TEC-0065

Scalable Processing for Distributed Simulation and Scene Generation

Brahm A. Rhodes
Steve D. Bronson

Onyx Sciences Corporation
124 Mount Auburn Street
Suite 200N
Cambridge, MA 02138

July 1995

U.S. Army Corps of Engineers
Topographic Engineering Center
7701 Telegraph Road
Alexandria, Virginia 22315-3864

US Army Corps of Engineers
Topographic Engineering Center

T

E

C

Destroy this report when no longer needed.
Do not return it to the originator.

The findings in this report are not to be construed as an official Department of the Army position unless so designated by other authorized documents.

The citation in this report of trade names of commercially available products does not constitute official endorsement or approval of the use of such products.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1995	Technical Apr. 1994 - Oct. 1994

4. TITLE AND SUBTITLE

Scalable Processing for Distributed Simulation and Scene Generation

5. FUNDING NUMBERS

DACA76-94-C-0007

6. AUTHOR(S)

Brahm A. Rhodes
Steve D. Bronson

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Onyx Sciences Corporation
124 Mount Auburn Street
Suite 200N
Cambridge, MA 02138

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Topographic Engineering Center
7701 Telegraph Road
Alexandria, VA 22315-3864

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

TEC-0065

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This report details the results of researching and evaluating massively parallel architectures and visualization techniques. The main focus of this project was to evaluate and determine the potential of parallel computing systems for visual simulation applications, such as distributed interactive simulation, scene generation, visualization and others. This report summarizes results of our performance tests; describes scenarios and architectures that exploit parallel processors for visual simulation applications; discusses the state of the parallel computing industry; and presents recommendations for further research and development. Furthermore, this document describes a prototype toolkit to explore various options for using parallel processors for visual simulation.

14. SUBJECT TERMS

Massively Parallel Processing, Terrain Rendering, Computer Graphics, Visual Simulation

15. NUMBER OF PAGES

44

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UNLIMITED

Table of Contents

Illustrations

Figures

Tables

Preface

This report was prepared under contract DACA76-94-C-0007 for the U.S. Army Topographic Engineering Center, Alexandria, Virginia 22315-3864 by Onyx Sciences Corporation, Cambridge, Massachusetts 02138. The Contracting Officer's Representative was Mr. John Kim.

1.0
Project Description

1.1 Problem Identification

Simulation and visualization of the modern battlefield is becoming more difficult due to the increasing sophistication of modern weapons systems, the nature of future conflicts (e.g. low-intensity, humanitarian support, etc.) and the large (and increasing) volumes of imagery and other data now available to commanders and other users. This diversity of missions and the increasing complexity and cost-per-use of modern weapons systems has lead to a recognized need for advanced simulation capabilities. To be effective training and planning devices, future military simulation systems will have to provide high-fidelity views of real or simulated battlespaces, including environmental and man-made effects.

The use of increasingly complex simulations to support weapons systems development, training, combat planning and operations is increasing and will continue to grow into the next century. Current parallel processing-based simulation/visualization systems often lack the performance required for high-fidelity simulations, due to a variety of limiting factors, including availability of modeling and graphics software, computer equipment, I/O technology and user interface restrictions. To develop the systems needed to perform high-fidelity simulations requires advances in all of the above areas. The specific focus of this project has been a study of parallel processing technologies that enable high-performance graphics and visualization.

1.2 Scope of Report

This report is organized as follows. Section 1.2 offers a problem identification and an overview of the issues addressed or not addressed by the current state-of-the-art in hardware based render-

ing systems. Section 1.3 provides a summary of potential benefits and discusses the key benefits obtained through the use of parallel processors. Section 1.4 provides an overview of a brief market analysis and highlights the technology trends in the parallel computing industry.

Section 2 focuses on a discussion of the issues of parallel processing, implementation issues relevant to parallel processing platforms and an overview of possible system architectures that readily exploit a parallel processor for distributed simulation applications. Section 3 offers examples of parallel rendering prototypes implemented on a Single Instruction Multiple Data (SIMD) platform. Section 4 offers our conclusions and recommendations for incorporating a parallel processor into the computing environment at the Topographic Engineering Center and future software development.

1.2.1 Visual Simulation and Rendering: Issues

In general, visual simulation is limited by a closely coupled combination of performance specifications: simulation speed or frame rate, image complexity or realism, and user interaction. Frame rate is a specification of how many images or frames per second can be generated and displayed to the viewer. Rates of 24-30 frames/second are used for motion pictures and video. Image realism, a more qualitative factor, is closely tied to image complexity, i.e. the more realistic an image, the more complex (or numerous) its individual components and the mathematical models used to render them. Image realism is also directly affected by the type (stereoscopic, 3D, helmet-mounted, etc.) and quality (colors, resolution, angle of view, etc.) of the display.

User interaction is how the individual handles, indicates, manipulates or otherwise interacts with objects in the computer generated environment. This interaction is usually achieved through a conventional mouse, joystick, or trackball; a 3D mouse or a glove with built-in sensors to detect hand position; or a helmet-mounted tracking device, to track head position and motion. In general, tracking head motion in a realistic manner and presenting the varying views smoothly requires a frame rate of 24-30 frames/second. The challenges in providing high frame rates with high visual quality are exacerbated when multiple participants require differing points of view and interact within the same simulation environment.

1.2.2 Traditional Pipeline Graphics Technology

Successful manufacturers of 3D graphics systems and subsystems, such as Silicon Graphics, Inc. (SGI) and Evans & Sutherland (ES), and most others in the industry, have adopted, and are limited to, the same basic rendering architecture, namely, the graphics rendering pipeline. Pipeline systems contain a number of processing elements that are arranged so the output of one is the input to the next in line. Pipelined computations are partitioned into stages that can be executed sequentially. The limiting factor in any pipeline computation is the slowest stage in the computation. Currently, the rendering pipeline is based on polygon rendering and texture mapping for added image complexity. The vendors mentioned above offer 3D graphics systems primarily based on Very Large Scale Integrated (VLSI) implementations of pipeline technology.

1.2.2.1 Performance Limits

Compromise is ever present in visual simulation and rendering systems; in particular, between the capabilities of the system, mainly its computational components, and the need to create an illusion that is sufficient to meet with the human operator's built-in expectations of reality. Currently available simulation systems, based on the graphics pipeline technology, often make severe trade-offs between image realism (complexity) and motion realism. Complex (realistic) images tend to take longer to generate and 3D workstations cannot produce and display these complex images at the required rate. Motion in these systems tends to proceed in a stop-start (non-smooth) manner, particularly when high levels of image complexity are required. On the other hand, simple scenes may be displayed at an acceptable rate and smooth motion obtained, although the cartoon or blocky look of these systems limits their usefulness and acceptance. The level of performance exhibited by these systems falls far below the public's perception of what simulation devices are capable of achieving and what a practical and useful graphical rendering system is required to achieve.

"More extensive use of parallel processing could help to improve update rates in virtual reality systems and reduce delays in reaction to operator movement, as well as adding detail and better resolution in the simulated environment", according to Duane Boman, head of the Virtual Perception Program at SRI International, [Select News 21 Jun 1993: Topic 357, HPC Wire].

1.3 Potential Benefits of MPP for Graphics

Onyx Sciences' past experience and current evaluation of multiprocessor and massively parallel processors (MPP) indicates the potential for significant performance advantages of these systems over existing specialized graphics subsystems. **Massively parallel systems can provide high-performance computing capabilities to support high-fidelity visual simulation applications that are beyond the capabilities of traditional pipelined graphics boards.** Current MPP platforms offer hardware scalability, coupled with flexible software design, which enable a long-term strategy for using massively parallel systems. Results of the Phase I research highlight several key benefits that MPP systems offer future visualization and distributed simulation needs and goals, including:

1. **Improved price/performance;**
2. **Multi-user interaction with common data-sets;**
3. **Increased visual fidelity;**
4. **General purpose rendering techniques;**
5. **Very large databases (~10^6 polygons);**
6. **Reduced time for data processing and reporting;**
7. **Supports Client/Server Computing**

1.3.1 Key Benefits

Candidate MPP systems that have the potential to surpass the performance of existing specialized graphics systems include Intel Corporation's Paragon XP/S, Cray Research Inc.'s Cray T3-D, nCUBE Corporation's nCUBE-3, MasPar Computer Corporation's MP-2, and IBM's POWER Parallel SP2. This list is confined to U.S. based vendors, even though several other foreign companies produce equally powerful systems. All of these platforms were evaluated based on several criterion during the Phase I contract.

1.3.1.1 Improved Price/Performance

SGI's top graphics subsystem is the Reality Engine^2. One Reality Engine^2 can deliver up to 1.2 GFlops peak performance. An Onyx/24 Reality Engine workstation lists at about $635,000. This works out to about $517/MFlop peak graphics performance. The various MPP platforms have peak price/performance values ranging as follows: 256-node Cray T-3D ($240/MFlop),64-node SP-2 ($318/MFlop), 256-node Paragon XP/S ($390/MFlop),128-node KSR-1 ($332/MFlop), 16k-node MP-2 ($255/MFlop) or 128-node CM-5E ($244/Mflop). With current competition, future MPP costs are expected to reach $100-$200/Mflop this decade. The Onyx/24 has additional performance, but not directly applicable to the graphics rendering pipeline. In addition, these MPP platform prices usually include peripherals such as Terabyte disk arrays and at least 4 GBytes or more total memory.

1.3.1.2 Multiuser Interaction

Current configurations of massively parallel processors support multiple users. A properly configured system could support each user with an independent point-of-view (POV), at performance levels near or exceeding those obtained from conventional graphics hardware. Alternatives include utilizing the MPP system as a dynamic environment server in a distributed simulation environment. In this mode, the system would monitor and compute man-made and natural alterations to the environment (e.g. bomb craters and terrain repair). Another mode of operation is as an animation server, handling multiple moving and articulated objects and outputting proper display lists to standard graphics systems. A medium sized MPP could theoretically handle thousands of articulated objects. Conventional graphics systems typically can handle a limited number of moving objects on the order of hundreds of objects (~250).

1.3.1.3 Increased Visual Fidelity

With much higher performance capabilities and greater memory, rendering software on MPP platforms can handle larger polygon sets, implement more computationally intensive processes such as Phong shading, shadow casting or ray-tracing, or use non-geometry based rendering methods, such as volume rendering. All of these capabilities contribute to producing more realistic simulations. As fidelity improves, simulations look more realistic, enabling users to receive quality training at drastically reduced cost. Present simulators are at times cartoon-like, have jerky or delayed response and limit the number of user inputs.

1.3.1.4 Rendering Techniques

The current specialty graphics systems, such as SGI's Reality Engine and Evans and Sutherland's Freedom Series, are firmware implementations of the polygon rendering pipeline. Polygons, however, prove to be limited in their ability to accurately model many real phenomena. This is especially true in the case of natural phenomena such as clouds, fog, haze, rain, smoke or dust storms. At best, the conventional graphics systems characterize general phenomena, such as fog or haze, in a global manner i.e. an effect applied to a complete scene. With MPP systems, on the other hand, no restrictions are placed on the rendering techniques.

On a MPP system, the rendering is defined in software, thus allowing a flexible extension to any state-of-the-art technique. Examples include volume rendering, particle methods, radiosity, terrain renderers and ray-tracers. Since the main tool is software, the correct technological tool may be enlisted to model and render those entities they describe best. For example, ray-tracing and radiosity methods will produce scientifically accurate lighting and shadows, particle methods will realistically describe smoke and clouds, while volume techniques would prove invaluable in describing model deformation, such as a bulldozer moving earth.

1.3.1.5 Very Large Databases

One of the most favorable aspects of parallel processing for graphics rendering and visual simulation is its scalability. Both the Reality Engine and Freedom Series peak out at 1.6 million and 3.5 million z-buffered, unlighted, flat-shaded, 50-pixel, meshed, random oriented triangular polygons per second, respectively. Since the hardware is a pipeline, adding additional graphics pipelines will not improve system performance. An MPP platform, on the other hand, can increase throughput by adding processors to the system. Naturally, the rendering software must be designed with this in mind.

As an example, a naive implementation of a polygon renderer on a 256-node Cray T-3D recently achieved 4.5 million non-anti-aliased, z-buffered, lighted, flat-shaded, 50-pixel, independent, randomly oriented triangles per second. Scaling this system to a possible 2048 nodes, the increase in performance can theoretically reach 36 million triangles per second, depending on the software methodology.

1.3.1.6 Client/Server Model

The client-server model of computing is, in many cases, substantially enhanced by the use of parallel processors. The use of a parallel processor in a client/server environment enables a flexible approach to accessing a powerful computing resource. In the client server model, a single large parallel processor is seen as a powerful resource accessible to many users on the network. However, configurations containing a number of smaller MPP platforms fits a more distributed architecture that still fits within the client server model. That is, each individual MPP provides a computing resource, although to a smaller number of users. This flexibility allows system designers to make trade-offs to avoid bottlenecks, while providing more computing resources to end users than would normally be available in a fully distributed architecture.

1.4 Marketplace and Technology Trends

The introduction of parallel machines was motivated by the potential for incredible performance gains obtained from linking-up multiple processors. Machine designers have explored a large number of exotic and esoteric architectures and implementation strategies for parallel platforms. In general, these systems reflect the particular designer's own bias toward emphasizing the importance of certain types of functions. The result is that the machines built were very good at performing certain operations, but often performed very poorly on others. This imbalance was also a result of uneven technological progress in various aspects of parallel processing (computation, communication, I/O) in both hardware and software. The development of a simple, accurate parallel model of performance characteristics was frustrated by the wide variety of machines available, both research and commercial. Despite this wide variety, a number of technological and market factors are driving the design of current and future parallel platforms along a common path. Thus, the current situation in parallel computing indicates a trend in parallel system design.

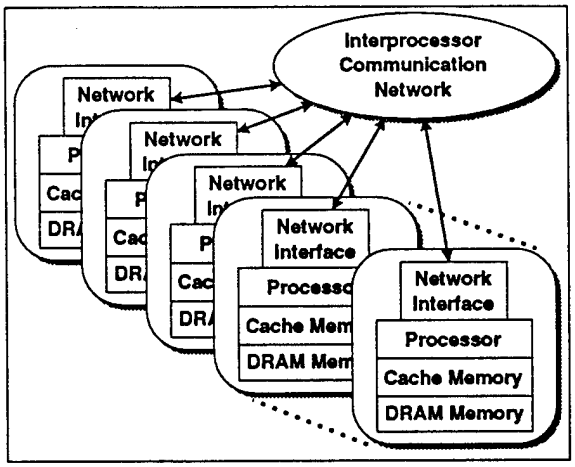

FIGURE 1-1. Schematic of a Generic Parallel System

1.4.1 Processor and Memory Technology

Bell[1] and others predict the majority of parallel platforms will be similar in form as the field matures. Figure 1-1 shows a schematic of the typical parallel processor. These systems will contain tens to thousands of processing nodes, each consisting of a microprocessor, cache memory and Dynamic Random Access Memory (DRAM), linked by an interconnection network. The main reason for this trend is the increasing use of commodity processors to build parallel machines.

This trend is a function of various technology and market pressures. First, microprocessor performance has been increasing at a rate of 50 to 100% per year over the past decade. In fact, Bell

1. Gordon Bell, "Scalable, Parallel Computers: Alternatives, Issues and Challenges", International Journal of Parallel Programming, Vol. 22, No. 1, 1994.

predicts a 60% yearly improvement in microprocessors, representing a quadrupling of performance every three years. Current leading edge processors include Digital's Alpha chip, which recently operated at 300MHz and achieved 1 billion operations/sec, and IBM's Power2 chip, rated at 266 Mflop/s. These gains are made at an enormous cost, approximately 100 man years and $40 million for development and initial fabrication. However, the manufacturers of these chips benefit from the economies of scale made possible by the large market for microprocessors. It has become clear to parallel vendors that in order to offer viable products, they must follow the same technology growth curve. Due to this, many parallel vendors rely on the use of commodity chips in their MPP platforms. Table 1-1 lists several vendors and the corresponding microprocessor.

DRAM capacities have been quadrupling every three years, indicating a thousand-fold increase over 15 years, a million-fold in 30 years. The memory capacity of available systems has followed a similar curve. Typical PC and workstations use 8MB and 32MB, respectively. These are based on the currently available 4Mb DRAM chips. Predictions indicate that by 1997 256Mb DRAM chips will be available, offering 64 times the memory capacity for the same number of DRAM. In fact, 16Mb DRAMs are already being used in prototype and commercial devices, such as Cray's C90 vector supercomputer. It must be noted that processor cycle times and memory access times continue to diverge. Therefore, complex cache structures will have to be incorporated into DRAM technology. Again, parallel processors will have to follow this technology curve and incorporate state-of-the-art memory systems to remain viable. Most parallel systems will scale to only thousands of processors, since the cost of millions of processors will be prohibitively large for the foreseeable future. The largest Multiple Instruction Multiple Data (MIMD) systems currently offered have no more than 2000 nodes.

TABLE 1-1. Parallel Platforms and Microprocessors

Vendor	Platform	Processor	Mflop/s per node (64bit)
TMC	CM5	SPARC	160[2]
IBM	SP2	RIOS-2	266
Cray	T3D	DEC Alpha	150
Meiko	CS-2	SPARC	100[3]
Convex	SPP	PA-RISC	200
nCUBE	2S	Custom	3.0
KSR	KSR2	Custom	80
MasPar	MP-2 (SIMD)	Custom	0.39

In conclusion, technological and market trends point toward the availability of parallel machines composed of processors capable of hundreds to thousands of Mflop/s with hundreds to thousands of MB of memory. These systems will scale to thousands of nodes, not millions. This trend has implications for the assumptions applied to parallel algorithm development. Namely, algorithms will require a large number of data elements per processor.

2. This includes the performance of the vector processors attached to each node.

3. This includes 2 vector processors per node. The SPARC processor provides 50Mflops peak performance.

1.4.1.1 Network Technology

The other component in parallel systems is the interprocessor communication network. Unfortunately, the same market forces fueling advances in memory and processor technology do not apply to network technology. The communication bandwidth severely lags the processor to memory bandwidth. This makes remote references far more costly than local references. Even though high bandwidth media are available (e.g. fiber optics), a major bottleneck still exists in the interface between the network and processing nodes. In general, commodity processors are not designed for this purpose and cannot efficiently connect to a network, which incurs substantial overhead costs. Therefore, parallel machines will continue to suffer from high latency and overhead, as well as limited bandwidth.

There are a number of different network topologies, such as mesh, hypercube, array, ring and switches. Implementations of these topologies can be found in a variety of commercial and research machines. Clearly, there is no general agreement on network topology and each vendor has their own reasons for their particular design choice. Some reasons for selecting a particular network include, ease of implementation (hardware and software), reduced network diameter, and increased bi-section bandwidth. In addition to specifying a topology, most manufacturers install fault tolerance mechanisms to ensure production applications continue execution when a system component is unavailable. For example, most machines have extra nodes that take over for a network node that fails. Unfortunately, these nodes are usually not in the network topology, and can not maintain the same performance characteristics as the original node. This has great impact on the performance of existing routing schemes, however, adaptive routing schemes are helping address the general variations in machine topology.

Typically, a parallel programmer will exploit the network topology to increase performance. For example, a programmer may use an algorithm requiring only nearest neighbor communication on a mesh. However, algorithms that exploit a particular topology are not robust in practice, incurring performance penalties when implemented on different topologies.

1.4.1.2 Programming Styles

Lastly, most available machines support a combination of programming methodologies, namely shared-memory, message passing and/or data parallel. Each style has its own requirements and places unique demands on the underlying hardware and programmer. In terms of usage, each style has achieved a certain amount of popularity and applicability, and as yet, there is no clearly dominant style. Regardless of programming style, a parallel programmer must contend with computation, communication and I/O.

1.5 Summary

With the past and current trends in parallel technologies (processors, networks and I/O), MPP systems are merging toward a MIMD environment based on finite numbers of processors with ever increasing performance, memories, storage and bandwidth. These indicate that MPP systems used in client/server mode for such applications as database mining, video-on-demand and

multi-user interactive gaming, virtual reality and simulation will be evident before the end of the decade. One key to this end goal is the continuing decrease in overall price/performance of future systems. As commodity processors increase in power and decrease in price, MPP systems will become more affordable to general businesses (mainly for use in database applications) and in turn will spurn more robust software to enhance current capabilities.

In addition to price/performance benefits of the current and pending MPP systems, processing performance increases alone will deliver the greatest advances for the U.S. Army's future needs in multi-user simulation and visualization requirements. MPP platforms have the potential to control a complete time-varying battlefield database of personnel, equipment and natural terrain and environment for simulation feeds to hundreds of users. In addition, they may serve as generators of physically-based models of weather, dynamic terrain and other realistic effects such as smoke and fire, which are computationally exhaustive on current workstation technology. Finally, the computational potential of MPP platforms may be harnessed more efficiently through software. Instead of relying on polygon rendering for all scene components, more appropriate techniques may be utilized to generate components such as smoke, weather and fire. It should be emphasized that MPP systems still have the versatility to render polygon databases, and as was shown, gain advantages over the pipelined systems through their ability to handle much larger datasets.

Future parallel platforms will primarily consist of up to thousands of nodes, at most, each containing large memory and powerful processors, connected by a low latency, high bandwidth network. Typical configurations will be in the 8-256 processor range. In addition, these machines will tend to lean towards a MIMD model rather than the SIMD paradigm, as embodied in machines such as the MasPar MP-2. Message-passing and distributed shared memory models will be the dominant form among scalable systems. However, symmetric multiprocessors (SMP) will continue to be in wide spread use for a number of applications. For use in a distributed simulation system, the scalable platforms offer more flexibility, while the SMP platforms may provide price/performance advantages for specific applications.

2.0
Rendering on Massively Parallel Processors

2.1 Introduction

Much research effort has been spent in recent years developing graphics algorithms and techniques for use on parallel architectures. One of the advantages of parallel machines for graphics lies in their flexibility, i.e. many rendering methods may be developed. In the following we discuss volume rendering methods on parallel machines. However, the greatest potential of parallel machines lie in their ability to provide enhanced computing power to more than the rendering aspects of visual simulations. Parallel platforms can be utilized for many other tasks required for distributed simulation, e.g. modeling, communication processing and non-visual computations.

This section covers the key issues of parallel processing and rendering on MPP platforms. Section 2.2 covers the general issues of parallel processing and distributed computing. In Section 2.3 we discuss these issues in the context of parallel rendering algorithms. We also provide overviews of three parallel rendering algorithms, implemented on the MasPar MP-1/MP-2 platforms.

2.2 Parallel Processing Issues

There are several important factors and concepts that must be considered when implementing software on a parallel processing or distributed computing platform. These can be broadly categorized as computation, communication and I/O. This section discusses several issues falling under these three categories and include:

- Computation: load balancing, virtual processor ratio;
- Communication: latency, bandwidth, global communication, grid communication;

- I/O: display, disk access, network access

When implementing software on parallel platforms, these issues cannot be considered in isolation, but must be considered as a whole. Trade-offs in one area, can adversely affect performance in others. For example, on many massively parallel platforms there exists a particular approach for I/O that is significantly faster than other available techniques. In many cases, the performance penalties for using other patterns for I/O data layout outweigh any potential performance gains in the computational or communication portions of the program. Below we discuss only computation and communication in detail; we will cover I/O issues specific to rendering in later sections.

2.2.1 Computation

The main factors affecting computational performance of a parallel processor is individual processor performance and the number of processors. In general, the faster the component processors, the higher potential performance of the parallel platform. However, we note that parallel platforms with low-performance individual processors can exhibit very high performance in aggregate. The highest performance is achieved when all processors are working at or near maximum. Two generic factors can affect how much work is required for each processor, virtual processor ratio and load balancing.

2.2.1.1 Virtual Processors

Many software programs developed for parallel processors utilize the notion of a virtual processor (VP). Simply put, each physical processor simulates some larger number of virtual processors. On any given physical processors all virtual processors act independently. The VP paradigm is executed through a simple looping mechanism and at no time is information exchanged between virtual layers in the physical processor. The virtual processor ratio (vpr) is an important figure, and is the ratio of the number of virtual processors versus the number of physical processors, vpr = v/p. For example, a vpr of 256 means that each physical processor simulates 256 processors; on a 4096 processor machine a vpr of 256 translates into 1,048,576 simulated processors. On some parallel machines, such as the Connection Machine CM-2, the virtual processor mechanism is built-in and a programmer need not implement the loops explicitly. On the MasPar MP-2 there is no built-in virtual processor mechanism and any *virtualization* or vp looping must be explicitly handled by the programmer. While in some cases this increases the difficulty of the programmers job, the extra control it gives the programmer is often rewarded by increased performance.

Strategic use of virtualization can help ensure that each processor is operated at or near peak performance. However, all processors executing at peak performance without duplication of computations is usually impossible. When a processor has to stop to get data, or if it has completed its computation and is waiting for other processors to complete, its performance capabilities are wasted. Interactive graphics places additional constraints on computational performance. For example, too large a vpr will adversely affect the interactive performance by requiring too much

computation; while too low a vpr may incur performance penalties due to increased interprocessor communication requirements.

2.2.1.2 Load Balancing

Load balancing is a major issue when dealing with multiprocessor and massively parallel systems. A poorly load balanced system is characterized by having a few processors heavily loaded, while most of the others are lightly loaded or even idle. Load balancing can be of particular concern when dealing with SIMD platforms, where thousands of processors work in lockstep and execute the same instruction simultaneously. Load balancing is often implemented in software through various computational techniques, and is an attempt to avoid performance penalties by keeping all processors working. However, proper balance must be maintained, since benefits of load balancing can be outweighed by communication penalties incurred when too much data must be moved around to keep all processors busy.

During each of these data movement procedures, the processors are usually idle, but performance improvements in other portions of the execution may make up for this loss. The decision to load balance or not is a key issue in any implementation. It should be noted that some adaptive load balancing techniques may use up too much time and degrade overall system performance.

2.2.2 Communication

Another programming issue in parallel systems is interprocessor communication. In any massively parallel or distributed computing architecture there is a system of processor interconnects that enable communication for both control and data. Interconnects may be based on switches, buses or uniform networks. Fixed-connection networks are designed to enhance local processor communications, while switches and buses are primarily designed to improve global communication[1]. Fixed-connection networks for interprocessor communication come in many different types, such as arrays or meshes, trees, hypercubes, butterflies or various combinations of these, i.e. meshes of trees. These interconnects play a major role in determining the overall system performance and scalability. The processors in an MPP system are typically close-coupled using tightly integrated connections (networks or switches) for processor communications.

Over the past few years the differences in per-processor performance of most available parallel systems has evened out. This trend arose from the vendor's increasing use of commodity processors (for cost effectiveness and convenience), and has allowed vendors to benefit from the large amount of experience in design and code optimization already invested in serial processors. This has somewhat removed the MFlops (millions of floating point operations per second) per processor figure as a true source of comparison between platforms. Overall performance now depends on the specific interconnection system and communication paradigm used in each machine, in addition to the computational power of the individual processors. Thus, interprocessor communi-

1. As mentioned, SIMD platforms often support two communication networks, one fixed for fast local communication and the other switch or bus based for global communication.

cation strategies represent a major source of performance differences between today's parallel systems. Latency and bandwidth are two communication metrics which can be useful in quantifying these differences. In general, however, raw metrics such as these can be misleading and difficult to interpret.

2.2.2.1 Latency and Latency Hiding

Latency is the delay caused by communicating between processors or between processors and memory over a network. The standard definition of latency is the amount of time required for one processing node to open a connection to a remote node, regardless of data size. It is usually measured in micro- or milliseconds. Figures often quoted by vendors are hardware-only (switch) latency. This being the latency inherent in whatever hardware switch is used in a particular machine. In the real world, latency may be affected (increased) by various layers of software at the sending and receiving end. Latency can often be amortized by sending larger amounts of data or messages. The larger the message, the smaller a fraction of transfer time is consumed by latency.

Latency is inherent in any parallel system, but takes on different traits when measured in a shared memory system vs. a distributed memory system. Many vendors offering virtual shared memory platforms provide features to minimize the effect of latency. These are known as "latency hiding" techniques. Some common techniques for latency hiding include: data pre-fetch, multithreading, pipelined communication or caching strategies. One basic problem with distributed memory machines is that interprocessor communication takes a finite non-zero amount of time. This is in contrast to the shared-memory programming model, which implies instantaneous data movement. The following are examples of common latency hiding techniques:

- Data pre-fetch is a technique for accessing data before it is needed.
- Multithreading hides latency by switching the processor to another task while it is waiting for a data transfer.
- Pipelined communication groups data into large packets and moves them in a single transfer.
- Caching strategies maintain local duplicates of memory blocks or move memory blocks closer to processors that need them.

Many of these techniques are of limited effectiveness in general purpose situations. However, when application parameters are known, programmers can implement similar methods to produce more efficient parallel code. This assumes that the overhead of the additional software is more than made-up for in efficiency of the actual computation. In general, if the computation-communication ratio is high, then latency hiding will have little effect. Like load balancing, if the cost of computing the load balanced situation increases execution time, it is hardly worth the effort. Variations on the above techniques are used in the three MPP rendering examples described in Section 2.4. Some of these techniques (e.g. multithreading) are inappropriate for SIMD platforms.

2.2.2.2 Bandwidth

The amount of data that can be transferred per unit time, once a connection is opened, is called the *bandwidth*, and is usually measured in MB per second. The numbers often quoted by vendors are total system bandwidth. Most often these figures have little or no meaning to the end-user, since a number of additional real-world factors affect both latency and bandwidth. For example, messages may be blocked while communication channels are being used by other messages[2]; data needs to be chopped into fixed size packets; and application-specific messaging style and message length requirements.

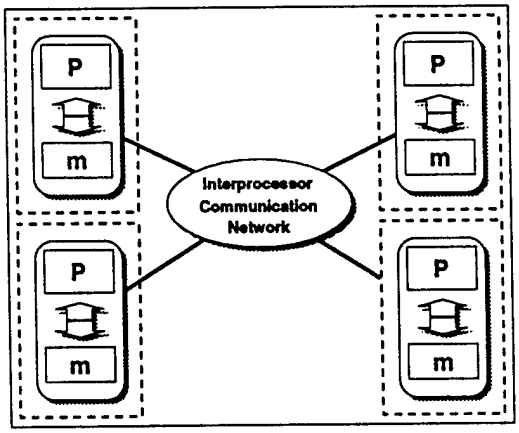

FIGURE 2-1. Total System Bandwidth

To illustrate the concept of total system bandwidth, consider a generic distributed memory architecture, where each processor/memory module is enclosed in a box[3] (Figure 2-1). The total system bandwidth is one-half the total amount of data that can cross all boundaries per unit time. For example, assume the bandwidth into each node is 50MB/second, then total system bandwidth would be 100MB/second. Generally, the total bandwidth is proportional to the number of processors. Thus, total bandwidth is a scalable metric independent of network shape.

Bi-sectional bandwidth is another useful measure for determining machine-algorithm performance potential. It is a measure of aggregate bandwidth for parallel machines. Bi-sectional bandwidth can be defined as follows: define a plane which separates the parallel system into two parts containing an equal number of nodes. This plane intersects some minimum number of network links. Bi-sectional bandwidth is the total possible bandwidth crossing this plane through these links. Bi-sectional bandwidth is a measure of the random communication performance of a given parallel platform. That is, for any general and random communication scheme, one half the messages from one side of the machine will have to travel to the other side. The total amount of data moving from one half to the other is limited by the bi-section bandwidth.

---

2. A form of resource contention.
3. This box should include all system resources dedicated to that node.

Effective bandwidth is the realized bandwidth for a given application. It is a much more useful figure than total bandwidth and it tends to increase as the size of the data transfer grows. This indicates that message length is important in maintaining communication efficiency. The communication efficiency is a source of distinct differences between shared memory and message-passing models. Shared-memory methods tend to pass short (~10 bytes) messages which are packed with ancillary data, such as addresses, error correcting codes, etc. For short messages, a large fraction of the system's communication bandwidth is used on addressing and controlling a large number of messages. Often the overhead-data ratio can reach or exceed 50%. Message-passing systems typically send long (>100 Bytes) messages with low data-overhead ratios. In general, message-passing architectures allow for more efficient interprocessor communication, since messages tend to be longer and overhead information (i.e. addresses) is a smaller fraction of the total packet size. Communication efficiency for shared memory systems, with blocking, typically approach 30%, while message passing systems approach 50%.

2.2.2.3 General and Grid Communication

On most parallel platforms there exist at least two communication mechanisms. The first allows any-to-any (many) communication patterns; the second allows communication on a predefined grid. The example of a MasPar MP-2 is used for illustrative purposes.

The MasPar MP-2 is scalable from 1024 to 16384 processors connected in a 2-D mesh with wrap-around (2D torus). Processors are 8-way connected, with normal mesh connections (north-south-east-west), plus diagonal connections (NE,NW,SE,SW). Communication on the mesh is supported by an independent communication network called Xnet. Global or general communications are supported by a router. There is one router wire for every 16 processors (a cluster), in general, Xnet is faster than router communication, with approximately 16 times the bandwidth of the router. Xnet produces approximately 1.22MB/s peak bandwidth per processor and 22.2GB/s aggregate bandwidth on a 16,384 processor platform. However, all active processors must apply (or ignore) the same Xnet communication path simultaneously. The router has 1.3GB/s aggregate bandwidth on the same size system. This gives a per cluster bandwidth of 1.27MB/s, about 1/16 the bandwidth of Xnet since there is one router connection per cluster of 16 processors. As can be seen from this example grid (Xnet) communication has superior performance if it can be exploited, while general communications (router) can be costly for large amounts of data. Specific issues related to rendering implementations will be addressed in later sections.

2.2.3 Summary: Computation and Communication

The issues of parallel processing are varied and range from physical architecture designs to software implementations. In general, the potential advantages of most parallel systems and applications will depend on how well latency and bandwidth are dealt with by software designers, and how a programmer handles load balancing. The ultimate performance that may be squeezed-out of an MPP system relies on all aspects of communication, computation and I/O.

2.3 MPP Rendering

Parallel rendering has been an active research topic for many years. Spurred by the increasing use of parallel platforms for large-scale scientific calculations, the advantages of parallel machines for graphics production to support these calculations has become the focus of research groups around the world. Most of these groups have focused on scientific visualization, which impose slightly different constraints than fully interactive visual simulation applications. However, the key issues of parallel rendering remain the same, whether applied to scientific visualization or flight simulation. As above, for general parallel software development, the following are key issues to consider when developing parallel rendering algorithms:

1. exploiting parallelism
2. memory limitations
3. data distribution and ordering
4. communication
5. load balancing
6. image display

2.3.1 Exploiting Parallelism

The performance advantages of parallel platforms can only be obtained if we can find and exploit parallelism in our rendering algorithms. For massively parallel systems we typically exploit *data parallelism.* Data parallel computations take place on multiple data items simultaneously, and typically achieve much higher levels of parallelism than control parallel methods.

In rendering algorithms there are two main types of data parallelism: object parallelism and image parallelism. Object parallelism usually refers to operations which are performed on the primitives which make up the objects in the scene. These can be polygons, voxels, or any other primitives (e.g., spheres) suitable for describing the components of a scene. In the case of polygons, these operations can include coordinate transformations, backface culling, clipping and lighting calculations. These operations can be performed independently on each primitive, and require little or no interprocessor communication. Since the scene may contain up to millions of primitives, object parallel computations exploit data parallelism very well.

Image parallel computations take place on a pixel level and depending on the algorithm can occur early or later in the rendering process. For example, in a polygon rendering algorithm it would occur later, since pixel level calculations don't occur until rasterization. In theory, pixel values can be computed independently, but efficient interpolation methods require that pixels be computed in groups over a primitive or subset of the primitive. However, complex scenes will contain many rasterization tasks and also exploit data parallelism. Other rendering methods, such as image-based terrain rendering exploit image parallelism from the outset. This is primarily due to the raster-based format of the input data.

2.3.2 Memory Limitations

The fact that the object dataset, output image, program instructions, and possibly, application software must simultaneously reside in memory, parallel rendering algorithms must avoid consuming too many resources, either for data or instructions. In many case, the restrictions on being memory efficient conflict with the requirements to produce very fast rendering code. The memory restrictions on SIMD platforms are often quite severe. The rendering software discussed later in this report, was written on MasPar MP-1 with only 16K bytes of memory per node.

2.3.3 Data Distribution and Ordering

In general, rendering algorithms use two large data structures: the objects to be rendered and the image buffer to hold the final image. In some cases there are intermediate data structures, but the existence of these depend on the particular algorithm and implementation. The size of the object database is dependent on the scene, but how these are laid out across processors is dependent on the rendering technique and algorithm used. Often the simplest method is to spread the primitives across the processors more or less evenly, without regard to primitive ordering. However, this can often lead to poorly load balanced situations, as discussed below.

The image buffer stores the pixel data for the output image and may include, as a minimum, color and depth information. Typically, the image buffer is partitioned a spread across all the processors. Many arrangements are possible and can be in scanline strips, interleaved scanlines, scanline segments, rectangular blocks, or even adaptive decompositions. In many cases, the distribution scheme used for object data and image data can effect the scalability of the rendering algorithm, making it easier or more difficult to handle more complex scenes or higher image resolutions. Also, the ordering of the image buffer, i.e. how it is mapped to processors, often has a major impact on I/O performance. Especially in SIMD platforms, which may have an optimal ordering to achieve the fastest I/O possible. In our implementation of a terrain renderer, discussed below, this was the case.

2.3.4 Load Balancing

Interactive graphics place special constraints on parallel machines and virtual processor ratios, and data distribution becomes increasingly important. Namely, careful attention must be paid to how much computation is required in each processor. In general, real-world scenes are not uniform and this can cause load imbalances. The most obvious distribution approaches result in poor load balancing, since the time to scan convert a polygon is determined by its size and shape. Since there is no practical way of knowing the size and shape of each primitive beforehand, some processors may end up with significantly more work than others. Other simplistic schemes, such as mapping scanlines to processors, are subject to the same problem.

The image parallel portion of the computation may also be subject to load imbalances, since the distribution of primitives can be localized in image space as well. For example, sharp curves, such as those that occur at a wing tip, require a larger number of polygons to describe. This means that a majority of the rasterization steps occur in a small percentage of the image space

and the processors responsible for these pixels have more work to complete than other processors. This can also lead to imbalances in communications.

2.3.5 Communication

The selection of a rendering algorithm typically has a major impact on communication costs incurred for a given MPP platform. Unless each node on the MPP has a local copy of the entire data set, or viewing positions are severely restricted (for certain rendering methods), any rendering algorithm will intrinsically require some form of interprocessor communication. In general, replicating a data set at each node is impractical for large numbers of nodes and impossible for large data sets that may exceed the local memory size. In shared memory machines, bottlenecks in memory access may impose other limitations.

The rendering process can be viewed as a mapping of three-dimensional world space to two-dimensional screen space. This mapping process invariably requires interprocessor communication to map between the distinct data orderings. In polygon rendering this is often referred to as a sorting step. It can result in massive amounts of communication between processors as primitives are moved to the processors corresponding image pixels. Molnar et. al.[4] have developed a taxonomy based on where in the rendering pipeline the communication occurs. They use three classes of algorithms, *sort-first*, *sort-middle*, and *sort-last*. Sort-first algorithms map primitives to their final destination early in the rendering process, i.e. polygons are assigned to the processor that has the corresponding pixels. Sort-middle algorithms place a communication step in-between the rasterization and transformation phases. Finally, sort-last methods send pixels to their final destination for output late in the rasterization process. The approach used is dependent on the architecture and application considerations. Sort-last approaches avoid computational overheads which occur when primitives must be split before rasterization. However, these methods typically require high-bandwidth to be effective.

The process of communicating can often be the costliest aspect of a rendering algorithm. In fact, the interprocessor communication is often the limiting factor in achieving high rendering and display rates. Namely, the communication bandwidth and latency limit the amount of data that can be moved in a given amount of time. We often say that an algorithm is communication-bound, which means that performance will never exceed a maximum limit set by the communication performance of the parallel platform.

2.3.6 Scalability

A parallel system's usefulness is often given in terms of its *scalability*. Scalability is that which measures the capacity of a parallel machine-algorithm combination to solve a problem of increasing size with an increasing number of processors. The basic assumption being that the parallelism of a given algorithm has already been effectively exploited. Scalability is mainly used to predict the performance of large problems on large machines based on the performance

4. S. Molnar, M. Cox, D. Ellsworth and H. Fuchs, "A Sorting Classification of Parallel Rendering", IEEE Computer Graphics and Applications 14, 4 July 1994.

of small problems on small machines. It can also be used to help determine the "best" parallel algorithm for a given platform. This is considerably harder than determining the "best" sequential algorithm, which is usually the fastest one. For example, a fixed number of processors can solve a fixed-size problem in a given amount of time, but altering either parameter can adversely affect the application's performance, making algorithm selection more complicated. There are a number of scalability metrics, such as isospeed and isoefficiency[5].

Scalability is generally given two different, yet related, definitions. The first is in terms of the parallel computer hardware. A scalable parallel machine is one in which increasing the number of processors does not create system imbalances or bottlenecks. The type of processor interconnect and memory model used in a particular machine are determining factors in a particular machine's scalability. The second definition is based on scalability of parallel algorithms or applications. Since the scalability of a parallel algorithm cannot be evaluated independent of the architecture on which it is implemented, this measure is related to hardware scalability. Thus, in the second view, we speak of the scalability of a parallel algorithm-machine combination (parallel system). The following will focus on the scalability of parallel systems.

In general, both load imbalance and communication costs limit scalability. Communication costs is the more serious issue, since it can effect load balancing schemes. Crockett and Orloff[6] have shown, that given a uniform load, latency involved in communicating places fundamental limits on the scalability of rendering algorithms which distribute both image and object data. This is mainly due to communication patterns that are all-to-many or all-to-all, meaning the number of messages handled by each processor, and the corresponding message overhead, increases with the number of processors in the system. The latency hiding techniques mentioned in Section 2.2.2.1 can often be used to reduce message overhead and improve scalability.

2.3.7 Image Display: I/O

The data distribution methods for image buffers, mentioned above, may enable improved rendering performance, however, they can pose severe problems when attempting to display results. In order to display the final image, individual portions of the image must be collected from each processor and passed to an external display device. Preferably, this output can occur in parallel, since a serial process would impose a severe bottleneck. Two basic output forms are possible: output to a file or output to a display.

Consider the case of display for a 1024x1024 24 bit RGB image at 30 frames per second. This requires an I/O bandwidth to the display device (e.g. HiPPI framebuffer) of approximately 90MB/s. While the peak internal bandwidth of a few parallel platforms meets or exceeds this limit, the problem is trying to orchestrate the rendering and data flow to operate the I/O channel at these speeds. If the images are being sent to a front-end or remote workstation, the bandwidth

5. A. Y. Grama, A. Gupta and V. Kumar, "Isoefficiency: Measuring the Scalability of Parallel Algorithms and Architectures", IEEE Parallel and Distributed Technology, August 1993.

6. T. Crockett, T. Orloff, "A Parallel Rendering Algorithm for MIMD Architectures", ICASE Report 91-3 (NASA CR 187571), ICASE June 1991.

limitations are more severe. In our example terrain renderer, the display rate was limited by the bandwidth from the parallel array to the workstation front-end.

In many cases, we note that parallel machines have not been constructed for the purpose of outputting images at high frame rates. And while the bandwidth available on current mature MPP platforms will be high, developing rendering algorithms that can sustain the high I/O rates require skilled programming.

2.4 Summary

Many issues must be addressed simultaneously when developing parallel rendering algorithms and tools. Parallel rendering is still an active research area and will remain so, as long as there exist a multitude of parallel architectures. In general, parallel rendering performance depends on a particular algorithm-machine combination. Each possible parallel architecture provides advantages and disadvantages. The requirements of a parallel rendering toolkit or library are driven by the properties of the applications it is required to support, as well as the target architectures. Practical and successful parallel rendering algorithms must achieve a balance between conflicting goals and requirements. Load balancing, communication overhead and image output are the main areas requiring particular attention in order to achieve scalable performance.

3.0
Parallel Rendering Examples

3.1 Introduction

This section discusses selected results from rendering tests on massively parallel platforms. Examples from ray tracing, polygon rendering and terrain rendering are discussed. The system architecture, both hardware and support software, are discussed in Appendix A. Reference to the issues discussed in Section 2, such as load balancing and interprocessor communication are addressed. This Chapter is organized as follows: Section 3.2 gives an overview of terrain rendering, while Section 3.3 provides a discussion of Onyx's prototype MPP terrain rendering system. In Section 3.4 we describe a simple parallel ray tracing implementation. Section 3.5 discusses two polygon rendering implementations, one for SIMD platforms, the other for a MIMD platforms.

3.1.1 Onyx's In-house MasPar System

We provide a full description of the features of the MasPar platform in Appendix A, while here we give a brief overview of the system configuration used in development of the prototype software. The in-house system was a 4096 processor MP-1, with a DecStation front-end. Each processor had 16Kb of local memory. The system had no external disk storage other than what existed on the Unix workstation. In addition, there were no special purpose display devices available. All software was written in standard ANSI C or MPL (MasPar's parallel extension to ANSI C). The MasPar Programing Environment was used in debugging the code, however, no other standard MasPar Libraries were utilized. For instance, although convenient, we found the Data Display Library (MPDDL) too slow to support the required frame rates. In several cases we wrote our own specially designed routines for image display or data communications.

3.1.1.1 Basic tools

In developing high-performance rendering code for parallel machines, we generally find there are a number of calculations or functions that must be performed regularly. In particular, these include functions that dice up polygons and lines, shading functions, coordinate transformations, and image display routines (e.g. dithering), among others.

There are also a number of generic tools developed for the MasPar that have nothing to do with specific algorithms or applications at all, but merely provide support for writing programs such as timing functions, useful macros, display tools, user interface tools, and general C programming tools. Some tools specific to the MasPar were developed that primarily deal with custom routing or communications mechanisms that are not supported in the standard MasPar languages, libraries or macros. For instance, fast macros were developed for computing physical processor addresses, which are otherwise unavailable in certain useful forms or use slower mechanisms provided by MasPar.

3.1.2 Programming Techniques Employed

Considerable programming effort was required to extract the required level of performance out of the system, particularly for the terrain renderer. Below, we highlight some techniques used to implement Perigee. We note that many of these methods were required due to limited memory and the necessity of storing the input image on the machine in the same memory space. This meant that not only did we have to put approximately 48MB of image and height data into memory, but also the output image buffer, sample window buffer, program instructions and all temporary stack space had to fit in 64MB of memory.

3.1.2.1 Constants versus Variables

Multiple macros, based on machine size, were developed; this was to ensure that certain statements compiled to constant values rather than variables. Wherever possible, constants were used instead of computed values. Other macros which compute various processor specific values were employed to ensure efficient data movement via interprocessor communication.

3.1.2.2 Automatic Code Generation

Loops were unwrapped to eliminate the overhead of computing loop variables. Since many similar statements or lines of code were required, we developed a method of automatic code generation. For example, to implement a particularly long sequence of statements with very similar form, we wrote a program to automatically write out the 256 (or more) variations on the basic code fragment.

3.1.2.3 Interprocessor Communications

In performing communications, we used the grid communications network (Xnet) wherever possible, since the router is 16 times slower. However, in certain cases, the simplicity of using the router outweighed any possible performance advantages. Again router calls were used only when absolutely necessary.

3.1.2.4 I/O Methods

The implementation of Perigee utilized an asynchronous method for image output. This required us to write the output image to I/O RAM in the fastest possible manner. We used a processor ordering called 1D hierarchical in the MasPar manuals. This processor order assigns vpr pixels in a scanline to a single processor, where vpr is the virtual processor ratio. For example, assume a hypothetical array of 4 processors (PE0, PE1, PE2, PE3) and 16 data elements. In a 1D hierarchical virtualization scheme for this example there is a VP ratio of 4 and PE0 gets data elements 1-4, PE1 gets 5-8, PE3 gets 9-12 and PE3 gets 13-16. This data layout is the fastest possible output format on the MP-1 and MP-2. We note that higher bandwidth is achieved with higher VP ratios. One side effect of using this pixel to processor mapping was that it required placing the display monitor on its side to see the output image upright.

There was also a performance problem using the XWindows display system running on the Unix Front-end. In order to achieve maximum display speed for the terrain rendering demo, we had to disable the XWindows software and write our own display routines that placed the output image directly into the framebuffer of the Front-end workstation.

3.2 Terrain Rendering

Much research effort has been spent in recent years developing graphics algorithms and techniques for use on parallel architectures. In this section we discuss performance of an algorithm for real-time terrain rendering on a massively parallel computer. The underlying concept is not specific to terrain rendering and is being developed for use in other rendering techniques such as polygon, volume and particle rendering, as well as ray tracing.

Perigee, Onyx's proprietary software, is a real-time terrain rendering system developed on the MasPar family of massively parallel processors. Initial development was done on an MP-1 platform, but additional tests were performed on an MP-2. Real-time animation is achieved by exploiting temporal coherence and maximizing the interprocessor communication versus computation trade-off. This enables pre-computation and avoids the requirement for full computations to render each frame. Spatial coherence is exploited since we are using a scan-line method for computation at one point.

3.2.1 Methods of Terrain Rendering

Terrain rendering is a central component of image generation systems used in flight simulators and ground vehicle simulators. More recently, terrain rendering has become an important component in GIS systems, where the ability to render perspective views of terrain allows for intuitive interpretation in analysis and planning.

Surface views of terrain are generated using forward or reverse techniques. In forward projection, the mapping takes points in the input image and projects them into the 2D output view plane. In reverse projection, every point in the output view plane is determined by following rays into the input domain, traversing all required objects or elements to test for intersection. For

large data sets, access to data elements can dominate processing time for determining visibility using reverse techniques. Note that resampling is generally required for any geometrical transformation of data. Reverse projection allows correct resampling and associated filtering within the limits of physical constraints; forward projection, on the other hand, presents interpolation problems that, in general, require approximations.

3.2.1.1 Projection Methods

Computation of perspective projections can be performed independently for each point in the input surface height field, although only a subset of the projected points will be visible in the output window. This makes projection a highly parallel operation, essentially to the order of the resolution of the image. The main goal of developing parallel terrain rendering algorithms is to maximally exploit the necessary trade-offs in computation and communication costs to provide superior performance; either in frame rate, image quality or both.

As stated above, forward projection maps a point from the 2.5D input field (terrain data set) to a 2D output field (image). This output field is non-unique, since multiple input points may map to the same output point. Forward projection presents interpolation problems that are not necessarily solvable, and may require approximations that produce artifacts in the output image. For example, holes in the output view plane may arise if no input points are mapped to specific output points; due to a lack of information only reasonable approximation in the interpolation process can be used to fill them in.

On the other hand, a reverse projection method is potentially artifact free. If formulated correctly, a correct resampling in the output domain can be performed. If the view geometry is regularized such that the output resampling domains have a constant Nyqusit limit, then the required filtering is known. Reverse projection methods are also known as ray-casting, as opposed to ray-tracing. A major difficulty with reverse projection methods is the computational requirements of testing intersections with the surface and the work involved in ensuring a correct hit point.

If we limit the application of forward projection to cases where physical constraints can be applied, in order to minimize artifacts, a number of advantages can be gained. The primary area where physical constraints can be effectively applied is visibility determination. For all other components of view generation, reverse projection may be used. In Perigee, we exploit this idea and limit the use of forward projection methods to visibility determination. We use a reverse projection method for all other aspects of the computation.

3.3 Perigee Algorithm

In this section we give an outline of the algorithm. It is important to note that due to the proprietary nature of the algorithm and implementation, we omit certain details. The algorithm consists of four basic steps:

1. Determine the appropriate sample window
2. Determine visibility

3. Display output
4. Animation and repeat

Step 1 is accomplished using a reverse projection method (geometric coordinate transformation) to select the appropriate samples for the given viewport and window size. In the demonstration software, we use a 512x512 sample window and a 512x512 display window. We note that this is not a ray casting method and the main purpose is not to determine intersections. Step 2 uses a forward projection technique to test visibility, since multiple samples may still map to one or more output pixels. This method is considerably less costly in terms of computation than a standard intersection method, as would be required in a full reverse projection approach. Step 3 displays the output image and utilizes the general communication network (router) on the MasPar to pass the output pixel data to an I/O buffer. For this demo we use a device called I/O RAM as the I/O buffer. I/O RAM is essentially a memory device connected to the router network of the MP-1 and is accessible (read/write) in an asynchronous manner by the front-end. While the DPU is calculating the next frame, the Front-end is reading the previous frame from I/O RAM and displaying it on the workstation monitor. Step 4 is accomplished by moving the sample window to select the next set of samples based on the direction of motion; then the process is repeated.

3.3.1 Input Requirements

The basic input for Perigee consists of an image or texture map for the terrain and a corresponding height-field. In the case of our demo, this consisted of a 24-bit RGB false color landsat image with 30 meter resolution and a registered array of heights derived from DTED (Digital Terrain Elevation Data). The array of height values and image were provided in pre-registered format. The image had approximately 4000 pixels on each side. We note that the input format is not a fixed requirement; that is, there is nothing specific to the landsat image or DTED information that is required for the Perigee rendering system.

The lack of a parallel disk array posed some severe limits on our implementation of the demonstration software. In order to provide fast access to the data, we were required to place the whole data set (image + height) into the PE memory of the MasPar. We stored the first n image samples in PE0 and the next n samples in PE1 and so on.

3.3.2 Algorithm Description

3.3.2.1 Step 1: Coordinate Transformations and View System

We use a simple viewing system based on aircraft (vehicle) position and orientation in terms of Euler angles defined in the *yxz-convention*. The camera coordinate system is fixed on the vehicle and is left-handed, with the default orientation being: $(x_e, z_e, y_e) = (x, y, z)$, and (x,y,z) are world coordinates in a right-handed system. We define the orientation in terms of ϕ= yaw, θ= pitch, and ψ = roll, with z_e fixed along the body axis in the direction of view. We used standard 4x4 transformation matrices in homogeneous coordinates for all calculations of coordinate transformations.

3.3.2.2 Step 2: Visibility Determination

As was mentioned earlier, the visibility computation involves a forward projection and exploits spatial (scanline) coherence to minimize interprocessor communication and simplify the detection of visible surfaces. The scanline approach simply finds the screen location of each sample from the input image height-field, and based on the viewing angle and height of the sample, determines visibility of that sample. Non-visible samples are overwritten by the appropriate visible sample color.

3.3.2.3 Step 3: Display

Displaying the image on our system turned out to be one of the more difficult tasks. There were two main problems in transferring the output image from the PE memory to the front-end framebuffer. The first involved writing from PE memory to the I/O RAM device in the fastest possible manner. This required layout of the output pixels in a particular processor order, known as hierarchical. Then the front-end would read from I/O RAM and place the output image directly into its framebuffer. This occurred in an asynchronous manner, with the DPU calculating the next frame, while the front-end displayed the previous frame. Since the front-end monitor only supported 8bit color, displayed images are only 8bit RGB. The display window on all tests was 512x512 pixels.

3.3.2.4 Step 4: Animation

Motion over the terrain is achieved by moving the sample window, in the appropriate manner, over the data set. We note that full flight controls were not implemented since this demo was a simple proof-of-concept. However, four directions of flight, altitude and pitch change are implemented. Movement of the sample window is accomplished using a method that exploits computations done in previous time steps to estimate the region where a pixel will get its next sample for the next frame.

3.3.3 Performance Results

Performance on the MP-1 was approximately 13.5 frames/second displayed, and on the MP-2 it was approximately 20 frames/second. It is important to note that in both cases the DPU was calculating at a higher frame rate (15 frames/second, MP1 and 22 frames/second, MP-2), than could be displayed. The limiting factor in both cases was the bandwidth (approximately 10 MB/s) between the front-end and the DPU.

The Perigee terrain rendering demonstration software indicates superior performance for this type of rendering. A main advantage of this approach is that computational complexity is bounded by the display size and not dataset size. That is, increasingly complex scenes do not impose additional performance penalties as may be incurred with polygon data sets.

3.4 A Simple Ray Tracer

In order to test the performance for other rendering methods, we implemented a simple ray tracing program. This program implemented a ray tracing algorithm, which requires each processor to hold the entire object database. This implementation requires no interprocessor communication and is typically known as broadcast method, since the object database is broadcast to all processors simultaneously. The demo supported polygons or spheres, multiple light sources, shadows, and multiple reflections, with adaptive termination (e.g. below a certain threshold, ray contributions were terminated). We used no methods to accelerate intersection tests, except a simple bounding box method for the polygons. Any number of spheres (available memory being the only limit) were generated with random colors, reflectivity and positions. Rays were traced until the maximum number of reflections were performed, background space was intersected, or the intensity contribution fell below a given threshold. We used an approach, such that each processor had its own "pool" of rays it had to trace. Each processor grabbed the first ray in its pool, and began testing for intersections against every object in the bounding box or all the spheres in the database. If there was no intersection, a processor grabbed the next ray in the pool, while other processors continued to follow their original rays through multiple intersections.

Given two light sources, 20 spheres, four orders of reflection, and a 1024x1024 output image, rendering took approximately 20 seconds on an MP-1. We note that ray tracing is a floating point intensive computation and the MP-1 has no floating point unit and a 4bit processor. Output on the 32bit floating point MP-2 was nearly instantaneous for similar parameters, and approximately 100 spheres. Timing results for ray tracing approximately 1500 triangles with a 1Kx1K display was approximately 40 seconds for the MP-1. While this demo is by no means an optimal parallel ray tracing implementation or a practical implementation for interactive applications, it does hint at the potential performance of parallel platforms for alternative rendering techniques.

3.5 Polygon Rendering on Parallel Platforms

In this section we discuss results of a SIMD polygon renderer and a MIMD polygon renderer. Although state-of-the-art performance of polygon rendering is set by platforms such as SGI's Reality Engine, or Evans & Sutherland's ESIG platforms, these parallel renderers offer a glimpse of the potential to be gained from parallel platforms. We note that general purpose parallel platforms will not compare ideally with specialized platforms (e.g. Reality Engine) for polygon rendering, but their advantage comes from being able to perform other tasks that may be associated with the rendering.

3.5.1 SIMD Renderer

The polygon renderer was implemented and tested on the MasPar MP-1. We used an algorithm originally developed for the Connection Machine CM-2. This approach is a combination of a sort-middle and sort-last method. We distribute the polygon data set across processors in uniform manner and perform world-space to eye-space transformations. After transformations, but

before rasterization, polygons are passed to multiple processors (sort-middle), which handle particular scanlines in the image. Polygons are scan converted in this step. After scan conversion, pixels are sent to the appropriate processors holding the image buffer (sort-last) and z-buffered.

The uniform distribution of primitives for transformations provides a load balanced situation. Similarly, load balancing occurs in the scanline order rasterization, since the size of each polygon has limited effect on a processor basis. That is, each processor only operates on a fixed number of scanlines, and large polygons are shared among processors. The final step is the source of a communication bottleneck, when many primitives are writing to the same area of the image buffer. We tested the algorithm on a 5000 polygon dataset of the Space Shuttle, and achieved a frame rate of approximately 6 frames per second. An alternative implementation of our polygon renderer, based on the underlying principle used in our terrain rendering demo, achieved 10-15 frames per second on a 5000 polygon data set. Again, these results were on a 4K processor MP-1 with no floating point support. Significant performance gains were achieved in our MIMD polygon renderer described below.

3.5.2 MIMD Renderer

Onyx's polygon renderer, developed and implemented on IBM's SP2, currently computes complete scenes on each node and uses the High Performance Switch to output each frame to a specific node holding the image buffer. Since this work was done remotely, all timings are only for machine rendering since actual display devices were not available. Timing starts once the polygon set is loaded into the processor, and ends when the "Display Node" receives the full 24-bit per pixel screen data from the "Rendering Node." The current package includes several levels of detail including anti-aliasing from 2x2 to *nxn* samples per pixel, depending on available processor memory and final screen resolution. The system includes several illumination models including flat, depth, Gouraud or Phong shading techniques.

For the timing results being presented here, 100,000 randomly oriented right triangles with random colors are rendered into a 512x512 pixel screen, with 24-bit color, lighted, depth shaded, aliased, Z-buffered with a 50-pixel average fill per triangle. These specifications are made to keep somewhat consistent with the previous system's performance tests. A rendered triangle scene can be found at the end of this report (See Figure 3-3). A major difference between this rendering system and those described above are that this system renders complete frames at each node for a different view location per processor. This system is useful for movie generation but is inadequate for real-time interactive display, unless smaller polygon sets per frame are used.

We note that the system is very generic and unoptimized. It was developed to predict the potential performance of IBM's SP2 for polygon rendering. The platform used for these tests was IBM's 12-node SP2 at Northeast Parallel Architectures Center (NPAC) with a 32 MB/s High Performance Switch. Nodes 1-8 are thin nodes with 66.7 MHz clocks, while nodes 9-12 are wide nodes with 66.7 MHz clocks. All nodes are based on IBM's POWER2 version of the RS/6000 family. The details of the rendering system are described as follows.

Each participating node of the SP2 reads in the complete 100,000 triangle dataset into their own local memory. Since each node currently has 256 MB memory per node, this isn't a problem. The 100,000 element set contains triangles randomly positioned in a box with dimensions 100x100x100 units. They polygons are defined by three vertices and an RGB color. Each entity is a right isosceles triangle with sides of length 2, 2, and 2.8 units. When mapped to a 512x512 pixel screen, the triangle side lengths become approximately 10, 10 and 14 pixels. This is for all triangles oriented with their outward normal in the direction of the eye coordinate. The orientation of each triangle is also random, therefore approximately half of the triangles face away from the view location and will not be rendered. In addition, many triangles will be viewed from their edge and will have substantially less pixel fill. In order to maintain the 50-pixel fill average per triangle, the set is viewed from far away with a very small view angle (See Figure 3-1).

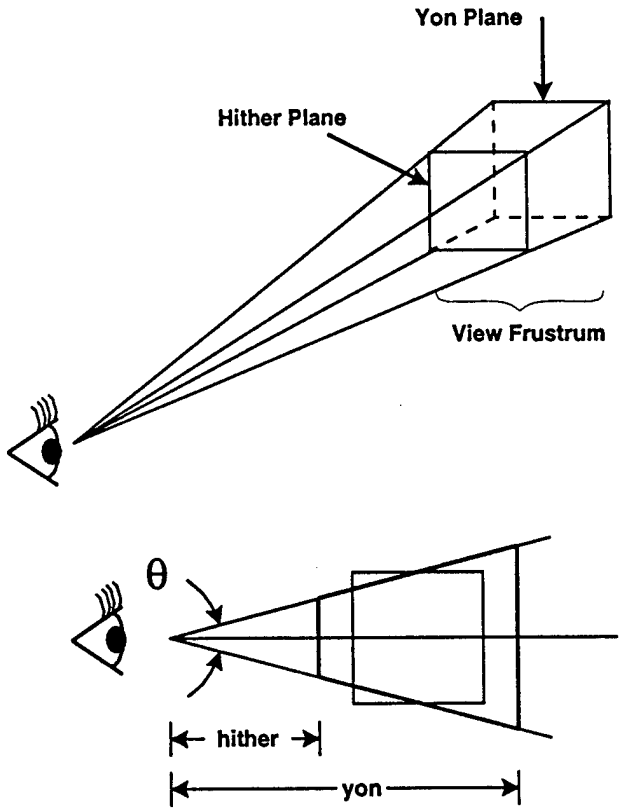

FIGURE 3-1. Viewpoints: hither = 11,400, yon = 11,560, q=.5deg

After loading the dataset into each node of the SP2, the timer is started for the following rendering computations. The first operation involves computing the local eye position at each node according to a pre-defined script, and computing all other global viewport values. Next, each node's local 512x512 structure of RGB values is set to the background color, and the Z-values

for each pixel are set to the back of the viewing frustrum. At this point, each node computes the outward normal for every polygon in the set. Approximately 50% of all triangles are then removed from the set through back-face culling, and the remaining vertices and normal components are transformed from world space to eye space.

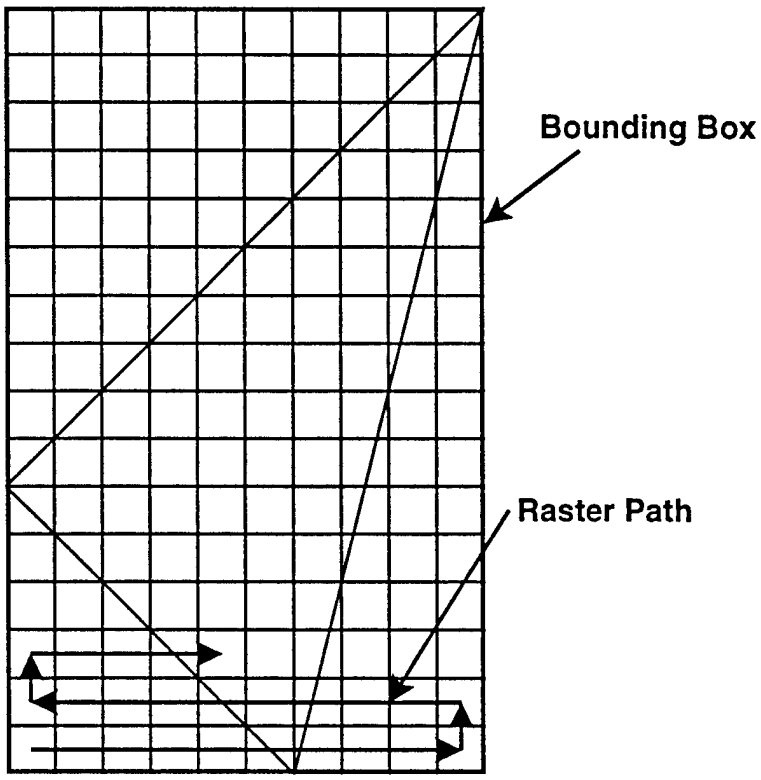

FIGURE 3-2. Rasterization Through Scanning the Bounding Box

As with most polygon rendering methods, the polygons are clipped from the viewing frustrum. To take into consideration as many aspects of polygon rendering as possible, the viewport is chosen so that some triangles intersect the viewing frustrum (See Figure 2-3). In Onyx's system, the polygons aren't "clipped" as many renderers do, but are instead flagged. When these polygons are filled, the rasterization process will stop at the edge of the screen limits. Many rendering methods will reform the polygon by clipping off the protruding portion, sometimes creating two triangles out of the original one. After the clipping process, all eye and lighting coordinates are transformed to view space, and all triangle vertices are mapped to screen coordinates.

The polygon set is now ready for rasterization. The rasterization performed in this system is very simple, however creates more computation than refined routines. The rasterization method used first computes a rectangular bounding box for each triangle. The rasterization involves incrementing the computed triangle edge functions over every pixel inside the bounding rectangle (See Figure 3-2). Essentially twice as many pixels are checked for inclusion in the screen than

when using edge searching algorithms. These algorithms traverse the triangle until an edge is hit, then move to the next scan line and proceed until another edge is encountered. This method degrades the performance greatly, however it was only chosen to quickly produce a prototype algorithm for immediate implementation on the SP2. As the bounding rectangle is traversed, Z values are tested against the current values and updated as necessary along with the pixel color.

After all polygons on a node have been rasterized, the RGB structure is sent to another node on the SP2 via the High Performance Switch. When that node responds that it has received the data, the next eye position is computed and the rendering process is repeated. Each node must wait for their turn in the frame sequence to send their composed picture to the display node. This causes some nodes to wait until the display node receives all previous frames in an animation. Only until a node delivers their respective RGB structure are they allowed to proceed to the rendering of their next frame.

This routine was repeated 20 times per node using four nodes for computation and one for receiving the rendered RGB data. Initial timing results gave an average run-time of 55 seconds per node to render the 100,000 triangle set 20 times. This is equivalent to approximately 36,360 triangles per node per second. For a 32-Node system, over 1.15 million polygons may be rendered per second. Similar tests were run on 10,000 triangle datasets where approximately 29,000 polygons were rendered per second, per node. In addition, tests were run on a 1024x1024 screen without sending the screen data to the rendering node. In this instance 30,800 triangles per node, per second were rendered.

3.6 Summary

The results of these prototype rendering implementations indicate high potential for increased graphics performance from massively parallel platforms. We note that while some of the performance tests did not achieve spectacularly high frame rates, it is important to remember that most of these tests were performed on first generation hardware. The newer platforms in the MasPar family, as well as offerings from other parallel vendors, promise performance improvements of more than 5-10 times that of the MP-1 system used in these tests. Code optimization, larger memory machines, improved processor and communication performance, all indicate that interactive frame rates can be attained using parallel machines.

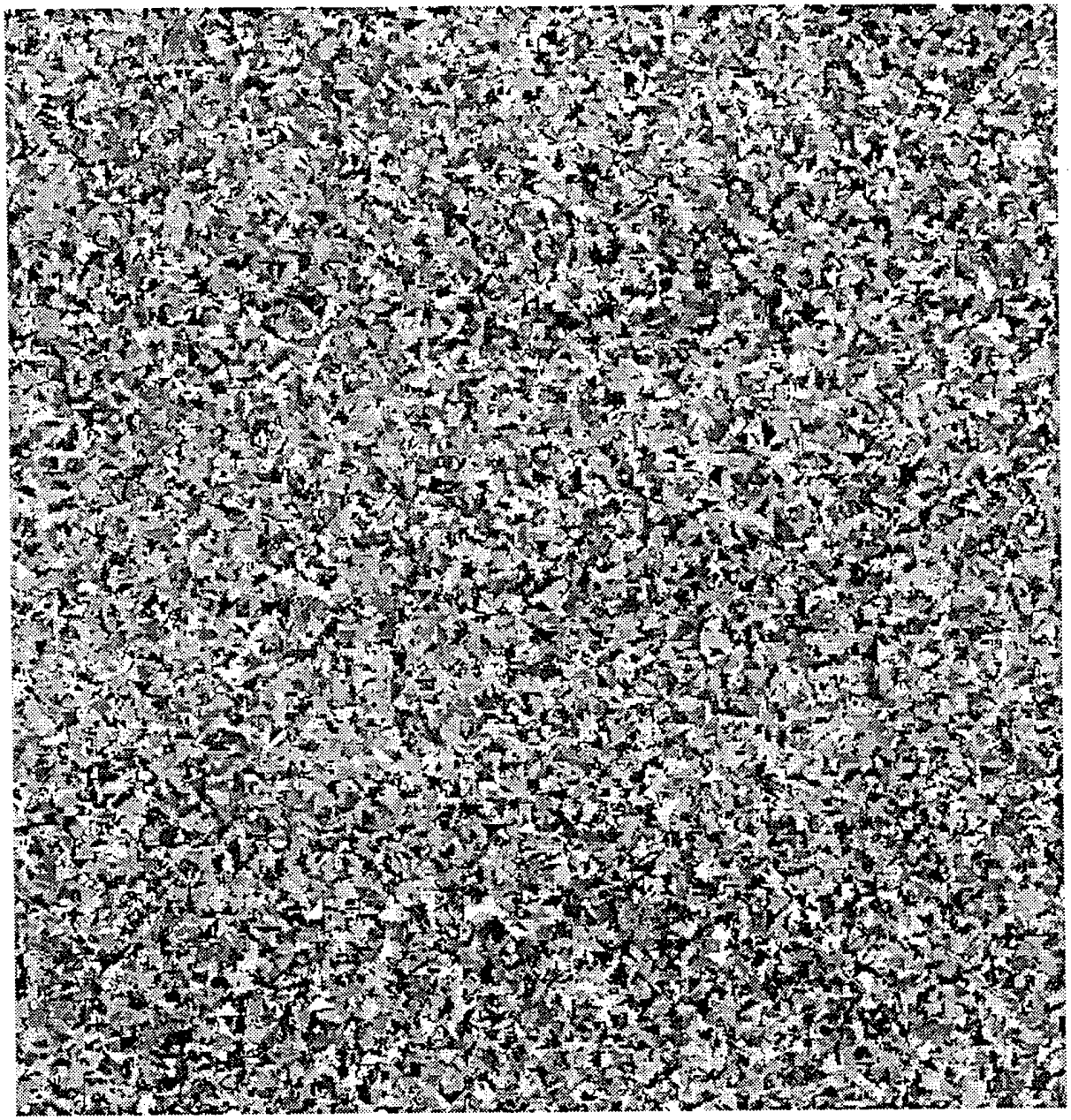

FIGURE 3-3. 100,000 Randomly Oriented Right Isosceles Triangles

4.0

Summary and Conclusions

4.1 Future Direction and Recommendations

The results of this project indicate the potential for increased graphics performance from massively parallel processors. However, in order to meet the widely varying needs of the Army's visualization and visual simulation requirements, we have proposed the construction of a Scalable Visual Simulation Toolkit (SVST). This toolkit is aimed at providing basic functionality to develop visualization and visual simulation applications on massively parallel and multiprocessor systems. The fundamental design goal for the SVST is to deliver a software development layer that provides a high level of graphics performance on a multiprocessor platform.

There are a number of reasons for selecting the above approach:

1. **Flexibility:** the toolkit approach will support a number of application development paths and provides a long-term strategy for exploiting multiprocessor systems for the U.S. Army's visualization applications.
2. **Standards:** the use of standards and commercial technologies support portability of the system, which is required for a successful commercialization effort.
3. **Interface:** an object oriented and multi-layer design separates the interface (user and system) from the implementation. This means a consistent API is used to access low-level routines, which may have different implementations on different parallel systems.
4. **Performance:** separating the interface from the implementation also enables the use of optimized low-level code, while presenting a machine independent interface. This applies to the development team as well as the toolkit end-users. End-users are free from worrying about low-level performance optimizations that have little to do with a particular application.

5. **Cost:** the object-oriented approach within the framework of the toolkit enables reduced life cycle costs, by reducing application development time (through reuse), amortization (build multiple applications), maintenance and system upgrades.

4.2 Parallel Systems in Distributed Simulation Applications

A typical entity simulator consists of some host software that runs the simulation (e.g. vehicle dynamics, terrain model, etc.), DIS communications software and image generator. This functionality can be implemented in software on currently available parallel platforms. Given the scale envisioned for DIS simulations and a requirement to monitor large scale scenarios, one possible application of parallel platforms in DIS is as a Stealth Platform (SP). The parallel platform serves as an independent node on a DIS network and is used to provide direct non-interfering access to all portions of the battlespace. The SP models all entities in the battlespace at an appropriate level of fidelity and provides interactive visualization of the battlespace. Global entity models are maintained in a local database, and state changes and dead reckoning are computed across the parallel machine. The parallel system may be close-coupled with a traditional graphics platform in order to provide additional visualization capabilities. However, obtaining high-fidelity views of the battlespace may require high resolution models which could overwhelm the rendering capabilities of a standard image generator.

4.3 Conclusion

As mentioned above, currently available "pipe-lined" graphics workstations are inadequate for the task of interactive visualization of very large-scale databases. Also, the visualization techniques being introduced today for producing realistic images, such as volumetric imaging, ray tracing, radiosity, particle systems, fractal-based rendering and 2.5D terrain rendering, bear little resemblance to the types of graphics (e.g. polygon rendering) for which the existing graphics workstations have been optimized.

Many of these techniques have yet to find their way into the visual simulation community mainly because they are computationally intensive and workstations cannot support the frame rates (30 frames/second) required for simulation purposes. With the increasing performance of parallel and distributed computing systems, coupled with the continuing decreasing cost of these systems, it is now practical to use these high-performance general purpose systems for simulation and virtual reality applications where high frame rates must be achieved. In the future it is possible, and even likely, that individual nodes of distributed simulation systems will consist of multiple computer architectures, including parallel computers, depending on the requirements of a given node.

Current implementations on parallel computers suggest they are well-suited for these "non-traditional" rendering techniques. Clearly the same performance advantages which a parallel architecture exhibits for large-scale numerical simulation or data processing tasks, apply equally well to large-scale visualization tasks. High performance computing systems, in particular parallel com-

puters, are becoming far more affordable with low-end prices falling in the range $100,000 to $500,000 (a similar range for high-performance graphics systems). These parallel systems have high-performance for real applications and often exhibit improved price/performance ratios, as compared to hardware augmented graphics workstations. Also, currently available MPP systems, since they are targeted for mission-critical commercial applications, also have high reliability.

It is clear that massively parallel systems can exhibit very high-performance for computer graphics, provided the proper hardware and software configurations are available. In particular, MPP systems exhibit clear advantages, primarily due to their flexibility, for various types of graphics (e.g. volume rendering, terrain rendering) that are not currently supported by hardware-based graphics subsystems. High polygon rendering performance, while supporting much larger polygon databases, is also possible with MPP platforms, although well-developed graphics workstations will continue to exhibit performance advantages while rendering small polygon data sets (<100K). Also, given the wide availability of polygonal databases, a total system architecture that combines the best features of graphics workstations and MPP systems, will provide a potent vehicle for the advanced graphics capabilities required for sophisticated visual simulations.

Appendix A:
Description of Development Platform

A.1 System Configuration Used for Prototypes

This appendix describes the architecture of the massively parallel processor system used to develop the prototype terrain renderer. All systems mentioned below have been developed by Onyx Sciences Corporation or its affiliates. The primary platform for prototype development was a MasPar MP-1 with 4096 processors. Below we describe the MP-2 platform, noting that the software developed for the MP-1 is binary compatible with the MP-2. We note that the MP-1 has been supplanted by the MP-2, and the soon-to-be-announced MP-3. Both platforms exhibit significant performance increases. The features of the soon-to-be-announced MP-3 are not available for public release, and we cannot quote any performance results for this platform.

A.1.1 Development Platform - MasPar MP-1/MP-2

The MasPar MP-1/MP-2 system is comprised of the following major subsystems:

- Front End (FE), a Unix Subsystem;
- Data Parallel Unit (DPU);
- I/O Subsystem.

The Front-End is a conventional UNIX workstation and, along with the I/O subsystem, is not discussed in detail in this document. Figure 1-1 shows a schematic diagram of the entire MasPar platform configuration. We note that our in-house system was not fully configured and did not have special I/O or storage facilities (Display systems, Disk Array, etc.).

Front-End

The MasPar Front-end is a standard unix workstation, (DECStation 5000). The primary purpose of the front-end is to provide the environment in which all programming tools and system utilities (compilers, etc.) are executed. In MasPar's asynchronous SIMD paradigm, the Front-End interfaces with the DPU, but does not explicitly control the execution of software on the DPU.

Data Parallel Unit

The DPU consists of the following major subcomponents:

- Array Control Unit (ACU)
- The Processor Element (PE) array
- Interarray Communication Mechanisms

The Interarray Communications Mechanisms ICM consists of the following interprocessor networks:

- X-Network (X-net), for grid communication between neighboring PEs
- Router Network, enables global or general PE-to-PE communications
- Common bus used by the ACU to broadcast instructions to all or selected PEs
- An OR-tree that consolidates responses from all PEs back to the ACU

The last two networks are global busses and are not discussed in detail further.

A.1.1.1 The ACU

The ACU executes the PE array instructions (broadcasting instructions to the PEs) and executes instructions within the ACU. The MP-1 ACU consists of a 32-bit RISC processor with a two address, load/store instruction set. It also has a 4GByte virtual instruction address space, managed in 4096 pages. Instruction and address data spaces are byte-addressed and independent. Both are independent from the FE workstations virtual address space. The ACU uses queues for interaction with the UNIX subsystem (FE). The ACU has 32 32-bit general purpose registers where all computation in the ACU takes place, and 32 32-bit special purpose registers that provide system status and control information for the ACU.

A.1.1.2 DPU - Processor Element Array (PE Array)

The PE array can consist of 1K to 16K processors. These operate in a SIMD fashion, meaning all PEs operate in parallel and execute from a common or single instruction stream. Each PE has its own data and can either execute or ignore an instruction based on a data-dependent condition code. Setting this condition code is often called "context switching".

The MP-1 PE processors each have separate instruction and data spaces. We note that a single instruction store provides the instruction stream for the ACU and all the PEs. This is the SIMD mode of operation. Each PE has local registers (PReg) and memory (PMem). PE registers are bit-addressable or can be used as variable length registers (1,8, 16, 32 or 64 bit registers). The regis-

ter size is implementation dependent, but at least 1280 bits is addressable. Local PE memory is only directly accessible to the PE that owns it. In general, register access is 10 times faster than memory access. It is important to note that PMem and PReg accesses (loads or stores) can occur concurrently, as long as no conflict results. In the event of a conflict, a hardware interrupt stalls the PReg until the memory operation is complete. PMem on another PE is accessed by requesting the PE to access its memory and send the result to the requesting PE.

A.1.1.3 DPU - Topology & Communication

The MP-1/MP-2 is scalable from 1024 to 16384 processors connected in a 2-D mesh with wrap-around (2D torus). Processors are 8-way connected, with normal mesh connections (north-south-east-west), plus diagonal connections (NE,NW,SE,SW). Communication on the mesh is supported by an independent communication network called Xnet. Global or general communications are supported by a router. There is one router wire for every 16 processors (a cluster). In general, Xnet is faster than router communications and has approximately 16 times the bandwidth of the router. However, all active processors must apply the same communication path simultaneously. Xnet produces approximately 1.22MB/s peak bandwidth per processor and 22.2GB/s aggregate bandwidth on a 16,384 processor platform. The router has 1.3GB/s aggregate bandwidth on the same size system. This gives a per-cluster bandwidth of 1.27MB/s, about 1/16 the bandwidth of Xnet since there is one router connection per cluster of 16 processors. Note the above performance figures are quoted for the MP-2 platform. The MP-2 has approximately twice the communications performance and up to five times the floating point performance of the MP-1 platform.

A.1.2 MP-2 (MasPar)

We note that the interprocessor architecture of the MP-1 and MP-2 are essentially identical. The main differences lie in the specific processor implementations and performance figures. The most relevant differences lie in the processor type. The MP-1 uses a 4-bit processor operating at 12.5MHz, while the MP-2 uses a full 32-bit processor for each PE. We give a brief overview of the MP-2 platform below.

A.1.2.1 Processors

The MP-2 uses custom 32-bit RISC processors operating at 12.5MHz, each with 25 64-bit registers and peak access bandwidth of 50MB/s. Each processor has from 64KB to 256KB of memory with a processor to memory bandwidth of 1.22MB/s. Each processor can access approximately 2.5 words of memory for each floating point operation. The processors have floating point performance rated at 0.15Mflop/s (64-bit) or 0.394Mflop/s (32-bit) peak performance and 4.15MIPS for 32-bit integer operations. These numbers are based on quoted figures of 68,000 MIPS (32-bit integer operations) and 6.3Gflop/s (32-bit or 2.4Gflop/s (64-bit) total peak performance for a 16,384 processor system.

A.1.2.2 I/O Support

The MP-2 supports HIPPI, VMEbus and ethernet. The MasPar Disk Array (MPDA) contains up to 528GB of storage capacity, accessible via the HIPPI channel at up to 200MB/s (bi-directional).

A.1.2.3 Compilers, Libraries & Software

The MasPar is specifically programmed in a data-parallel SIMD mode. This is supported by two compilers, MPL- a parallel extension to ANSI C and MasPar Fortran- a parallel extension to Fortran 77. The MP-2 also has MasPar VAST-2, a tool that automatically translates Fortran 77 source code to MasPar Fortran and vice-versa, which significantly aids programmers in porting scalar code to the MP-2 or MP-2 code to any scalar platform. Additional tools include the MasPar Programming Environment (MPPE) which allow users to interactively run, monitor, debug and analyze performance of programs. Included with these systems are a mature set of libraries including the MasPar Image Processing Library (MPIPL) consisting of convolution, filter, FFT and histogram modules and the MasPar Math Library (MPML) composed of linear algebra building blocks and solvers, as well as an FFT. Finally, the systems include the MasPar Data Display Library (MPDDL) which allow display of 8-bit and 24-bit images on 1-bit, 8-bit and 24-bit X windows or framebuffers.

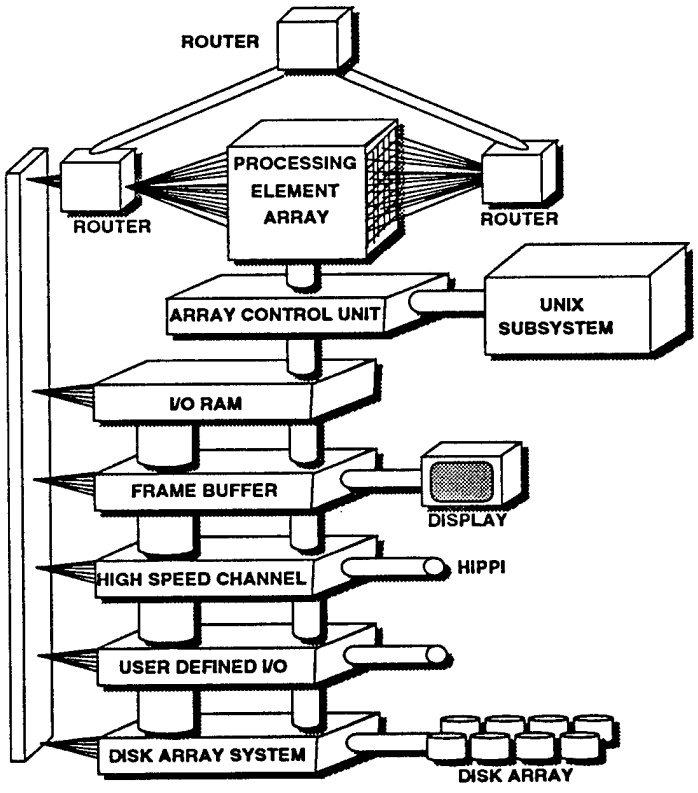

FIGURE A-1. MasPar Architecture