International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:2, 2009

Experimental Parallel Architecture for Rendering 3D Model into MPEG-4 Format

Ajay Joshi, and Surya Ismail

Abstract**—**This paper will present the initial findings of a research into distributed computer rendering. The goal of the research is to create a distributed computer system capable of rendering a 3D model into an MPEG-4 stream. This paper outlines the initial design, software architecture and hardware setup for the system.

Distributed computing means designing and implementing programs that run on two or more interconnected computing systems. Distributed computing is often used to speed up the rendering of graphical imaging. Distributed computing systems are used to generate images for movies, games and simulations.

A topic of interest is the application of distributed computing to the MPEG-4 standard. During the course of the research, a distributed system will be created that can render a 3D model into an MPEG-4 stream. It is expected that applying distributed computing principals will speed up rendering, thus improving the usefulness and efficiency of the MPEG-4 standard

Keywords**—**Cluster, parallel architecture, rendering, MPEG-4.

I. Introduction

THE goal of this project is to create a working cluster of computers to perform distributed parallel rendering of an interactive 3D model. The system is designed to be used as the basis for more advanced research in the future. For example, the system could be used to study new distributed rendering algorithms or to measure the efficiency of different high performance techniques.

The focus of this stage of the research is on building a stable, working system in a reasonable amount of time, for a reasonable amount of money. Therefore, performance and system optimization are not a high priority.

The system will use off-the-shelf, low to mid end, readily available computers. Linux, an open source operating system will be used. This choice is actually beneficial as Linux has a long history of being used for distributed rendering. Many modern rendering farms are built around Linux based clusters.

Open source applications and libraries will be used. As much as possible, exiting libraries and source code will be used for parallelization and rendering. The system will be based on open standards and established protocols. These choices will help keep the cost down and will allow the project to move faster by legally taking advantage of existing work.

Ajay Joshi is with the University of the West Indies, St. Augustine campus, Trinidad and Tobago (phone: 868-662-2002; e-mail: ajoshi@ eng.uwi.tt).

Surya Ismail was pursuing Masters degree at Multimedia University, Persiaran Cyberjaya, Malaysia.

The project proposes two additions to a standard distributed rendering system, interactivity and output into an MPEG-4 stream. These features aim to add functionality and usability to the system.

Interactivity will allow the model to evolve based on external input. The input can take many forms, including interaction through input devices, input from sensors and input from data files. Interactivity will allow the system to function as more than a static visualization tool. It can potentially be used for applications such as simulation, modeling and game-playing.

MPEG-4 is a standard from the Moving Pictures Experts Group [1]. It is gaining industry support. For example, it is the basis of the popular DivX player. The standard focuses on adding new features to existing video and audio streaming capabilities. In incorporates elements from Apple QuickTime and VRML. It adds support for things positional sound and 3D objects.

Many devices, including PDAs and mobile phones, can or will support viewing of MPEG-4 streams. Adding support for MPEG-4 means that any of these devices can potentially view the output of the distributed rendering system.

II. Physical Setup of Cluster

At the hardware level, the system is configured as a small cluster. Readily available computers and components were used. It is expected that as more resources become available, more computers and better quality components can be added.

The computers are connected to a switch at 100Mbps, or fast-ethernet, connection. The switch is capable of connecting four computers. This is acceptable for a small cluster. If resources become available for a large cluster, the switch will have to be upgraded.

For this initial phase, the cluster will consist of four computers, two mid-range Pentium 4 machines, and two mid-range Pentium 3 machines. These computers can be deployed in a variety of manners depending on the needs of the software architecture of the system. One of the most common setups would be to use one of the Pentium 4 machines as a master with the other three as slaves.

The current setup is probably not good enough for interactive rendering of high-polygon models. However, it should be good enough to demonstrate distributed rendering of medium sized models at a reasonable frame-rate. This should be sufficient to meet the goals of this project.

III. Software Setup

Given the hardware setup, there are many software approaches that can be tried. These approaches differ in the level of implementation effort, performance profiles and effectiveness.

These software approaches share some commonalities. All the software systems are built with the assumption that the 3D model is stored in a Blender file. Blender [2] is an open source application for 3D modeling, similar to 3D StudioMax and Maya. It is a mature program with a rich feature set, strong API support and solid documentation. There are many Blender files, with a variety of different types of models, readily available for testing.

All the proposed software systems are expected to encompass some form of interactivity and generate output in the form of an MPEG-4 stream.

A. Parallelization through Animation Frames

Parallelization through animation frames is achieved through the use of an OpenMosix [3] cluster. OpenMosix is an application that automatically parallelizes applications based on processes. Suppose a rendering application is run on the cluster. The application is multi-threaded and spawns multiple processes. OpenMosix will assign the processes to different machines in the cluster. OpenMosix performs load balancing to make sure that each machine is used optimally.

The approach uses Blender's existing animation rendering engine. The engine renders blender files into animation frames. The engine is multi-threaded and is ideal for parallelization on an OpenMosix cluster [4].

The idea is to schedule the rendering engine to run at fixed intervals. At the beginning of each interval, any input from user interaction is captured. The appropriate changes are made the animation sequence. Then the rendering engine is called to generate the sequence. The rendering engine spawns processes which in turn are fed to OpenMosix. OpenMosix assigns these processes to the various machines in the cluster. At the end of the interval, all the frames are collected and fed to an MPEG-4 component to be turned into an MPEG-4 stream. (See "Fig. 1. Example of OpenMosix cluster")

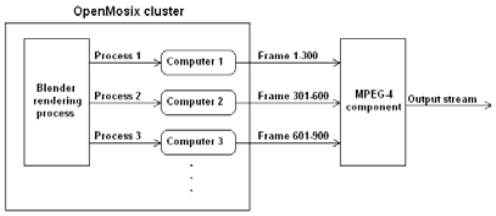

Fig. 1 Example of OpenMosix cluster

This approach is among the easiest to implement because it requires no changes to the rendering engine to allow it to work. However, there is a reasonably high communication overhead for using an OpenMosix cluster. Current implementations show that this form of parallelization only provides performance benefits for animation sequences of more than 1000 frames.

With a frame rate of 50 to 60 frames per second (standard for interactive games), this means that 15 to 20 seconds worth of animation need to be rendered at one time. This means that the system can only respond to input 3 or 4 times a minute. This is acceptable for applications with low interactivity, but unacceptable user interactivity. The frame by frame approach also limits the system's ability to utilize the full potential of the MPEG-4 standard.

B. Parallelized Rendering Library

This software approach works by replacing the underlying 3D rendering library with a parallelized version. This technique works because a large number of applications do not use their own rendering engines. Instead they rely on generic rendering libraries like DirectX or OpenGL.

OpenGL is of primary interest to this project because it is an open standard with open source implementations. It has strong industry support with many popular graphics cards supporting it in hardware. The Blender GUI and rendering engine utilize OpenGL.

The WireGL project [5], at the Stanford University Computer Graphics Lab, is a good example of a parallelized rendering library. WireGL implements the exact same API calls as OpenGL. Therefore any program designed for OpenGL can run in a distributed environment using WireGL.

WireGL uses a sort first parallelization algorithm along with some other optimizations for memory and band-width management. It was initially designed for tiled display systems, so essentially, it divides the work among the various machines in the cluster by partitioning the camera space into tiles. Each machine is assigned a tile to render. All the tiles are then collected and reassembled. For this project, as a final step, the reassembled tiles are built into an MPEG-4 stream. (See "Fig. 2. Example of WireGL cluster").

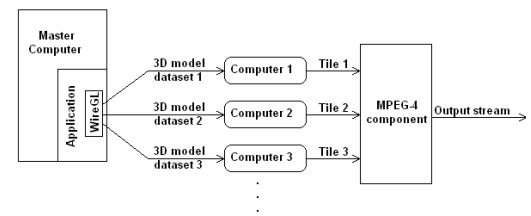

Fig. 2 Example of WireGL cluster

When tested on a medium sized cluster (16-32 nodes), WireGL was able to achieve good frame rates, even for high-interactivity applications. One of the test applications was a high-interactivity game. However, this approach still does not fully utilize the capabilities of MPEG-4.

C. Custom Rendering Engine

A custom rendering engine is the end goal of this project. The basic structure of the system will involve a PVM driven Linux based cluster. The interactivity is driven by Python scripts embedded into the Blender files.

Parallel Virtual Machine (PVM) [6] is a software package that permits a heterogeneous collection of networked computers to be used as a single large parallel computer. At

its' heart, it is a C API that can be used to simplify distributed computing.

Python [7] is an interpreted, interactive, object-oriented programming language. Python's interpreted nature makes it a good match for Blender. Blender provides a growing collection of Python APIs for a variety of purposes. Python scripts are available for controlling animation, textures and most other aspects of the 3D model. There is even an API for game logic.

The goals of the custom rendering engine should be consistent with the goals of the project. Therefore, the focus here is on creating a stable source code base that can be used for developing and testing new rendering algorithms (See "Fig. 3. Architecture of custom rendering engine").

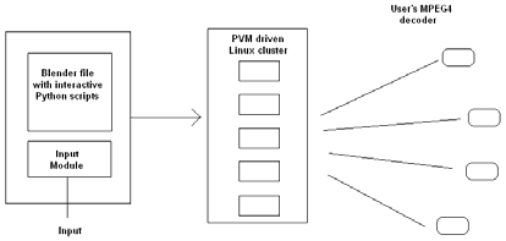

Fig. 3 Architecture of custom rendering engine

The proposed architecture will be based on Multi Instruction Multi Data parallelization with each node running its own set of data. A polygon based rendering approach will be used as opposed to ray tracing.

At this point, it is planned that a rendering library approach similar to WireGL will be used. The Mesa [8] implementation of OpenGL will be used as the initial code base. Mesa is open source, mature and well documented and is therefore an ideal starting point.

Unlike WireGL, the system will not use camera space partitioning. A sort-middle algorithm is proposed for the initial implementation. Since the focus is on rendering and not on network management, much of the 3D modeling data will be stored on each individual machine, distributed before rendering begins. This will reduce network traffic. However this approach assumes that each machine has a reasonable amount of memory and the models themselves are not too large.

One interesting aspect of this setup will be experimenting with how to best utilize the newer features of the MPEG-4 standard, mainly ability to specify 3D-objects. It will be possible to study the balance between performing work on the cluster and on the output device. It may be possible to attempt different divisions of labor for different output devices.

IV. Conclusion

This stage represents a good initial step in a long term research project to study distributed rendering. Even in this early stage, many things can be learned about new protocols (MPEG-4) and techniques for (OpenMosix, WireGL).

It is hoped that this initial design will form a strong basis for future, original, research into new distributed rendering algorithms and techniques. Along with the hardware and software setup, the process of designing and implementing this stage will deepen the researchers' understanding of distributed rendering.

References

[1] http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm
[2] http://www.blender.org
[3] http://openmosix.sourceforge.net/
[4] http://spot.river-styx.com/viewarticle.php?id=12
[5] G. Humphreys, I. Buck, M. Eldridge, and P. Hanrahan, "Distributed rendering for scalable displays", SC2000: High Performance Networking and Computing, ACM Press and IEEE Computer Society Press, Dallas Convention Center, Dallas, TX, USA, November 4–10 2000, pp. 60-60.
[6] http://www.csm.ornl.gov/pvm/pvm_home.html
[7] http://www.python.org
[8] http://www.mesa3d.org
[9] Rudrajit Samanta, Jiannan Zheng, Thomas Funkhouser, Kai Li, and Jaswinder Pal Singh "Load Balancing for Multi Projector Rendering Systems",SIGGRAPH/Eurographics Workshop on Graphics Hardware, Los Angelos, California - August, 1999.
[10] A. Bilas, J. Fritts, and J. P. Singh. "Real-Time Parallel MPEG-2 Decoding in Software."InProceedings of InternationalParallel Processing Symposium, 1997.
[11] Y.Chen, C.Dubnicki, S.Damianakis, A.Bilas, and K. Li. "UTLB: A Mechanism for Translations on Network Interface." In Proceedings of ACM Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII),pp193-204, October 1998.
[12] T.W.Crockett. "An Introduction to Parallel Rendering." Parallel Computing,Vol 23,pp819-843, 1997.
[13] S. Upstill, The Renderman Companion, Addison-Wesley, Reading, MA, 1989.
[14] Bengt-Olaf Schneider, Parallel Rendering on PC Workstations, International Conference on Parallel and Distributed Processing Techniques and Applications (PDTA98), Las Vegas, NV, 1998.
[15] M. Berekovic, P. Pirsch, "An Array Processor Architecture with Parallel Data Cache for Image Rendering and Compositing," cgi, p. 411, Computer Graphics International 1998 (CGI'98), 1998.

Ajay Joshi (M'04) became a Member (M) of **IEEE** in 2004. Author holds a Ph.D. with specialization in advanced computer architecture, from the Institute of Science, University of Mumbai, India in 1996.

He is currently with the department of Electrical and Cmoputer engineering, The University of the West Indies, Trinidad and Tobago. Earlier he has worked with the Multimedia University, Malaysia as a lecturer, prior to this he was head of technology at the IBM advanced training center at Nasik. His previous publications are in the field of embedded systems hardware related to memory and parallel processing.

Dr. Joshi is a member of IEEE and acts as a member of the technical committee for IADAT conferences. He has acted as a session chair at the IADAT conference at Spain.