Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Multi-layered Image Caching for Distributed Rendering of Large Multiresolution Datasets

Jonathan Strasser[† 1] Valerio Pascucci[‡ 2] Kwan-Lui Ma[§ 1]

[1]University of California, Davis
[2]Lawrence Livermore National Laboratory

Abstract
The capability to visualize large volume datasets has applications in a myriad of scientific fields. This paper presents a large data visualization solution in the form of distributed, multiresolution, progressive processing. This solution reduces the problem of rendering a large volume data into many simple and independent problems that can be straightforwardly distributed to multiple computers. By completely decoupling rendering and display with image caching, we are able to maintain a high level of interactivity during exploration of the data, which is key to obtaining insights into the data.

Categories and Subject Descriptors (according to ACM CCS): I.3.2 [Computer Graphics]: Distributed/network graphics, remote systems; I.3.3 [Computer Graphics]: Picture/image generation; I.3.8 [Computer Graphics]: Applications
Keywords: image caching, interactive visualization, isosurface, multiresolution data, parallel and distributed rendering, progressive visualization, volume visualization

1. Introduction

Scientists nowadays can simulate fairly sophisticated physical phenomena or chemical processes at high fidelity using massively parallel supercomputers. Each simulation run can generate a vast amount of data which can easily overwhelm most of the data analysis and visualization software tools. Compressing the data would defeat the original purpose of performing the high-resolution simulation. Rendering and viewing the highest possible resolution of the data directly, if not impossible, would lead to long wait time unless a sufficiently powerful parallel computer and a dedicated high-speed network are used. Considering those who wish to use visualization on their large data can have a wide variety of computational power, a true solution should be able to fully take advantage of their system however large or small it is. The system should be distributed to take advantage of clusters, which are becoming more common due to their relatively low cost. An approach that can somewhat relieve the computational and hardware requirements is to visualize the data using progressive refinement. This is especially attractive when the scientist is exploring their data to search for either known or unknown features in the data. In such data exploration mode, it is very important to maintain a high level of interactivity.

In this paper, we present an interactive visualization system based on progressive refinement and distributed rendering. Progressive refinement is made possible with a hierarchical multiresolution representation of the volume data. A particular level of interactivity is guaranteed by 1) completely decoupling rendering and displaying using an image caching approach [LP03], 2) using a multiresolution representation of the volume data, and 3) distributing the rendering calculations to multiple computers. When using this system to visualize a large data set, the user can always receive immediate visual feedbacks as he browses in the spatial do-

† jkstrasser@ucdavis.edu
‡ pascucci@llnl.gov
§ ma@cs.ucdavis.edu

main of the data, and as the user pauses the system continuously and progressively refines the visualization by switching from a coarser level of the data to a finer level. The main contribution of our work is to integrate a set of techniques introduced in [LP03] and [PLF*03] into a coherent system. The results of our performance study using a cluster computer and a terascale dataset show the practical value of our system especially in a remote visualization setting.

2. Related Work

As stated before, one of the main problems of the visualizations of large dataset is the fact that the data cannot be stored in main memory. Prohaska *et al.* [PHKH04] handle this problem by using remote data storage. This storage was designed to support very efficient access of subvolumes. Throughput was increased by taking care of several high-level operations remotely and transferring the results over the network. Threading is used to allow for data access while the current data is being rendered. After the data is loaded, 3D texture rendering can be used to render the image.

Guthe *et al.* [GWGS02] used a preprocessing step to convert the data into a hierarchical wavelet representation. The data is decompressed during rendering and uses hardware texture mapping for the actual rendering. This was later improved upon by adding empty space skipping and occlusion culling [GS04]. Wang *et al.* [WGS04] also preprocessed the data into a multiresolution hierarchy using the wavelet transform. This hierarchy is partitioned and distributed over several nodes. Visualization is completed by traversing the wavelet tree and reconstructing data blocks. The results from the nodes are then composited. Akiba *et al.* [AMC05] take the wavelet based mulitresolution representation of the data and partition it into blocks. These blocks are packed into a more compact form on the fly according to the transfer function. This packing and the rendering are both hardware accelerated to achieve fast data reduction and visualization.

Ahrens *et al.* [ALS*00] address the problem of rendering times by providing the ability to render large datasets in parallel. This is achieved by adding parallelism to the visualization toolkit(VTK) [SML96]. This allows a scientist using VTK to easily upgrade to their version and benefit from the new parallel abilities. Since it is using VTK, modules can be added to give the scientist more power over how the scene is rendered. Engel *et al.* [EEH*00] give scientist the power to visualize by using high end remote servers to visualize medical data on a local desktop.

When dealing with multiple levels of detail, Lamar *et al.* [LHJ99] used an adaptive octree texture visualization. Cells near the region of interest were high resolution while those far away were low. Weiler *et al.* [WWH*00] showed how to use a hierarchy to insure smooth interpolation between resolution levels. Our work differs from these previous efforts since we break the problem into many simple rendering problems. This allows us to not depend on a specific rendering algorithm or hardware rendering in order to remain interactive.

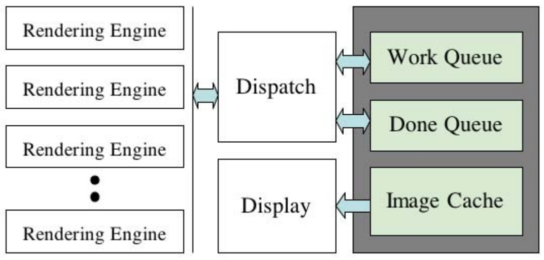

Figure 1: *The overall structure of MLIC. The display and dispatch access the shared memory from different processes. The dispatch communicates with the remote rendering engines and updates the cache with the images that they return*

3. Base Technologies

Our work started by building upon previous projects at Lawrence Livermore National Laboratory. One of these projects is MLIC [LP03], which stands for Multilayered Image Caching. MLIC was a visualization system that uses the concept of image caching to decouple the displaying of images from the rendering. The other project was ViSUS [Pas04], which gives us a series of tools for efficiently loading and visualizing large mulitresolution datasets. We briefly describe these two projects in this section.

3.1. MLIC

As seen in Figure 1, MLIC is divided into the display, the dispatch, and the rendering engines. The dispatch and display communicate through a set of queues stored in shared memory. During a single display cycle, the display first updates the image cache with any new images from the done queue. It then draws what is currently in the image cache, and checks to see if more images are required to meet the user requested image quality. Any required work is then put into the work queue, and the cycle repeats. The dispatch checks the work queue and distributes the required work to the cluster of remote rendering engines. When the engines return the result is placed by the dispatch into the done queue.

Even though MLIC was designed to more cleverly utilize computing resources, it fails to handle large data sets. The actual rendering is decoupled from the displaying of the final images by using a process for the displaying of subimages currently in the cache, along with a process for dispatching work to the rendering engines. This allows the user interface of the display and the displaying of images to remain interactive at all times. This interactivity granted the

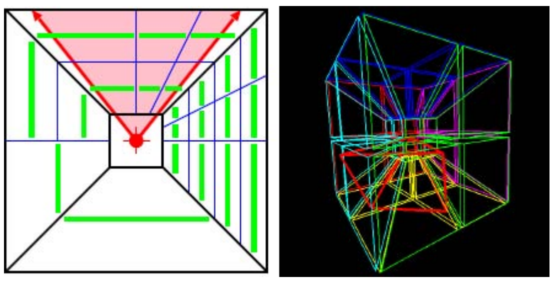

Figure 2: *A 2D and 3D example of how the image cache surrounds the camera. Figure provided by [LP03]*

user the ability to browse the data as the image is continually improved over time. The resolution of the screen can be modified at any time, though the resolution of the subimages is fixed so the system only takes advantage of a high resolution display when there are many sub images. Actual rendering was taken care of by VTK [SML96].

The design of MLIC's image caching is to have a stationary camera with images placed around the camera like a cube. Each side of the cube corresponds to a viewing frustum for that direction in space. At first this a single image is rendered using this frustum. Then the frustum is spit down one axis like a KD-Tree. Each split doubles the number of images and each individual sub-frustum corresponds to a smaller section of space. This will give us the power to load only the data required to render a subimage, letting us load higher resolution versions of the data as the size of the sub-frustums decreases. [LP03] provided a figure to visualize the cache around the camera as seen in Figure 2.

This project works well as a base due to several reasons. It already works in a distributed environment. The system parallels the multiresolution idea, so fewer subimages for low resolution data and many subimages for high/full resolution. Finally the independent nature of the rendering engines allows us to treat the rendering of subimages as a traditional simple rendering problem in a divide-and-conquer sense.

3.2. ViSUS

ViSUS stands for Visualization Streams for Ultimate Scalability. It provides a suite of tools which allow a user to visualize and browse large simulations on nearly any computer. ViSUS streams data to the users computer as fast or as slow as the computer will allow. Slower computers will be able to visualize very coarse data in real time and improve to a higher resolution when desired. Higher end computers may be able to start at a higher resolution, decreasing the time it takes to finish the final image.

ViSUS is also capable of streaming data to the user as it is being simulated. This allows scientists to begin to visualize and browse the data during the simulation, and if needed, refine the parameters of the simulation if they see fit possibly saving valuable computer time.

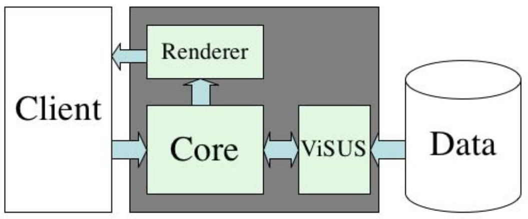

Figure 3: *A figure of the modified remote rendering engines. Information is sent from the dispatch and analyzed. The engine then asks for the required data at a specific resolution from ViSUS. This data and information is then passed on to a renderer. The result image is then sent back to the dispatch*

To enable more efficient access to large data, ViSUS uses a Z-order space filling curve [PLF*03] to avoid loading data that is not required for a requested subvolume. Data is also loaded progressively, loading the data from coarse to fine. This data can be visualized at the coarse resolution as the data continues to stream in.

ViSUS offers many tools for the user, but all rendering is limited to power of the users machine. More information on ViSUS can be found at the project website [Pas04].

3.3. Combining the projects

We combined the two projects to utilize the parallel and distributed nature of MLIC while taking advantage of the useful data tools given to us by ViSUS. To accomplish this, ViSUS was taken off the displaying computer and moved to the cluster. The remote rendering engines were redesigned, removing VTK, and adding an interface to the ViSUS data loading libraries. With this change, the rendering engines are now composed of 3 important sections. The core, ViSUS, and the renderer. The core of the engine refers to part that deals with in incoming request and sending that information to ViSUS, which is used to load the required data at the requested resolution. The core also sets up a non-multiresolution environment for the renderer, giving us more options on what the actual renderer can be. A diagram of this new design can be seen in Figure 3.

With the combination of the two previous projects we have VMLIC (ViSUS enabled Multilayered Image Caching).

4. VMLIC

VMLIC takes an image and splits it into multiple images by splitting the frustum corresponding to that image along each axis. These new sections correspond to specific areas in space and, in turn, correspond to specific sections of data. We can now take advantage of this fact, and load different resolutions of the data based on the size of the section needed. Once the section has been identified, it is put into a queue to be rendered by one of the remote rendering engines.

Once one of the remote rendering engines retrieves this information, it calculates what section of data is required, and then determines what resolution of the data can be fit given a memory constraint. The data is then loaded using the ViSUS library. Now the information given originally from the dispatch is converted to make it relative to the new section of data loaded.

This information is passed onto the renderer to be rendered. After rendering, the result is sent back to the dispatch. The dispatch takes this subimage and stores it into the image cache. Now when the final image is displayed, the subimage will be displayed along with any other subimages already in the image cache.

Increasing the number of subimages displayed will increase the overall picture quality in two ways. First increasing the number of subimages increases the resolution of the final image, since the subimages have a fixed size. This of course cannot increase the resolution beyond what the Display is capable of. Secondly, increasing the number of subimages decreases the size of the sections the subimages correspond to. This allows higher resolution versions of the dataset to be loaded. Examples of increasing the number of subimages can be seen in Figure 4. It is clear that the system is capable of showing both low resolution(data and pixel) as well as high resolution final images.

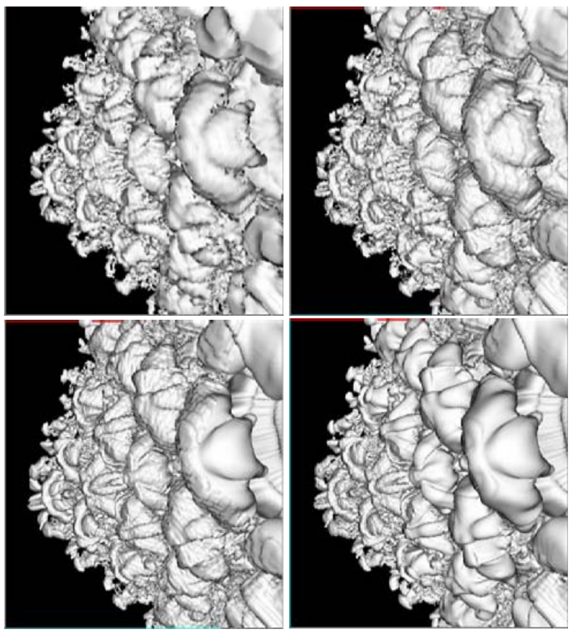
Figure 4: *An example of the same scene with different numbers of subimages. It is clear to see that as the number of subimages increases, both the resolution of the data and images quality are greatly increased. The increasing order is: Top left, Top right, Bottom left, Bottom right.*

4.1. Rendering

This process breaks the large scale rendering problem down into many small rendering problems. These images are rendered individually and have no knowledge of each other. The actual renderer of these images has no multi-resolution requirements, and can be a simple standard renderer. This allows the use of algorithms and renderers that may not seem to be possible with large datasets. With this in mind, the system was designed to be renderer independent with only a small set of restrictions.

What we call, renderer independence, was achieved by designing the new remote rendering engines to format the information from the dispatch, along with the data from ViSUS in a way that hides the mulitresolution nature of the problem. We wanted it to have a divide-and-conquer feel to it. Taking a large complicated problem, and turning it into many small simple problems.

On an implementation level, the engines were designed with a skeleton rendering class designed to be inherited from. It has a small set of simple functions (to set the dataset, viewing frustum, etc.) that are easily overloaded. The user simply creates a subclass of of this renderer (which can either contain the actual rendering code, or call functions in a desired renderer) and builds the system with it. When testing this idea, a simple ray casting isosurface renderer was written. This ray caster did not take into account anything involving the "big picture" and was easily integrated into the remote rendering engines. Another renderer was later tested that took advantage of graphics hardware and 3d textures.

4.2. Browsing

As stated before, one of the powers of this system is it's ability to browse and navigate through data while it renders. Figure 5 gives an example of looking around within the dataset while it renders. The use of the subimages as impostors allows for several types of navigation that do not require replacing the subimages already in the image cache. These types of navigation are camera rotation and zooming.

The reason zooming falls into this category is rather obvious. The system simply zooms in on the subimages. Camera rotation is also rather simple. This is not to be confused with data rotation, but rotating the camera direction while the camera location is fixed. This is only really useful when the location of the camera is very near, or actually within the

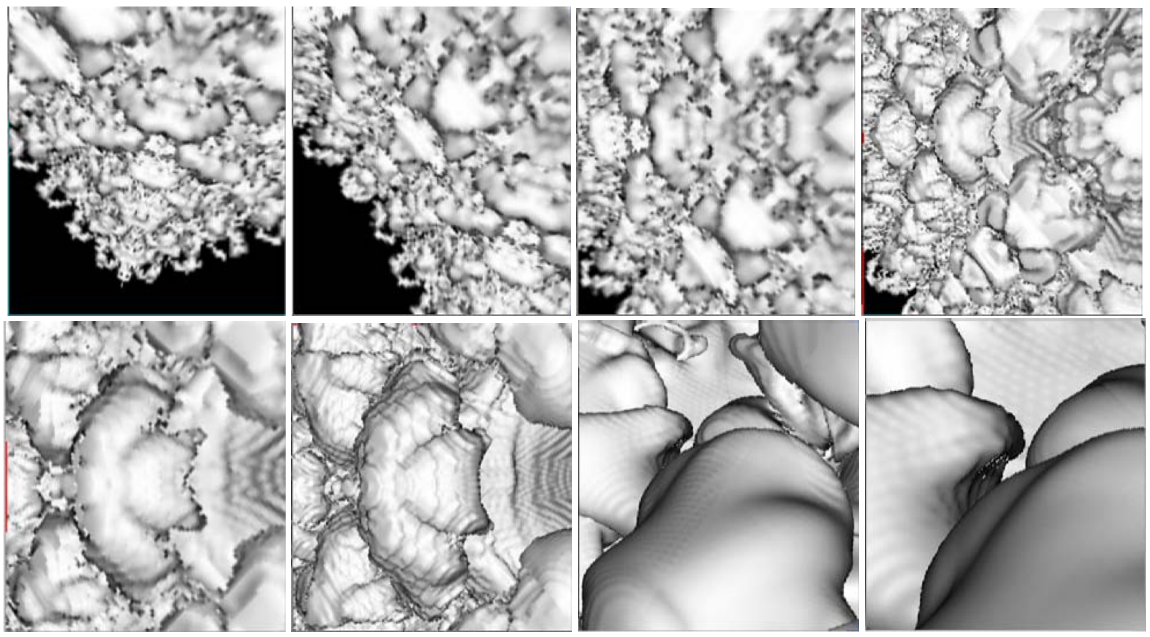

Figure 5: *An example of navigating through the dataset. The user first rotated the camera and increased the data resolution. The example follows the user as the camera is repeatedly zoomed and the data resolution increased again. The display remains interactive as these images are rendered.*

Rendering Engines	Total Time	Images per Second
16	85.5637	6.5382
24	58.4335	9.7204
32	44.3990	12.7930
40	35.6591	15.9286
48	31.7465	17.8917
56	27.4257	20.7105

Table 1: *A table showing numbers using the ray casting iso-surface renderer. The resulted image created was at near full data resolution.*

data itself. All other forms of navigation require the entire cache to be rerendered.

What this allows, is the user to locate possibly interesting information on the final image at low resolution. Then rotate that information to the center and zoom in. The user can now increase the number of subimages for this section by a large amount, allowing for a final image to be rendered at near full data resolution.

5. Testing

We have tested VMLIC on a 56-node cluster with a Quadrics switch interconnect. Each node has 2GB of memory and an Intel Xeon 2.8 GHz processor. Our tests were conducted using between 16 and 56 nodes. The dataset used in testing was generated from the Richtmyer-Meshkov instability simulation [MCC*02] with $2048 \times 2048 \times 1920$ voxels. ViSUS is commonly used in an out-of-core environment, because of this the data is read from a fileserver over the network, rather than stored on the local disk of each rendering engine.

For our performance study, frames-per-second does not work as an accurate measurement of this system because of the decoupling of the rendering and the display. Images are rendered remotely on the cluster and placed into the image cache. When the display draws to the screen it simply uses the images currently in the cache. As stated before this leads to the system as a whole being interactive, even though it may be actively rendering images. For the same reason, the size of display is also independent of the performance. Instead, we recorded the number of images rendered per second, and the number of images rendered per second per remote rendering engine.

The test itself corresponds to the system starting after dividing the scene into subimages 5 times. We then timed how long it took for the system to subdivide to a total of 10 subdivisions. The starting point for this test was used because in the early stages, only a few remote rendering engines can be used at a time(since the are only a small amount of images to be updated). This situation only occurs when the system starts, or the image cache is cleared. A total of 568 subimages were required to be rendered and transfered over the

Rendering Engines	Overhead	Images per Second
16	15.7298	36.1098
24	10.5001	54.0947
32	7.9486	71.4591
40	6.5462	86.7678
48	5.5466	102.4050
56	4.8330	117.5253

Table 2: *A table displaying the overhead of the system. This data was created by having the renderer return instantly, giving us an idea on how the actual system itself performs. The test involves the data loading and sending of 568 sub images.*

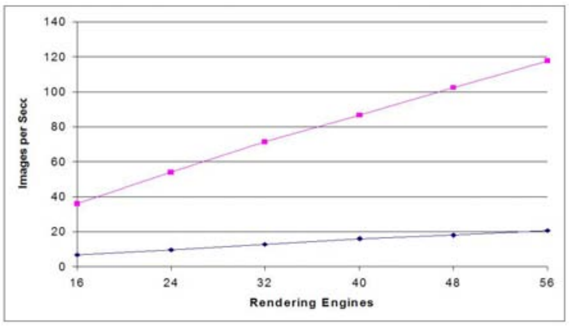

Figure 6: *A chart showing the number of images per second, relative to the number of remote rendering engines. The bottom line corresponds to the first full render test, while the top line corresponds to the second test with no rendering. This chart shows us that the system does in fact scale very well.*

network. The primary renderer used for this testing was a software ray casted isosurface renderer.

Table 1 shows the results when the system fully renders the scene. As you can see the total time to render the final image with fifty-six rendering engines was around 27.5 seconds. The renderer tested is not commonly used in large data visualization and is slow. These tests lead us to believe that the renderer we were using was a large bottleneck for the system. For this reason, we also felt that it was important to test the system without any actual rendering taking place. This would give us an idea of the actual performance of all the other components. Table 2 shows these results. Using fifty six rendering engines, the system completes the previous test in around 4.8 seconds or around 117 subimages updated per second. These value confirmed that our software render was indeed a large bottleneck. Our Preliminary testing of a 3D texture based hardware renderer finished the test in under 6.4 seconds, corresponding to around 89 subimages per second.

Engines	Images per Second per Engine
16	2.2568
24	2.2539
32	2.2330
40	2.6919
48	2.1334
56	2.0986

Table 3: *This table gives us an idea on how the system scales by looking at the number of subimages rendered per second for each rendering engine.*

5.1. Scalability

An important aspect of all distributed system is how the system scales when more processors are used. Figure 6 gives us a view of the number of images per second, relative to the number of remote rendering engines. This graph shows us that the system scales well enough that a simple graph like this cannot display how much performance we are losing.

We calculated another value to help us view the scalability. Before, we kept track of the number of sub images rendered per second and the number of rendering engines. From this we find the number of subimages rendered per second, per generator. This gives us an idea of how much work each rendering engine is doing relative to the number of total engines. Table 3 shows us these values for a range of 16 to 56 total rendering engines.

The resulting graph and table show us that the scalability is not perfect, which of course is expected. We feel that the performance loss is relatively minor and mainly due to the fact that sometimes there are fewer images that need to be rendered, than there are rendering engines. This is true during early parts of the rendering, and even partially true during medium quality renderings when dealing with a large number of remote rendering engines. During actual browsing, and when dealing with actual camera movement, we expect that the scalability to be even better.

A test was also performed to check for a possible performance bottleneck due to the network traffic. This test was conducted without the loading of data or the rendering of images. It simply tests how long it takes to send all the required images over the network. Forty eight rendering engines were used and a total of 568 128x128 images were transfered to build a final image that would be at near full data resolution. This test took 0.3821 seconds. This lets us believe that the network itself is not a large bottleneck and not likely a scalability issue for the number of engines we used.

6. Conclusion and Future work

We have introduced the VMLIC system. This system is capable of interactively browsing large multiresolution datasets through the use of image caching. We are able to update the cache in a timely manner by distributing the work over many remote rendering engines. This system does not depend on the actual rendering algorithm used, and is capable of integrating algorithms that would otherwise not be able to render large data.

More testing regarding the VLMIC system can also be done. More renders can be tested as well as testing how VMLIC can be adapted to clusters based on the power of each node. VMLIC contains several variables that can be modified based on the hardware. These variables include how much data is loaded to render each subimage along with the size of the actual subimage. As stated before, this also includes the actual rendering algorithm. As a user's system changes, VMLIC can possibly change along with it to maintain the level of interactivity desired.

The idea behind renderer independence has opened a door for new research. Taking a rendering method incapable of high resolution data and adapting it to work with our system is a definite next step. This can include very specific algorithms used by scientists who are currently forced to use the renderers included with other large data visualization systems. Renderers in these systems may not give them the answers they need.

The idea behind image caching and breaking a large multi-resolution problem into a small single resolution problem has more applications than just scientific visualization. We feel that this idea could be expanded to handle not only data, but high resolution models or meshes.

Acknowledgments

This work has been sponsored in part by the U.S. National Science Foundation under contracts CCF 0301194 (PECASE), CCF 0222991, OCI 0325934 (ITR), IIS 0552334 and the U.S. Department of Energy under Memorandum Agreement No. DE-FC02-01ER41202 (SciDAC) and DE_FG02-05ER54817 (SciDAC).

References

[ALS*00] AHRENS J., LAW C., SCHROEDER W., MARTIN K., PAPKA M.: A parallel approach for efficiently visualizing extremely large, time-varying datasets. Technical Report LAUR-001630, Los Alamos National Laboratory, 2000.

[AMC05] AKIBA H., MA K.-L., CLYNE J.: End-to-end data reducation and hardware accelerated rendering techniques for visualizing time-varying non-uniform grid volume data. In *Proceedings of the International Workshop on Volume Graphics* (June 2005), pp. 31–39.

[EEH*00] ENGEL K., ERTL T., HASTREITER P., TOMANDL B., EBERHARDT K.: Combining local and remote visualization techniques for interactive volume rendering in medical applications. In *IEEE Visualization* (2000), pp. 449–452.

[GS04] GUTHE S., STRASSER W.: Advanced techniques for high-quality multiresolution volume rendering. Computers & Graphics, 28(1):51–58, February 2004.

[GWGS02] GUTHE S., WAND M., GONSER J., STRASSER W.: Interactive rendering of large volume data sets. In *VIS '02: Proceedings of the conference on Visualization '02* (Washington, DC, USA, 2002), IEEE Computer Society, pp. 53–60.

[LHJ99] LAMAR E. C., HAMANN B., JOY K. I.: Multiresolution techniques for interactive texture-based volume visualization. In *IEEE Visualization '99* (San Francisco, 1999), Ebert D., Gross M., Hamann B., (Eds.), pp. 355–362.

[LP03] LAMAR E., PASCUCCI V.: A multi-layered image cache for scientific visualization. In *Parallel and Large-Data Visualization and Graphics, 2003. PVG 2003. IEEE Symposium on* (2003), pp. 61–67.

[MCC*02] MIRIN A., COHEN R., CURTIS B., DANNEVIK W., DIMITS A., DUCHAINEAU M., ELIASON D., SCHIKORE D., ANDERSON S., PORTER D., P.R. WOODWARD L. S., WHITE S.: *R2 Data Set âĂŞ Very High Resolution Simulation of Compressible Turbulence*. Tech. Rep. LLNL Report UCRL-MI-151066, Lawrence Livermore National Laboratory, 2002.

[Pas04] PASCUCCI V.: ViSUS (visualization streams for ultimate scalability) project page, 2004. httP://www.pascucci.org/visus/index.html.

[PHKH04] PROHASKA S., HUTANU A., KÃĎHLER R., HEGE H.-C.: Interactive exploration of large remote micro-ct scans. In *Visualization, 2004. IEEE* (2004), pp. 345–352.

[PLF*03] PASCUCCI V., LANEY D. E., FRANK R. J., SCORZELLI G., LINSEN L., HAMANN B., GYGI F.: Real-time monitoring of large scientific simulations. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing* (New York, NY, USA, 2003), ACM Press, pp. 194–198.

[SML96] SCHROEDER W., MARTIN K., LORENSEN. W.: *The Visualization Toolkit An Object Oriented Approach to 3D graphics*. Prentice Hall, 1996.

[WGS04] WANG C., GAO J., SHEN H.-W.: Parallel multiresolution volume rendering of large data sets with error-guided load balancing. In *Eurographics Parallel Graphics and Visualization '04* (2004), pp. 23–30.

[WWH*00] WEILER M., WESTERMANN R., HANSEN C., ZIMMERMANN K., ERTL T.: Level-of-detail volume rendering via 3d textures. In *VVS '00: Proceedings of the 2000 IEEE symposium on Volume visualization* (New York, NY, USA, 2000), ACM Press, pp. 7–13.