Distributed Virtual Reality Computation

Introduction

Virtual Reality is generally understood today to mean the combination of digitally generated graphics, sound, and input. The goal of most VR systems is immersiveness; generating the illusion that the user is in fact in another reality. This is best accomplished through the use of a multi-walled display system which surrounds the user with sound and images that respond to input. The applications of VR range from data visualization, engineering, and physics simulations, to anthropology, art, and entertainment.

One of the challenges in this field is that of computing power: where does one come up with the computing ability to drive interactive graphics and sound at such high resolutions? This paper will briefly outline a cluster-based approach to virtual reality computation.

The Traditional Approach

The “traditional” approach to immersive VR displays has been to use a single multiprocessor shared memory computer to drive all aspects of a program – I/O, simulation, and rendering. This has been and still remains an effective solution to the problems of multi-display VR. The shared memory architecture of such machines assures essentially no latency between display updates, and the multiple processors allow for a large amount of rendering and simulation power to be present in a “single bundle”. Memory access and allocation is handled transparently by the operating system. Distributing computation in such machines is largely trivial from the user’s perspective – the user simply makes graphics API calls which are easily parallelized by the operating system or the device drivers. Silicon Graphics Inc. is famous for
producing and selling such devices, and they continue to be used widely by academia, industry, and government.

The weakness of this approach becomes apparent however as soon as one tries to expand or upgrade such a system. The architectural challenges involved in engineering a user-expandable shared memory computer are nontrivial to say the least. This, paired with the high cost of such unique hardware and the pace of progress in the field of microprocessor design, makes the “single bundle” approach somewhat undesirable from a flexibility standpoint. Simply put: trying to make a bigger computer isn’t the answer to VR.

The Cluster Approach

It is of course possible for a single modern commodity desktop computer to effectively render three dimensional graphics to a single display, and for such a computer to remain responsive while responding to input, playing audio, etc. The focus of computer graphics hardware design has in the last decade shifted to a market with much more economic potential than engineering and data visualization – home entertainment. Gaming consoles and desktop computers are now available for less than $1000 that provide a significant portion of the computational power of a single multiprocessor graphics workstation that may cost tens of thousands of dollars or more. The obvious answer to the flexibility problem then would appear to be to construct a VR system consisting of a cluster of these machines.

In the field of high performance computing, a clustered system is nothing new, and effective methods and tools have been developed to create such systems. However the needs of VR impose a new and rather severe restriction: that of latency. A virtual reality system requires that the user be able to manipulate objects on the display in real time with what should appear to
be no delay between input and display state changes. Each node must update with input at the very least 20-30 times per second to maintain the illusion of continuous animation. This is somewhat of a paradigm shift from traditional HPC – rather than a single lengthy job that may take minutes or hours, this system is presented with an unending series of jobs each of which must be completed in a fraction of a second. The latency imposed by data transfer among nodes starts to become a very significant portion of the time the system has to render a frame, and it becomes clear that not all of the computations performed by a typical VR application are going to be distributable across memory spaces.

**Dividing the Work**

A typical VR application performs the following tasks for every frame:

- Receive & Send Input to Peripherals
- Run Physics or Animation
- Manipulate & Translate Geometry
- Render to Display

With the observation that a single node can easily perform all of these tasks for a single display, a simplifying step can be taken. Those tasks that are not easily distributable can be simply duplicated across the nodes. That is, each node can be running its own instance of the job, performing physics and geometry manipulations which are exactly redundant to those being done on the other nodes. This tradeoff unfortunately removes any benefits parallelization would yield in this area, but more importantly it removes a substantial amount of inter-node communication.
In most cases a given VR facility will only be able to connect I/O devices to a single node in the system, and in this situation network traffic is unavoidable. Fortunately I/O data tend to be quite light, and these data combined with synchronization commands comprises the entirety of inter-node communication for each frame. Typically only one node will be connected to audio output, although in theory audio output could be distributed with no added latency across a number of nodes equal to the number of sound channels available.

![Diagram of a VR cluster setup driving a four walled display](image)

**Figure 1 – A sample VR cluster setup driving a four walled display. Courtesy Eric Olson.**

The clustering benefits are great in one area - rendering. Each node in a VR cluster system will be rendering the same scene, consisting of the same set of textures and geometry and so forth. However, each node will be rendering a different view of this scene, which means that as far as pixels are concerned this task can be totally parallelized. Since rendering comprises the majority of the computation in most VR applications, this allows the clustering approach to be quite effective. With the exception of the minimal overhead of I/O transfer and synchronization, the rendering power of such a system can increase linearly with the number of nodes.
Figure 2 – Performance data for pantheon test application. Courtesy Eric Olson.

Figure 2 illustrates the difference in performance characteristics between a PC Cluster and an Sgi Onyx2, a multi CPU shared memory computer. The first thing to note is that the PC Cluster outperforms the Onyx2 by a factor of 3:1 even for a single display, which is a testament to the growing power of modern commodity hardware. But much more significant is that the PC Cluster experiences almost no drop in frame rate as the number of displays (and nodes) increases, whereas the Onyx2 pays a hard price for every active display. It is worth mentioning that a even a 4-computer cluster system is an order of magnitude less expensive than a single Onyx2.
The Epitome of Parallelism: The GPU

The GPU, or Graphics Processing Unit, is a specialized chip with functionality specifically designed for interactive rendering. While any CPU can of course render 3D graphics for VR, the custom design of the GPU allows it to be several orders of magnitude faster at this task. Everyday PC’s usually contain at least a modest GPU for 2D and 3D video acceleration, and nodes in a VR cluster would certainly be no exception.

The design of the GPU is highly constrained and specialized. The chip is typically designed to only accept as input a set of texture images and geometry, which it then quickly renders to an on-board frame buffer for display. Figure 3 is a greatly simplified diagram of a GPU rendering pipeline. Through a graphics API such as OpenGL, the user typically configures the way in which geometry will be rendered to the screen, and then passes a pointer to an array of vertices to be rendered. These vertices are processed by the vertex units on the GPU. Since each vertex can generally be transformed and lit independently of the others, a modern GPU will generally have 4-8 separate vertex pipelines, the combination of which can transform roughly 400-800 million triangles per second.

Each triple of vertices forms a triangle on the screen, which must then be filled with pixels. The results of the vertex pipelines are fed through a triangle setup engine, and then passed to the pixel pipelines. A typical GPU has anywhere from 4 to 16 pixel pipes, each of which is deeply pipelined and independent of the others. The pixel units are responsible for accessing texture images, performing lighting and shading, and other effects. These pipelines render the pixels to the onboard frame buffer at a tremendous rate – on the order of 2-8 billion pixels per second.
The result of all this parallelization and specialization is a rendering device which is hundreds of times faster than a general purpose CPU. The statistics are staggering; a modern GPU can reach floating point performance of 200 gigaflops or more when fully saturated. By comparison a fast CPU will generally perform in the 10-20 gigaflop range, and be further limited by its general purpose memory and cache structure.

The benefits of a GPU do not end with its sheer rendering speed. An off-CPU renderer allows the CPU to be free to perform other calculations while rendering is completed. This means that the complexity of a VR applications can now be far less limited by rendering
complexity, and gives programmers room to implement detailed physics simulations or other expensive computations. All of this parallelism is thankfully present within a single cluster node, preventing any network traffic between nodes for the purposes of rendering.

**Other Considerations**

To allow for synchronous display updates, all nodes in a VR cluster need to post their newest frames to the display at the same time, which can only be done after each node has finished rendering. Since the rendering load for each node is fixed, there is no benefit to having some machines in the cluster being faster than others, as the faster machines will only end up waiting for the slower ones at the end of each frame. Hardware for the cluster nodes is easily updated, although care must be taken to update *all* of the machines to get any benefit out of the new devices.

Developers that require more CPU cycles than a single processor could provide have the option of purchasing multi-processor nodes, or simply using MPI to distribute computation across several nodes. This must of course be done with the aforementioned latency concerns in mind.

Several tools exist to provide a framework for clustered VR development. Foremost among them is VR Juggler [www.vrjuggler.org], which allows a single application to be run in a cluster, as a stand alone, or on a large multiprocessor shared memory workstation. These tools remove the burden of distributed computing and threading from the user, allowing for rapid VR development across a multitude of platforms.