Binary-Swap and Shear-Warp Volume Renderer on the T3D

Roy Troutman, National Energy Research Supercomputer Center, Livermore, California, and *Chuck Hansen, Mike Krogh, James Painter,* and *Guillaume Colin de Verdiere*, Advanced Computing Lab, Los Alamos National Lab, Los Alamos, New Mexico

ABSTRACT: *Large parallel machines give today's scientists the ability to compute very large simulations which may generate equally large data. Not only does having visualization tools on the parallel system allow the scientist to take advantage of the large memory to visualize the data, the processing power allows interactive manipulation of visualization parameters.*

We will describe a volume renderer on the T3D which allows us to take advantage of the capabilities of the Shear-Warp renderer and the Binary-Swap compositing algorithm to produce an image in sub-second times, several seconds faster than other techniques. An interactive interface using AVS through a FDDI connection is described.

Introduction and Background

Many of the scientists working in a supercomputing environment are accustomed to performing their visualization tasks using postprocessing. The data is generated on the large systems, downloaded to a high end workstation and assimilated with one or more of a large variety of visualization packages. Figure 1 demonstrates an obvious problem with this paradigm. Many of the proposals requesting time on the MP systems appear to be more interested in the large memory rather than the computational power. Users are talking about adding more dimensions and scaling their simulations. This doesn't increase the resulting data by a few multiples, this increases the data size by several orders of magnitude. One possible solution is to provide visualization tools directly on the MP platform.

Ideally, the responsibility of creating and maintaining a parallel visualization system would fall on third party vendors. Unfortunately, because of the small numbers of MP systems sold, there isn't a great deal of market for this software. Hardware vendors havetaken up some of the responsibility, but usually can't afford to invest in creating a complete visualization environment. CRI offers a very efficient polygon renderer and ray tracer in their CAT product, but this system currently does not support transparency [2]. AVS is a complete solution which can be purchased for the MPP front end, but parallel modules are currently not an option. In many cases, special purpose software is written for a particular problem and a particular platform [1].

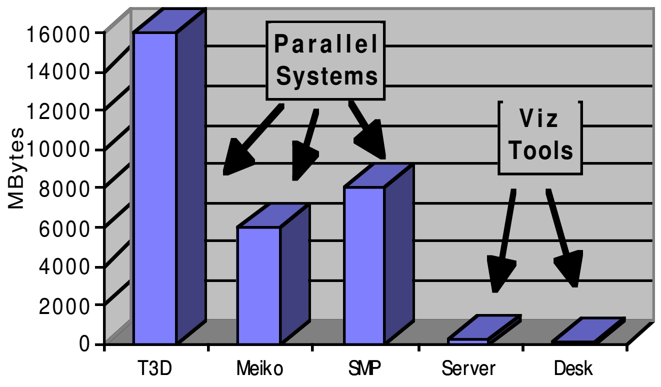

Figure 1: Memory comparison for machines at LLNL. SMP is 12 PE system, Server is a 128 MB machine and Desk is a 96 MB desktop.

It is our goal to construct a set of visualization utilities which allow us to take advantage of the unique features and problems of the MP system. This paper will concentrate on volume rendering. It is our goal to create a renderer which can scale to use a large portion of the MP system while still allowing the user to explore their data with an interactive component. It is also highly desirable to create a portable solution.

In this paper we will discuss a parallel volume renderer and enhancements needed to help us approach the previously mentioned goal. We will start by discussing the renderer which served as the foundation. We will then discuss enhancements made to the renderer. This will be followed by timing results of the rendering components.

Volume Renderer

The renderer that served as a foundation for this project was the Binary-Swap renderer of [4] and [8]. This renderer consists of a master process which transmits a color transfer function and a transformation to a set of slaves. The master then waits to receive an image to be sent to the console or a file. The slaves have four different stages as shown in figure 2. The setup stage receives the transfer function and transformation. The rendering phase renders the distributed data into several distinct images. The compositing phase combines the image into a single image which is also distributed among the processors. The collection phase then collects the distributed image into a single location and transmits the results back to the master.

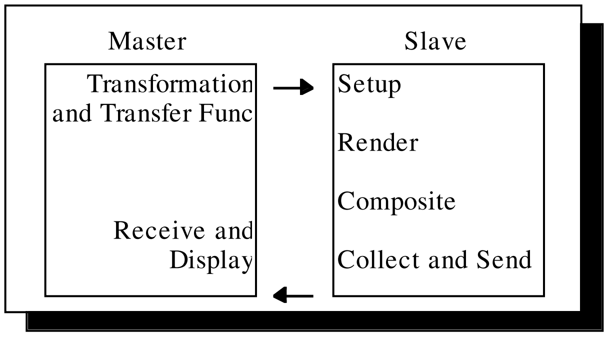

Figure 2: Phases of the Binary-Swap volume renderer.

The Binary-Swap renderer solves the issue of portability by thinking of the slaves as a heterogeneous collection of processors. Communication is done through PVM, which runs on virtually any UNIX platform. Scaleability was achieved in [4] by making only minor modifications to the initial code. Interactivity is difficult to determine. If a user can control the renderer in an event driven dynamic fashion, it can be said that the renderer is interactive. However, if the time between the users change and the displayed results is too long, the user looses the feeling of control over the 3D object. Higher speeds will actually enhance the feeling of looking at an object with depth. We set our interactivity goal at being able to render a reasonable sized image (256^2 to 512^2) in less than 1 second, which is at least 2 seconds shorted than the 32 processor cases of [4] and [8].

Hardware

The environment that we are using at the National Energy Research Supercomputer Center (NERSC) is a single cabinet configuration with 3 YMP processors acting as a front end for a 256 PE T3D. Each PE has 8 MW (64MB) of memory. There are 2 I/O gateways, 2 I/O processing units and no SSD. The operating system on the YMP is version 8.0.3.1 of UNICOS and the Alphas on the MPP side are running UNICOS MAX version 1.2.0.5.

NERSC imposes additional limitations to ensure efficient use of the T3D. A 32 processor maximum is set for interactive jobs. Obtaining the full machine can only be done through the batch system.

The code was compiled with version 1.0.3.3 of the C++ compiler. The strictly C code was compiled with standard C version 4.0.3.16. The version of PVM on the Cray was 3.3.4 or Cray version 2.1.1.

Some of the tests were run with the master process running on an SGI Onyx with a 90 MHz R8000 processor. Although the T3D has both a HIPPI and a FDDI connection, the Onyx only had the FDDI installed. Code compiled on the Onyx was compiled in 32 bit mode and linked with version 3.3.8 of the ORNL PVM library [7].

Renderer Optimization

This section will discuss the various modifications that were made to the Binary-Swap renderer to enable us to approach some of our goals. Our distributed algorithms is designed to have a master process control a collection of slaves on some homogeneous collection of processors. The slave does not change regardless of whether the program is an interactive, command line or batch oriented. Since most of the interesting work occurs in the slave, we will concentrate on the enhancements made to this code. Displaying the image is simply a matter of writing the received data to a file or passing the image off to AVS.

Setup Phase

Thinking of our collection of slaves as a large set of homogeneous processors within a single system would imply a different paradigm for broadcasting the transformation through the slaves. An MP system such as the T3D has a high speed connection between processors. The message should be sent to a single processor and then broadcasted through this high speed network. Sending such a message to PE 0 and broadcasting is the recommended method in [6] and was covered in [4]. Using this method increases the scaleability of our program.

During the setup phase, the original code waited for a transfer function and a transformation before the rendering phase was initiated. Since a new transfer function may imply having to reset a number of parameters associated with shading, the setup phase was changed to allow PE 0 to wait for either a transformation, a transfer function, a transformation and a transfer function, or a message indicating the master is finished. In essence, this makes the setup phase an event loop reacting to input from the master.

A problem with the setup phase was encountered while running the renderer with an interactive master process. When the master receives an image from the slaves, it displays the results and waits for the user to change something before signaling the slaves to render another image. During this time, PE 0 has executed the pvm_recv function to obtain the next message from the master. It was found that an enormous amount of time was being incurred executing system calls by the mppexec process associated with the slaves. The mppexec process appeared to be polling rather than waiting in a UNIX select call [9], which impacted other users on the system while our process was doing nothing more than waiting. A method was needed to signal PE 0 that a message was sent rather than allowing PE 0 to block in pvm_recv. The pvm_sendsig procedure appeared to be an obvious solution, but signals did not seem to arrive from a remote host with any reliability. Since this same behavior was encountered using two workstations, it is suspected that the problem may be somewhere in the version of PVM running on the Onyx. Another solution is to create a socket between the master and PE 0. When the master was ready to send a message, it sent a single byte through the socket to unblock PE 0. PE 0 then executed the pvm_recv call to receive the incoming message. Consultants from CRI had been informed and are working to determine if there is a problem with pvm_recv.

Rendering Phase

The real meat of the volume renderer is creating images from the data. This is the most time consuming phase of the Binary-Swap algorithm. We solve this problem by replacing the ray tracing renderer with the much quicker Shear-Warp renderer introduced in [5]. This volume renderer shears the data slices, composites the images produced by these slices and then warps the intermediate image. The result is a renderer which can produce an image in sub-second speeds.

Much of the speedup from the Shear-Warp renderer is obtained by encoding time saving structures such as the octree and encoding the volume in a structure called a classified volume. Additional speed is obtained by creating lookup tables for some of the shading functions rather than reevaluating the shading function at each sample.

The transfer function used in Shear-Warp is the multi-material model of [3]. This model is different from what is used by Binary-Swap and by AVS, but the interface to Shear-Warp is flexible enough to support this model. The shading model used in Shear-Warp is a phong shading function multiplied by an opacity similar to

$$I = a\,[\,I_a + I_d\,(N \bullet L) + I_s\,(N \bullet H)^n]$$

where I_a, I_d and I_s are the ambient, diffuse and specular components, N is the normal, L is the vector to the light, H is the halfway vector, n is the specular exponent and a is the opacity associated with the sample. The values inside the square brackets are encoded into a shading table with an entry for a predetermined number of possible directions. What we would like is to have the diffuse component vary with the data as does the opacity. Rather than have 256 different materials, we set the ambient to 0 and separate the equation into a diffuse and specular component. This creates two different shading tables. We then use Shear-Warps ability to replace the shading function with a callback function. This function simply uses the voxel data as a lookup into the transfer function and then multiplies that by an entry from the diffuse shading table. An ambient term is added in along with the entry from the specular table and the result is multiplied by the opacity.

Composite

This phase was the easiest to enhance. The version of the Binary-Swap renderer of [4] used several calls to the standard memory manipulation routines such as memcpy and memset. The version of memset that was being used was extremely inefficient. An hours worth of work produced a robust version in C which was 39 times faster. With the upgrade of UNICOS MAX from 1.2.0.4 to 1.2.0.5, we were given a version which was over 50 times faster. This resulted in significant speedup of this phase.

Image Collection and Sending

Communication to the MP can be enhanced by setting parameters with pvm_setopt [7]. One of the most important parameters to our interactive renderer is PvmRouteDirect and PvmFragSize. A direct route allows us to bypass the pvm daemon when sending pvm messages. When used on the Onyx, this option alone will result in considerable time savings. On the YMP front end, it actually increased the amount of time to receive large messages. PvmFragSize increases the size of internal communications buffers. On the Onyx, this resulted in a much smaller but still significant improvement.

Implementation

In order to combine Shear-Warp with Binary-Swap, the existing render in the Binary-Swap library was replaced with a simplified object oriented interface with functions to initialize, set the shading, set the transformation and render the image. The results from the rendering function is an image in a format compatible with the existing compositing functions. Separating the renderer from the Binary-Swap library in this fashion will eventually allow us to replace the rendering phase with different techniques such as isosurface extraction or even a different volume renderer.

Volpack version 1.0b3 was then obtained from the ftp server at Stanford and ported to the T3D. Getting to the point where we could create an image using only the Volpack library on a single processor was a time consuming process. Much of this time was due to the assumption that the original programmer made that the size of an integer was 32 bits and the size of a short integer was 16 bits. This assumption will cause us additional problems when we try to render larger volumes. Since structures are expanded to the size of a word (64 bits) a very large amount of memory is currently being wasted. We also encountered some difficulties with the Cray C compiler when using the function rint and using negative offsets from a pointer.

Once a serial version of Volpack was working we taught it to render only a portion of the data by introducing a collection of translations and scales that correctly positioned the data subset. Ideally, these transformations would be outside of Volpack to help encapsulate the library. Currently they are imbedded within the warp routines.

Results

Two different timing tests were run on the same 128^3 dataset of [4]. An image showing the orientation, but not the color, is shown in figure 6. The first is an interactive test where the master is a module in AVS. This allows us to use existing interactive features of AVS for the specification of the transformation and the transfer function. The AVS module was designed to have input ports for data, a colormap and a transformation and output an image. The output is typically connected to the display tracker module which can be used to interactively manipulate the transformation. A 256^2image was produced and display trackers ability to scale the image was used to double the resolution. We rendered the data using an octree encoding and the classified encoding schemes. Table 1 and Figures 3 and 4 show the times for the different phases using the octree and classified encodings. The timings were taken during midday hours with several other users on the T3D, which can cause a small amount of variation. The collect time includes the time to collect the image onto PE 0 and then send it out to the master. The collect and render phases have a wide variation in rendering times on different processors. The maximum time of any one processor is represented.

The previous timings involved only a change to the transformation. In this case, some of the internal structures do not have to be rebuilt. Rebuilding these structures can be somewhat expensive. Figure 5 shows the amount of time needed to complete the rendering cycle when a change is made to the transfer function. Although the classified renderer is much faster, the amount of time to build the structures is significant. The times for the both methods could be significantly reduced by using parallelism inside the encoding process. Although the encoded data is not shared, the shading tables are identical.

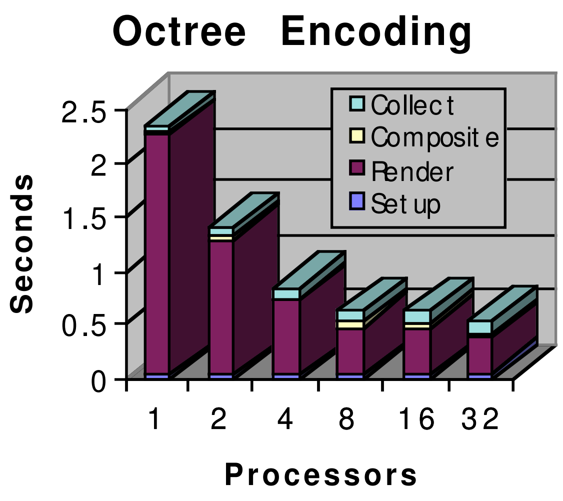

Figure 3: Accumulated time for interactive rendering using 128^3 data creating a 256^2 image with octree encoding.

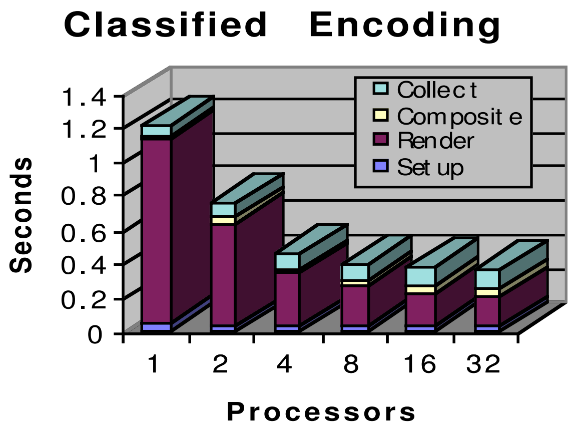

Figure 4: Accumulated time for interactive rendering using 128^3 data creating a 256^2 image with classified encoding.

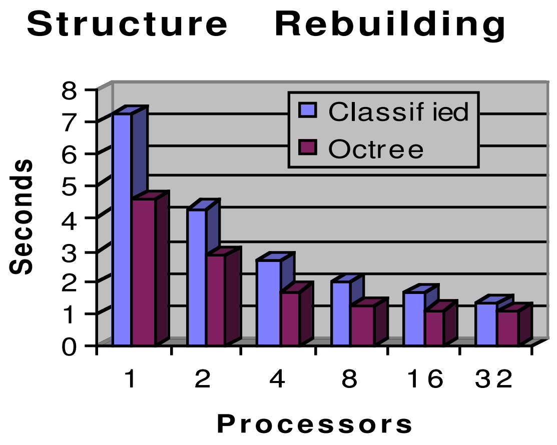

Figure 5: Times to complete the rendering process once a change in the transfer function is detected. The time for the octree method with 32 processors is 1.08 seconds.

The enhancements made to speed up the interactive version also enhance the command line version. In this case, we only have one transfer function and each new frame involves sending just a new transformation. Timings were taken using the Onyx as the master to produce the same image as in previous tests with a higher resolution as shown in table N. This table shows a the expected decrease in rendering times, but also shows a gradual increase in the collection time. Since the collect time for the 1 processor case is only due to transmission time, we can see this is a significant portion of the collection phase.

Conclusions

We have described a volume renderer which can produce an image at interactive rates. Although the interactive goal was obtained, there are several areas which can use additional enhancements.

Changing the transfer function resulted in an understandable increase in the amount of time due to the necessary recomputa-

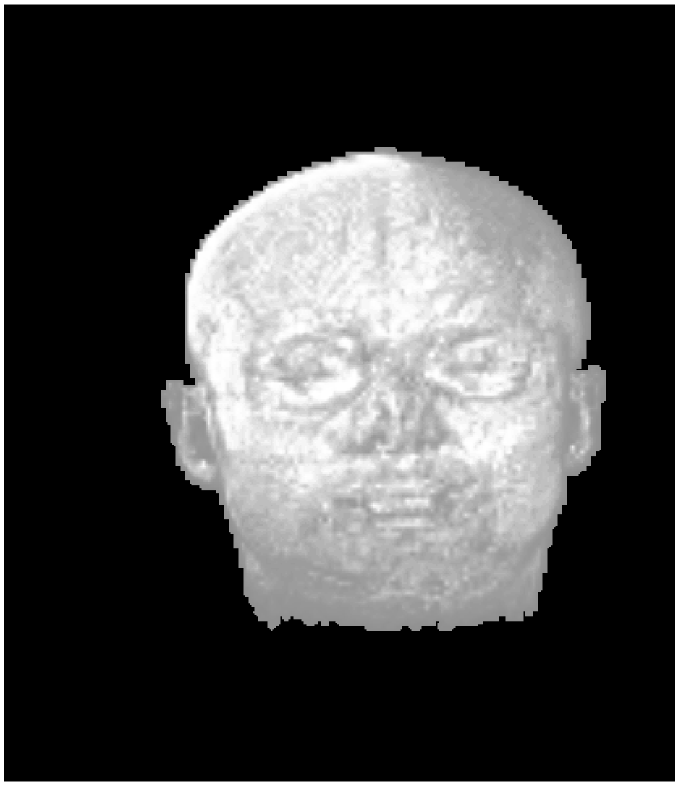

Figure 6: Grayscale image of the 128^3 dataset used in the timing tests. The image used for the actual tests was full color with areas of complete and partial transparency.

tion of the shade tables. Since this information is the same on all processors, a parallel algorithm is the obvious solution. Finding and exploiting other areas of parallelism would decrease rendering times even further.

The times given for the rendering all phases were the maximum of all the processors. The setup and composite phases did not have significant variation. The rendering phase had a great deal of variation due to the transparent regions and the octree encoding. Creating a balanced load in an environment with a dynamic transfer function is another challenging problem.

One of the most disturbing trends in the timing tables is the amount of time needed in the collection phase. A significant portion of this time is due to network bandwidth limitations. Exploring methods of fast parallel compression and decompression should make this phase much less troublesome.

Acknowledgments

This work was performed under the auspices of the US Department of Energy by Lawrence Livermore National Laboratory under contract number W-7405-ENG-48.

We would like to thank Mike Stewart and Scott Emery of CRI for their helping us understand PVM and the T3D. A special thanks goes to Phillippe Lacroute and Marc Levoy for giving free access to their source via the net. Their generosity has enabled many researchers to take leaps rather than steps.

Bibliography

[1] 1993 Parallel Rendering Symposium Proceedings, IEEE Computer Society in cooperation with ACM SIGGRAPH.

[2] Cray Animation Theater 1.0 Release Overview and Installation Bulletin, publication RO-5275, Cray Research, Inc.

[3] Drebin, Robert A., Loren Carpenter and Pat Hanrahan, "Volume Rendering", 1988 SIGGRAPH Conference Proceedings, pg. 65-74.

[4] Hansen, Chuck, Michael Krogh, James Painter, Guillaume Colin de Verdiere and Roy Troutman, "Binary-Swap Volumetric Rendering on the T3D", CUG 1995 Spring Proceedings, pg. 61-69.

[5] Lacroute, Philippe and Marc Levoy, "Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation", 1994 SIGGRAPH Conference Proceedings, pg. 451-458.

[6] PVM and HeNCE Programmer's Manual, SR-2501 5.0, Cray Research, Inc.

[7] PVM Version 3.3 Programmer's Manual, Oak Ridge National Lab.

[8] Rowlan, John S., Edward Lent, Nihar Gokhale and Shannon Bradshaw, "A distributed, parallel, interactive volume rendering package", 1994 IEEE Visualization Proceedings, pg. 21-30.

[9] Stevens, W. Richard, "UNIX Network Programming", Prentice Hall Software Series, 1990.

Table 1. Timings for the 32 processor case using 128^3 data

Octree	1	2	4	8	16	32
Setup	0.039	0.038	0.039	0.038	0.041	0.043
Render	2.236	1.250	0.700	0.438	0.418	0.336
Composite	0.004	0.030	0.018	0.041	0.040	0.041
Collect	0.061	0.086	0.082	0.090	0.117	0.114
Classified						
Setup	0.050	0.040	0.039	0.040	0.043	0.042
Render	1.094	0.599	0.323	0.232	0.192	0.176
Composite	0.004	0.031	0.018	0.042	0.040	0.041
Collect	0.062	0.081	0.085	0.091	0.113	0.117

Table 2. Timings for command line verson using same 128^3 dataset tc create a 512^2 image.

	1	2	4	8	16	32
Setup	0.049	0.036	0.033	0.038	0.038	0.039
Render	0.899	0.503	0.268	0.241	0.176	0.153
Composite	0.014	0.110	0.065	0.148	0.144	0.136
Collect	0.152	0.217	0.241	0.265	0.265	0.277