically organized such that input primitives such as triangles, quads, vertices are processed in a series of sequential algorithmic steps in the form of a pipeline[1].

The main stages of the graphics pipeline are the geometry, rasterization and per-fragment stages, as shown in figure 1(a). The "Geometry" stage performs geometric operations such as transformation, projection and clipping on individual vertices and primitives. The output of the geometry stage is fed directly to the "Rasterization" stage which converts primitives to fragments. This is later fed to the "Per-Fragment" stage during which fragments are discarded/modified on the basis of tests carried out.

[1]This is analogous to the process of refining crude petroleum wherein it goes through various purification processes

3. Pipeline Splitting

Typically, the entire 3D graphics pipeline is implemented as a closely-coupled library (such as OpenGL[9])which resides on a single machine. In this form, distribution requires that either the entire library resides on a client machine or on a surrogate server. We believe that this granularity of components is too coarse and propose a novel concept, called *pipeline-splitting* wherein we isolate and sub-divide the graphics pipeline into its stages such that the individual stages in figure 1(a) can execute on different machines. For instance, the Geometry stage could be rendered on the local client machine whereas the per-fragment and rasterization stages can be sent to a more powerful[2] server machine for execution. The functionality in each stage of the pipeline is replicated on both the client and server and the process of remote execution of a stage involves the following sequence of actions involving the client and the server: (1) Client packs the data required for executing the stage on the server into network format. This includes taking care of byte order, marshalling pointers[3] (2) Client sends this data across the network to the server (3) Server unpacks the data and converts it into the native format used by its machine (4) Server executes the stage (5) Server packs the data back into the chosen network format (6) Server sends this data across the network to the client (7) Client unpacks the received data into the format used on its local machine

For illustrative reasons, let us consider that the time taken to execute the rasterization stage on the client machine is 40ms and 10ms on the server for a certain graphics model. So, for remote execution of the rasterization stage, the time taken to transmit, pack, unpack data and execute the stage on the server must take less than 40ms for remote execution to be beneficial. Thus, the benefits of remote execution increases as the difference (assymmetry) in processing speeds of the client and server increase and network transmission times are reasonable.

Granularity of Pipeline Splitting: If each pipeline substage can be executed either locally or remotely, then we say that the granularity of pipeline splitting is a *single substage of the graphics pipeline.* Finer granularity implies more flexibility, more potential mappings and a wider range of supported mobile devices and graphics applications.

4 RMesa

To validate pipeline-splitting, we instrumented Mesa3D [11], an open source software implementation of OpenGL with socket networking code between the graphics substages (shown in figure 2)to create Remote Mesa (RMesa).

[2]in terms of CPU speed, memory, graphics card

[3]pointers need to be dereferenced before they can be sent across the network

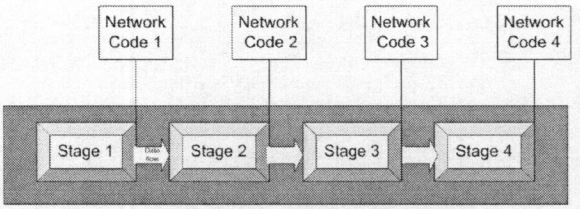


Figure 2. Networking the graphics pipeline

Thus, RMesa provides a framework that facilitates sub-stage level pipeline-splitting granularity. We shall now describe the main components of RMesa.

RMesa System Architecture: The base components include the RMesa Client and the RMesa server. The RMesa Client also has a 'Stage Map Control' unit to control the current stage mapping of the client.

RMesa Client: On the RMesa client, at the end of each stage,there is a placeholder for socket networking code. At each such point, the flow of a quick decision is made (or predetermined mapping is looked up) as to whether the data will be allowed to flow ahead if the next stage will be executed locally, or whether the data needs to be transferred to a remote machine for the execution of the next stage.

The RMesa Client consists of (1) *An OpenGL implementation with hooks in the graphics pipeline stages.* (2) *The capability at each sub-stage to execute the next stage on a remote machine with a mirror graphics pipeline.* (3) *A Provision to accept input that controls whether a particular stage is to be executed locally or remotely.*

Stage Map Control: The Stage Map Control Unit on the RMesa client is used to control or set on which machine (client or server) a given OpenGL sub-stage will run. The Stage Map Control is a separate process that connects to the RMesa Client using TCP/IP. Thus, this unit does not have to be present on the same machine as the RMesa client. The Stage Map Control unit in turn reads the stage map data from a configuration file which can be controlled either directly by the user or can be modified by any external decision component. Currently, we manually write to this configuration file. However, in the future , an external *Intelligent Unit* might decide on stage mappings based on more sophisticated algorithms and change this configuration file based on dynamic conditions of the wireless network and the machines in order to optimize speed, memory, power consumption and bandwidth usage.

RMesa Server: The RMesa server continuously waits for input from different client machines asking for particular stages of the graphics pipeline to be rendered, and processes such requests on a *First-In-First-Out (FIFO)* basis.

Each RMesa server can concurrently process many RMesa clients and state information about each connection is maintained by the RMesa client. Thus each RMesa client, informs the server of what stages it needs rendered and data (such as input or partially processed vertices) required for those stages. After executing the requested stages, the server closes the connection and does not maintain any further information about the closed connection. It should be noted that the RMesa server can concurrently handle multiple RMesa client requests which involve different stages of the graphics pipeline.

5. RMesa Implementation

In this section, we take an in-depth look at the instrumentation of Mesa carried out in order to obtain pipeline splitting.

Our RMesa implementation facilitates flexible mapping of 15 sub-stages to either the client or server (see figure 3(a)) and thus have location independence with respect to execution. ClipTest and Perspective Divide substages have been kept together due to the tight coupling of the data structures used within both in the Mesa implementation.

Configuration Files: The RMesa client connects to the remote RMesa server using a configuration file that specifies the server name and TCP port number. The Stage Map Unit uses a configuration file which *dynamically controls the current stage mapping of the RMesa client.* Thus if the values of the configuration file changes at runtime, then the stage mapping used by the RMesa client also changes.

Platform: RMesa is currently instrumented to work on a Win32 port of Mesa OpenGL. Under Windows, when compiled, RMesa produces a dynamic link library, OpenGL32.dll which must be placed in the current directory of the application. This causes windows to load OpenGL32.dll of RMesa into the applications' process memory instead of the default Mesa or any other OpenGL vendor libraries. Since RMesa is a modified Mesa library, therefore it has the basic OpenGL capability in addition to ability to split the graphics pipeline and communicate with a remote server.

6. RMesa Performance Analysis

We set up simple experiments to validate our pipeline-splitting concept and RMesa implementation. The key goal of these experiments was to determine specific mesh sizes and graphics workloads for which remote execution is an optimal strategy. We ran our tests on both a high end Dell laptop and an iPaq PDA. To better understand how well RMesa's stage mapping worked, we ran several test cases while monitoring overall rendering time, battery power consumption of the mobile device and network bandwidth usage. Before presenting our results, we shall first describe some of our test cases.

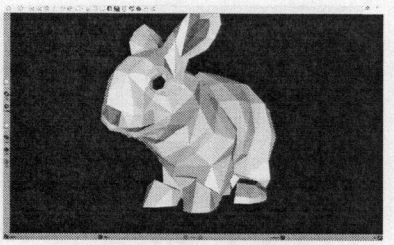


Figure 4. A sample VRML file

6.1 RMesa Test Cases

In order to evaluate the performance of a graphics application using RMesa, we chose to investigate the following stage maps in detail:

1. *Case I:* The Modelview substage of the Geometry stage is rendered on the server as is shown in figure 3(a) . All other stages and substages are rendered on the mobile client.

2. *Case II:* The Normal substage of the Lighting stage is rendered on the server and all other stages and substages are rendered on the mobile client.

3. *Case III:* The Depth-Test substage of the Rasterization stage is rendered on the server and all other stages and substages are rendered on the mobile client.

Specifically, using these test cases we will study the timing, power consumption details and network usage of graphics applications using RMesa library.

6.2. Experimental Setup - Laptop

The mobile client used in our test was a Dell Inspiron laptop with a 1.7GHz CPU and 64MB of RAM. The server had a 2.4GHz CPU and 1GB of RAM.

In order to control the number of vertices generated and sizes of our input VRML files, we created them using the AC3D[10]. A cylinder rendered with 'X' number of segments produces a VRML file containing roughly '2X' number of vertices. Each of these files are then rendered using OpenVRML[12], a VRML File Browser that uses OpenGL for rendering. OpenVRML was made to render using RMesa instead of the original Mesa OpenGL library. A sample rendered image of 'bunny.wrl', a VRML file, is also shown in figure 4.

6.2.1 Rendering Time Results Analysis

Case I: In this test case, the modelview substage is rendered on the server machine with all other stages and substages being rendered on the local machine. Figure 3(b) compares the time taken for remote rendering against a complete local machine rendering. The graph shows that for VRML files with lower number of vertices, the rendering time is by and large the same. Close to around 1000 vertices, the rendering time taken on the server starts decreasing.

Case II: Results in [1] reveals that the rendering performance for remote mapping shows a steady improvement in performance with increase in number of vertices. However, close to 40,000 vertices, the difference in rendering performance between the remote and local mappings gets marked, with the remote mapping being the better one.

Case III: Results in [1] shows that when the Depth Test substage is rendered on the server machine, the response is similar to that of Case II.

The trend clearly shows that as the number of vertices in the graphics image increases, remote execution starts to give a clear advantage over local execution.

6.3. Experimental Setup - PDA

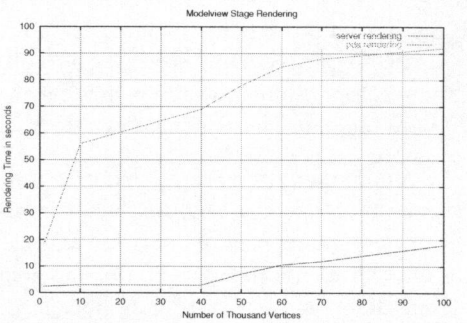
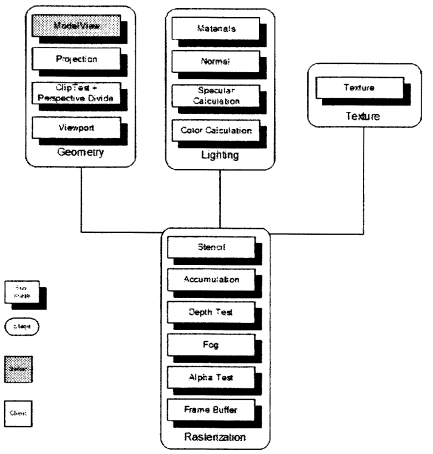


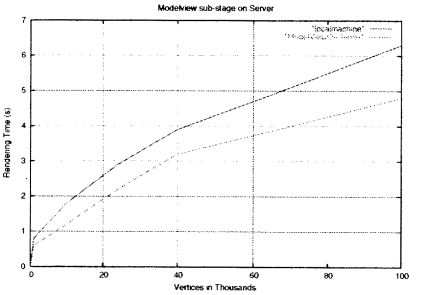
Figure 5. Results for PDA client with modelview stage rendered on the server

Encouraged by the improvement in performance on two processors with differing speeds, we performed our tests on a *iPAQ h4300* with 64MB memory and 400MHz processor *without floating point support in hardware*. The server used is the same as in section 6.2. The matrix multiplication operations involved in the modelview, projection and clipping geometry stages of the graphics pipeline were remotely executed on the server, while keeping all other stages on the PDA. The goal behind this experiment was to determine:

- What is the impact of increasing difference in processing speeds of server and client. We have already seen a gain in 1 second with a nominally weaker client in



(a) Pipeline View



(b) Modelview on Server Performance

Figure 3. Modelview rendered on server

section 6.2. By how much does this 1 second margin increase, or does it remain stagnant?

- What is the impact of executing floating point in software on a PDA? Since PDAs normally do not have floating point support in hardware, by how much does the difference in rendering time increase as a result.

As seen in figure 5, the difference in rendering time, increases by a large margin. Our analysis being that since PDAs lack floating point support in hardware and the Geometry stages of the graphics pipeline are floating point intensive, rendering them on a server using remote execution improves performance largely.

7. Power Profiling of RMesa

We decided to profile the energy usage of RMesa in order to ensure that the impressive speedups were observing were not at the expense of precious battery power. In this section, we discuss the power consumed by an open source VRML browser, OpenVRML [12]. OpenVRML is run with the modified OpenGL library, RMesa. The power measurement is carried out using PowerSpy[3].

7.1. Experimental Setup

The client machine used for this purpose is a Dell Inspiron laptop with 2.4 GHz CPU and 1GB memory. The server machine used is a desktop with a processor of 1.2 GHz and 128 MB of memory. VRML files with 1K, 10K and 100K vertices are profiled for their energy use. Each VRML file is tested under the following test conditions: (1) *Geometry Stage on server* (2) *Lighting Stage on server* (3) *Rasterization Stage on server* (4) *Texture Stage on server* (5) *Application Stage on server* (6) *Client Only*

Stage Mapped to Server	Power Consumed (mWH)
Geometry Stage	47
Lighting Stage	45
Rasterization Stage	49
Texture Stage	46
Application Stage	47
Client Only	45
Server Only	21

Table 1. Power Profile of 10K-Vertex File

The total power consumed by the application for each of the above stage mappings is tabulated. Rendering with the *Application Stage on the server* implies that the VRML file is present on the server side and that, only processed vertices are transferred back to the client.[4]

7.2. Energy Profiling Results

A VRML file with 10K vertices profiled for power consumption revealed the results tabulated in table 1. As observed previously in [3], the network interface consumes considerable amount of energy, and the network involved in network transmission during remote execution increases the expended battery energy compared to execution entirely

[4]processed vertices being those that have been operated upon by stages of the graphics pipeline.

done on the mobile client. Interestingly, complete rendering on the server shows a significant reduction in energy used, but is unattractive for graphics applications which are interactive in nature. In the future, we hope to explore ways of making use of more server side graphics techniques.

The power consumption for a *server only* rendering process is the least among all the other stage maps. This power saving increases as the number of vertices in a graphics file increases. The energy results for VRML files with 1K and 100K vertices show similar results to the 10K file and are shown in [1].

8. Network Profiling

The data transfer between the client and the server was profiled using Windump[13] in order to get insights into network activity. For each VRML file, the network data transfer is profiled for the different stage mappings. We found that overall network usage was moderate. Typically, first there was a period of client to server data transfer, followed by a middle period of lower or no network activity during which the server machine operates on the stage/substage for the client. Finally, there is further network activity when the server transmits results back to the client. [1] contains our complete results for network profiling including results for various VRML file sizes, as well as the network profile for the Normal substage and depth test substage being rendered on the server.

9. Related Work

Several bodies of work are related to our work. These include graphics architectures such as *WireGL*[8] and *Chromium* [7] that target clusters of servers, other remote execution frameworks for mobile devices such as that in Spectra in project Aura [6] that do not focus on graphics applications, and Parallel Mesa, [4] that parallelizes the graphics pipeline across multiple servers.

10. Conclusion and Future Work

In this paper, we have proposed a novel pipeline-splitting concept that facilitates remote execution of graphics applications that use OpenGL. RMesa provides the capability to flexibly render 15 graphics pipeline sub-stages either on a local mobile client or on a remote surrogate server. We have thoroughly investigated various stage mappings of the graphics client and observed performances in terms of rendering speed, power consumption and network activity. Although we include specific results of remote execution for a specific client, server and wireless LAN, optimal mappings will vary for different machines and thus, the true power of RMesa lies in the flexibility it provides and the potential it has for new mappings to fit into the requirements of future applications.

We are already working on ways to improve the overall structure of RMesa using a more object-oriented approach that will be implemented with middleware toolkits (such as CORBA [14]) that more naturally support distributed components. Work is currently on to provide automatic operating system support and integrate RMesa with an external intelligence module called Intelligraph which dynamically monitors the client machines' work load, memory consumption and network bandwidth to control the stage map used on the RMesa Client machine. Intelligraph is coupled with a database that has been filled with ideal stage maps for different client machines.[2].

References

[1] Kutty S Banerjee, Emmanuel Agu, *Remote Execution for 3D Graphics in Mobile Devices*, WPI CS Dept Tech Report, 2004

[2] E Agu *et al*, *A Middleware Architecture for Mobile 3D Graphics*, in Proc. IEEE MDC 2005 (to appear).

[3] Kutty S Banerjee, Emmanuel Agu. *PowerSpy: Fine-Grained Software Power Profiling for Mobile Devices*,in Proc. IEEE WirelessCom 2005 (to appear).

[4] Tulika Mitra, Tzi-cker Chiueh, Implementation and Evaluation of the ParallelMesa Library, *in Proc. IEEE Conf. Parallel and Distributed Systems 1998.*

[5] OpenGL(R) Reference Manual: The Official Reference Document to OpenGL, Version 1.2 (3rd Edition). *Addison-Wesley Publishing Company.*

[6] Jason Flinn, Dushyanth Narayanan and M. Satyanarayanan. Self-Tuned Remote Execution for Pervasive Computing. *in Proc HotOS-VIII, May 2001.*

[7] Greg Humphreys *et al*, Chromium: A Stream-Processing Framework for Interactive Rendering on Clusters. *in Proc. ACM SIGGRAPH 2002*

[8] Greg Humphreys *et al*,WireGL: A Scalable Graphics System for Clusters. *in Proc. of SIGGRAPH 2001.*

[9] OpenGL website, *www.opengl.org*

[10] AC3D 3D Authoring Toolkit, *www.ac3d.org.*

[11] Mesa3D Webite, *www.mesa3d.org*

[12] OpenVRML VRML Browser, *www.openvrml.org*

[13] Windump Website, *windump.polito.it*

[14] CORBA Website, *www.omg.org*