

**VOLUME RENDERING OF LARGE DATA FOR SCALABLE DISPLAYS  
USING PHOTONIC SWITCHING**

BY

SHALINI VENKATARAMAN

B.Sc, Computer Science, National University of Singapore, Singapore, 1996

PROJECT

Submitted as partial fulfillment of the requirements  
for the degree of Master of Science in Computer Science  
In the Graduate College of the  
University of Illinois at Chicago, 2004

Chicago, Illinois

## TABLE OF CONTENTS

<b><u>CHAPTER</u></b>		<b><u>PAGE</u></b>
<b>1.</b>	<b>INTRODUCTION AND MOTIVATION</b> .....	<b>5</b>
1.1	Large Data .....	6
1.2	Advances in commodity graphics .....	7
1.3	Scalable Displays .....	7
1.4	High-speed Networks .....	8
<b>2.</b>	<b>BACKGROUND WORK</b> .....	<b>11</b>
2.1	Parallel volume rendering .....	11
2.2	Scalable displays .....	13
2.3	Distributed data access .....	14
2.4	Volume Roaming .....	14
<b>3.</b>	<b>VISUALIZATION APPROACH</b> .....	<b>16</b>
3.1	Data service - Optistore .....	17
3.1.1	Data Management .....	18
3.1.2	Processing .....	18
3.1.3	Transport .....	19
3.2	Rendering .....	20
3.2.1	Isosurface visualization .....	20
3.2.2	Slicing .....	21
3.2.3	Texture-based volume rendering .....	21
3.2.4	Classification .....	23
3.3	User Interaction .....	25
3.3.1	Volume exploration .....	26
3.3.2	Clipping .....	26
3.3.3	Interactive isosurface/point extraction .....	26
3.4	Display .....	27
3.4.1	AGAVE – passive stereoscopic system .....	27
3.4.2	Perspectile – tiled display .....	29
3.5	Transfer function specification .....	31
3.5.1	The User interface .....	33
<b>4.</b>	<b>SYSTEM DESIGN AND IMPLEMENTATION</b> .....	<b>35</b>
4.1	Optistore .....	36
4.2	Vol-a-Tile .....	36
4.3	A roam example .....	36
4.3.1	Disk-Memory load .....	37
4.3.2	Subvolume extraction .....	37
4.3.3	Network data transfer .....	38
4.3.4	Histogram operations .....	38
4.3.5	Memory-Texture download .....	38
4.3.6	Render .....	38
4.4	Transfer function editor .....	39
<b>5.</b>	<b>RESULTS AND BENCHMARKS</b> .....	<b>40</b>
5.1	Disk-Memory load .....	41
5.2	Subvolume extraction .....	42

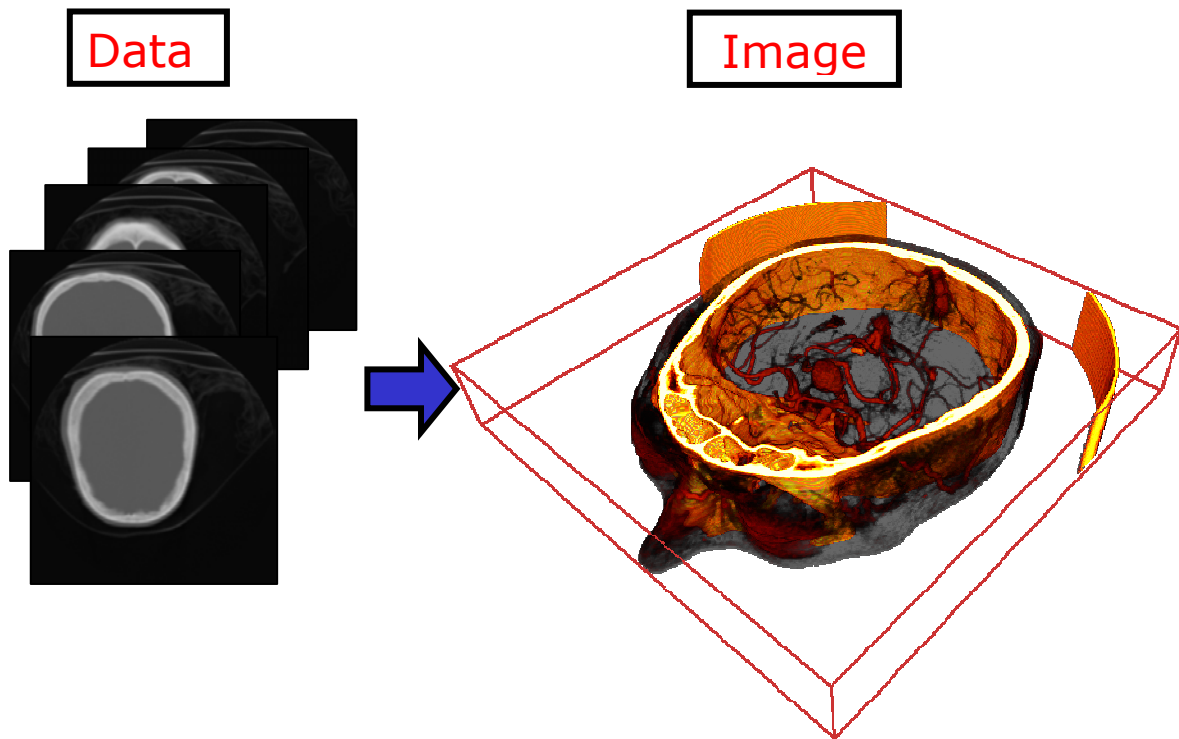
5.3	Network data transfer .....	43
5.3.1	Local.....	43
5.3.2	Cluster .....	44
5.3.3	Remote .....	45
5.4	Histogram operations .....	46
5.5	Memory-Texture download.....	47
5.6	Render .....	48
<b>6.</b>	<b>APPLICATIONS.....</b>	<b>49</b>
6.1	Neuroscience .....	49
6.2	Seismic Reflectivity experiments.....	51
6.3	Seismic wave propagation.....	53
<b>7.</b>	<b>CONCLUSION AND FUTURE WORK.....</b>	<b>55</b>
	<b>CITED LITERATURE.....</b>	<b>56</b>

## **SUMMARY**

In this report, we introduce a flexible and scalable volume visualization system for visual browsing of spatially and temporally large datasets in distributed computing environments. Our framework leverages advances in high-speed networks for access to shared scalable computation, data storage and graphics. In particular, we assume commodity off the shelf graphics hardware for both, rendering and data processing. We describe the implementation and integration of the various pipelines within our visualization framework and present examples of applications in seismology and biological tomography. The quintessential goal of this research is to create an adaptive visualization framework that is cognizant of the constraints in the various pipelines and accordingly, optimizes the visualization process dynamically.

## 1. INTRODUCTION AND MOTIVATION

Traditionally, computer graphics algorithms expect a scene description in form of geometric objects which then is used to generate suitable images through rendering. Volume visualization, on the other hand, is concerned with the representation, manipulation and display of volumetric data, typically represented by a 3D grid of scalar values. Volume rendering is the process of projecting this 3D grid onto a 2D image plane to gain an understanding of the structure contained within the data. Optical properties such as color and transparency may be assigned to the input scalar values by means of transfer functions. As shown in Figure 1 below, the advantage gained by a volumetric representation is, that objects have information inside of them. This allows us to render amorphous phenomena or structures which are impossible to specify geometrically.

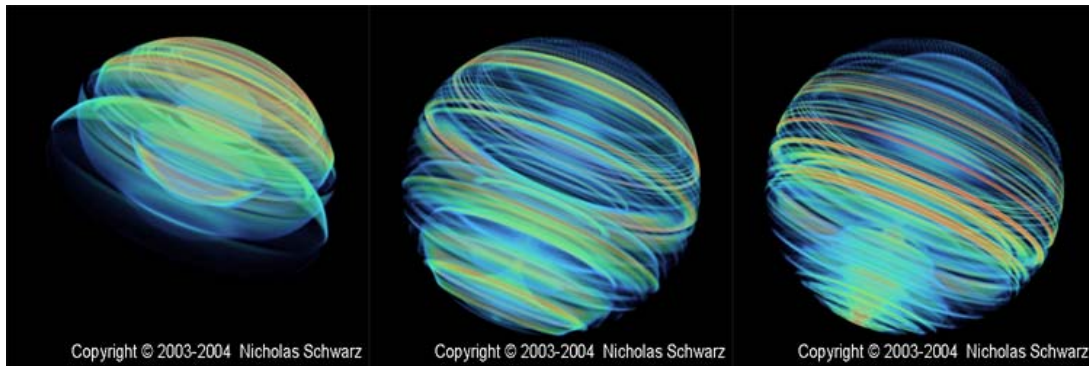


**Figure 1: Illustration of the Volume rendering process. The stack of 2D slices on the left constitute the 3D grid and the resulting image on the right is a projection of this grid on a 2D plane.**

This section explains some of the motivation and challenges that give rise to our project and introduce the approach taken.

## 1.1 Large Data

Volume data sets most commonly occur in two fields: imaging and computational science. 3D imaging devices continue to increase the resolution of their sampled volumes with the current generation approaching a gigavoxel. Similarly, computational science continues to increase the mesh resolutions for large scale simulation thereby increasing the size of the data to be visualized. For instance, by appropriately rendering a time-varying dataset, scientists can produce an animation sequence that illustrates how selected underlying structures in the data evolve over time. Figure 2 shows selected time steps from the visualization of the seismic wave propagation from a simulation of an earthquake in Bolivia. It demonstrates the ability to derive meaningful information such as the direction of wave propagation from the data.



**Figure 2 : A sequence of frames from the wave propagation simulation of the Bolivia earthquake. We see how the p-wave hits the earth surface and reverberates.**

Single-system visualization software running on high-end commodity machines can no longer sustain interactive browsing of these large data due to their limited I/O and processing

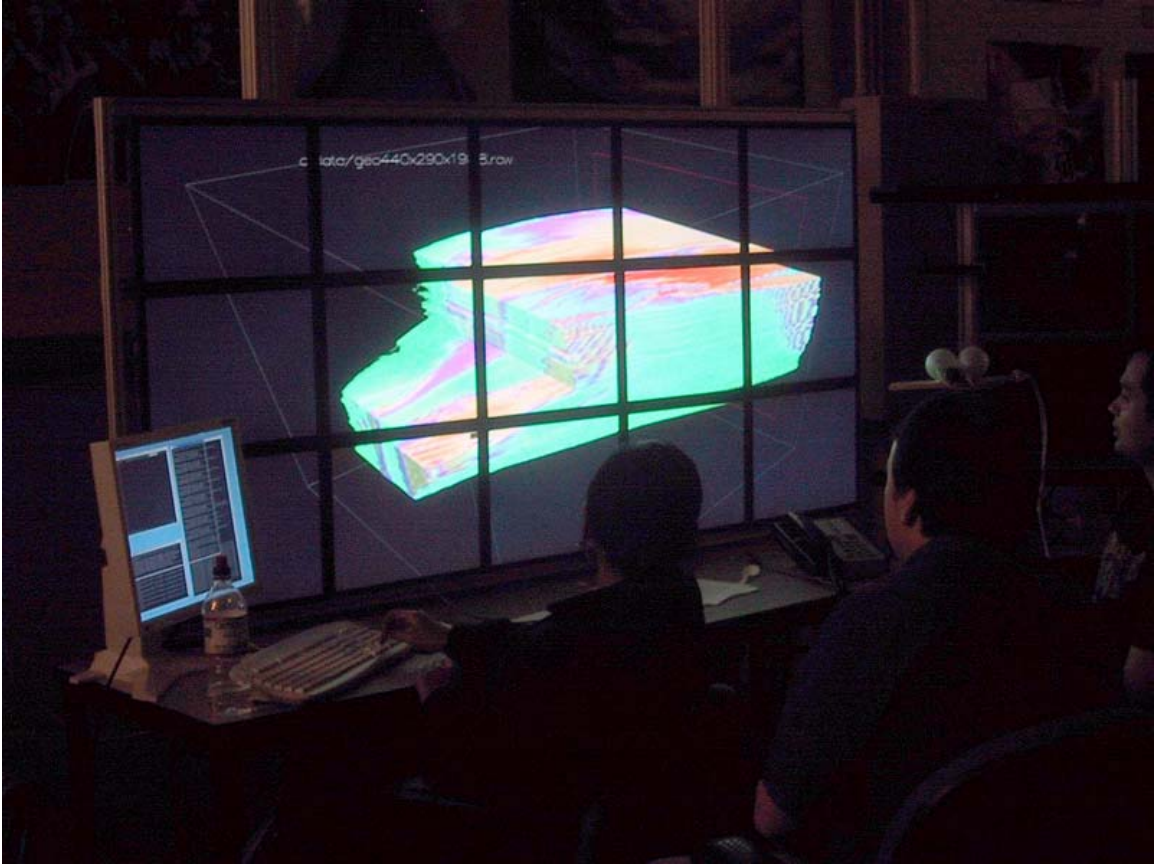
capabilities. A distributed, adaptive and scalable approach is needed for data access, processing, rendering and display of these spatially and temporally large datasets.

## **1.2 Advances in commodity graphics**

In recent years, the market for high end PC graphics accelerator boards has grown explosively, mostly driven by the computer gaming and entertainment industry. As a result, graphics chips like the NVidia GeForce4 or ATI Radeon 8500 are not only competing with professional graphics workstations but often surpass the ability of such hardware. The new programmability offered by graphics hardware allows many visualization algorithms to be mapped efficiently onto hardware. Specifically, hardware support for 3D texture mapping and dependant textures has aided greatly in volume visualization - a field that has very high computation and resource requirements. The parallelization of such boards to a render cluster provides us even more scalability. However, from the application development point of view, the variety of different extensions that give access to the hardware features are at a low abstraction layer making it hard to ensure compatibility across all graphics boards. The introduction of high-level shading languages such as NVidia's CGGL[1] raises the abstraction making GPU-based rendering less daunting for the application programmer.

## **1.3 Scalable Displays**

Using large, high-resolution displays is another trend that is currently emerging. High resolution allows for detailed scientific visualization, and overcomes the limited screen resolution of standard monitors. Large displays also have applications in physical collaborative environments, where multiple people are looking at a single large screen as shown in Figure 3 below.



**Figure 3 : A volume rendered image of a sediment basin on EVL's 6000x3000-pixel LCD tiled display, called the PerspecTile driven by a cluster of PCs. (now dubbed the GeoWall2 by the Geoscience community)**

#### **1.4 High-speed Networks**

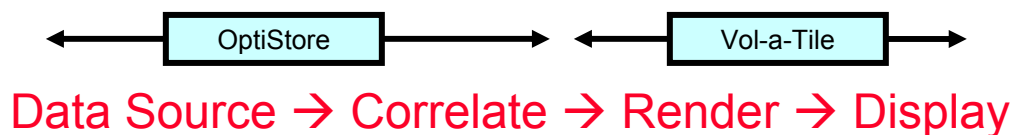
The availability of hardware support for volume visualization enables real-time rendering. However the sheer size of a data set from a contemporary scientific simulation can easily overwhelm the memory space of commodity graphics cards. With huge datasets, the time needed to alternately load the bricks of volumes into graphics alone can inhibit interactive rendering. Time-varying data exacerbates this problem as data needs to be read from disk to main memory before downloading to the graphics subsystem. Thus, the limiting factor for time-varying datasets



is the high-performance I/O. High-speed networks linked to large storage pools may alleviate this bottleneck as it obviates the need to continuously load data files from disk to memory.

The OptIPuter project [2] is one such attempt to interconnect distributed storage, computing and visualization resources using a backplane constructed from a grid of deterministic high speed networks. In partnership with the Scripps Institute of Oceanography and the Biomedical Informatics Research Network, one of the application goals of the OptIPuter project is to develop a distributed, scalable and adaptive framework to support visualization of very large time-varying volumetric data in the Geosciences and the Neurosciences.

Figure 4 below shows the pipeline of visualization in this model. In this pipeline, data (usually large collections of real or simulation data) are fed into a data correlation or filtering system which generates a sub-sampled or summarized result from which a visual representation is created and displayed on laptops, tiled displays, and even immersive CAVEs. In some cases several components of this pipeline may need to share a single compute cluster- for example it may be more economical in some cases to combine the data source with the data correlation component.



**Figure 4: The visualization pipeline in the OptIPuter model**

Based on the OptIPuter concept, we have developed an interactive distributed volume roaming system that simulates the way one traditionally looks at large data. Our system is composed of 2 main subsystems - Optistore and Vol-a-Tile. In the context of the OptIPuter model, Optistore implements the data source and correlate component while Vol-a-Tile encompasses the render

and display aspects. An optional third component is the transfer function editor, tfUI that allows for interactive manipulation of color and opacity for the volume data.

We are concerned with roaming a large volume interactively rather than trying to visualize the volume as a whole. Our approach focuses on using Vol-a-Tile to roam at interactive frame rates through 3D volumes using dedicated photonic networks to stream these time-varying datasets from the clustered data service – Optistore. The volumes typically do not fit into graphics memory or in the future, even into main memory. Users browse through data sets by moving around slices and volume rendering lenses. They traverse the low-resolution volume to take a closer look at local phenomena that is rendered in higher-resolution. Interactive frame rates are an important issue in this context and our system gives users the possibility to dynamically trade rendering quality for speed. Vol-a-tile emphasizes the scalability in displaying and can render to a single display, passive stereoscopic projection systems such as the AGAVE [3] or tiled-displays. In designing this system, we have also incorporated various current algorithmic researches in using commodity graphics hardware specifically in the use of transfer functions.

## 2. BACKGROUND WORK

### 2.1 Parallel volume rendering

The most prevalently used technique for parallel volume rendering is the use of sort-last [4] methods. The volume is statically partitioned and the sub-volumes rendered in parallel. These sub-images are then read back, composed from back-to-front in hardware or software and finally pasted on a single display. The underlying assumption is that a single display is sufficient for display of this data. The assumption is somewhat valid since a resolution of 1600x1200 suffices for a gigavoxel volume. The bottleneck with this approach however, is the framebuffer read-back and composition latency that occurs at every frame. Figure 5 shows the test bed for our sort-last experiments.

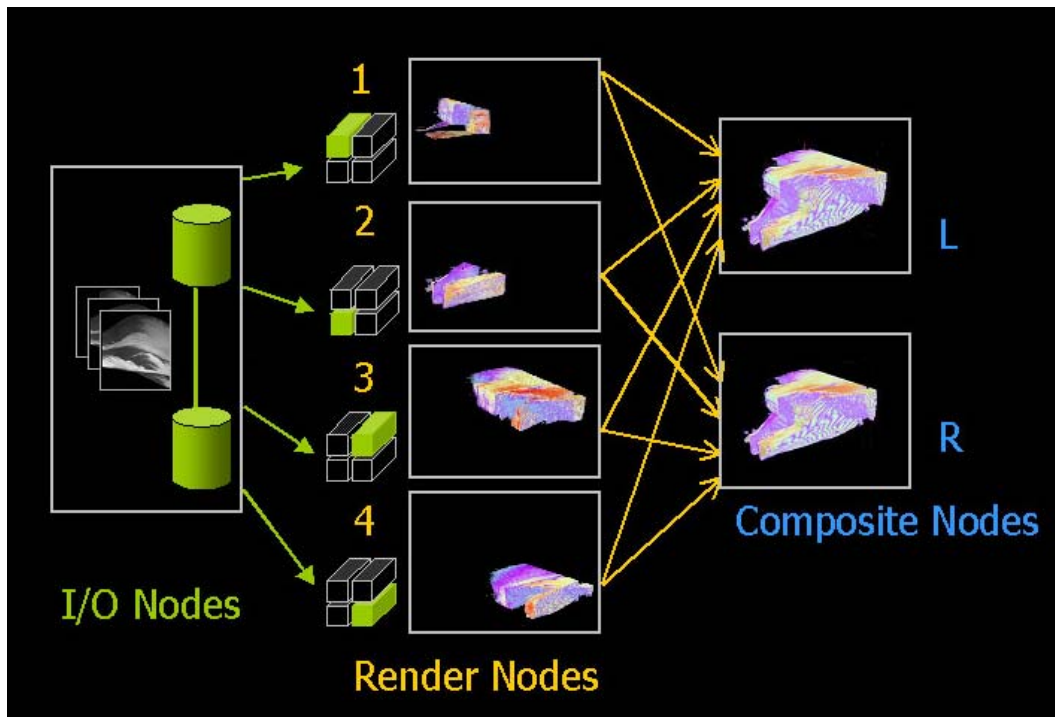
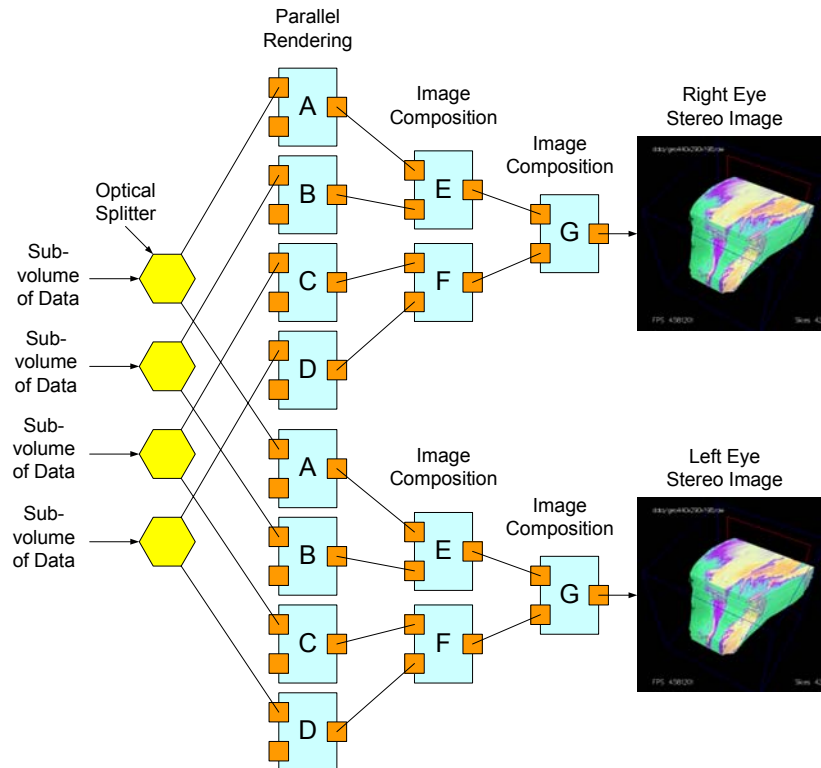


Figure 5 : Sort -last rendering testbed on the cluster at EVL. Data is divided into 4 subvolume, each sent to rendering nodes, read back and composited on the display

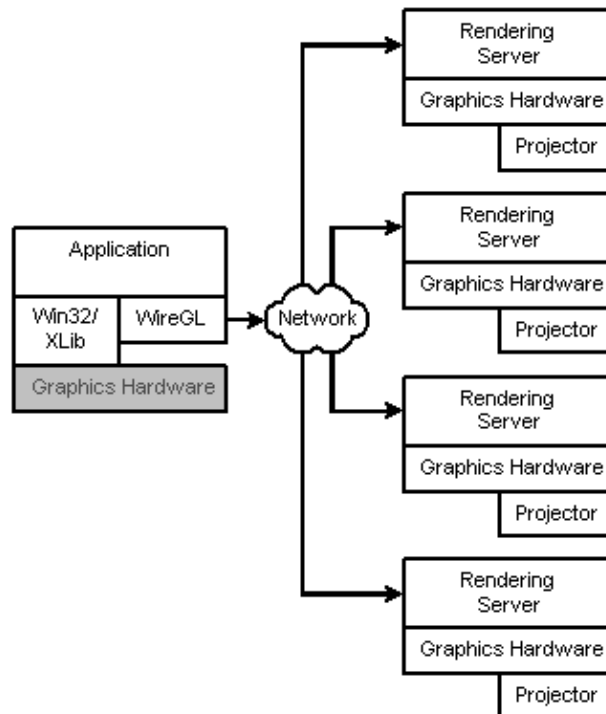
Kniss[14] developed a software-based approach called TRex using SGI IR-2 graphics hardware. Bajaj[15] implement this on a commodity cluster with NVidia GeForce3 processors. To overcome composition latency present in both the cases above, Lombeyda et al [17] show the use of Sepia-2 pro architecture and the VolumePro 500 ray casting engine [16] to render volumetric data that has already been partitioned. The images of these subvolumes are blended concurrently in hardware. They are able to visualize  $512^3$  (1/8 of gigavoxel) at 24fps. The drawback of this approach is that it is not intrinsically scalable due to the extra hardware that is needed. The maximum data size used in all the above cases was  $512^3$ . Leigh [5], in the context of the OptIPuter project, addresses the I/O bottleneck using photonic computing to allow all rendering nodes to freely access the dataset. Figure 6 below shows his approach.



**Figure 6 : The use of photonic multicasting to allow multiple render nodes to access the same subvolume.**

## 2.2 Scalable displays

The idea here is to scale the display resolution for a fixed data size. As the fragment has to be drawn to a larger display area, more time is spent doing hardware-interpolation which can lead to fill rate limitations. Chromium [18] developed at the Stanford University Computer Graphics Lab provides a way to distribute OpenGL graphics by replacing the standard OpenGL dynamically-linked library on the client by its own library. This new library, instead of executing the OpenGL commands, sends these to the appropriate rendering servers for display as shown in Figure 7. Chromium has the advantage of being compatible with all existing OpenGL programs. However, it requires that all geometry be moved over a network every frame resulting in poor scalability as the master becomes the bottleneck.



**Figure 7: Shows the Chromium framework where the application resides on 1 node (to the left) and streams geometry to the rendering servers across the network. Only the application node has access to the data files.**

Aura [19] presents a solution to this scalability problem, by using retained-mode graphics instead of immediate mode for rendering on a tiled display. So, the data is only transmitted when it is changed. The consequence is that its performance for objects that are more or less static during execution is much better than for Chromium.

Both Aura and Chromium are not designed to handle dedicated data access. The controlling application is still the central repository for data and the only conduit of communication with data stores. Ideally, we would like every node in the renderer to be able to communicate with a data service so as to parallelize the I/O streaming. The actual data could be residing on a single machine or on a cluster. However, these details are transparent to the renderer.

### **2.3 Distributed data access**

The Grid Visualization Utility/Grid Data Transport toolkit (GVU/GDT)[20] provides a framework that exploits distributed parallelism to scale the input data-handling (i.e data accessing and data processing) capabilities in a GRID [21] computing environment. This model assumes an active storage model where scientific data are relatively local to CPUs, i.e., local data access scales with the computational capability. We augment the functionality of GVU/GDT to also scale up the information viewing capabilities by parallelizing graphics and display hardware for large tiled displays.

### **2.4 Volume Roaming**

There has been considerable research of late in the use of multi-resolution and level-of-detail (LOD) approaches for roaming through large volumes. Adaptive schemes are used to render the volume in a region-of-interest at a high resolution and the volume away from this region at progressively lower resolutions [8, 9 ]. Li [10] extends this to automatically select appropriate LODs for different regions of the volume to guarantee a user specified frame rate. However, the underlying assumption in all the above cases is that the data is small ( $256^3$ ), locally residing so

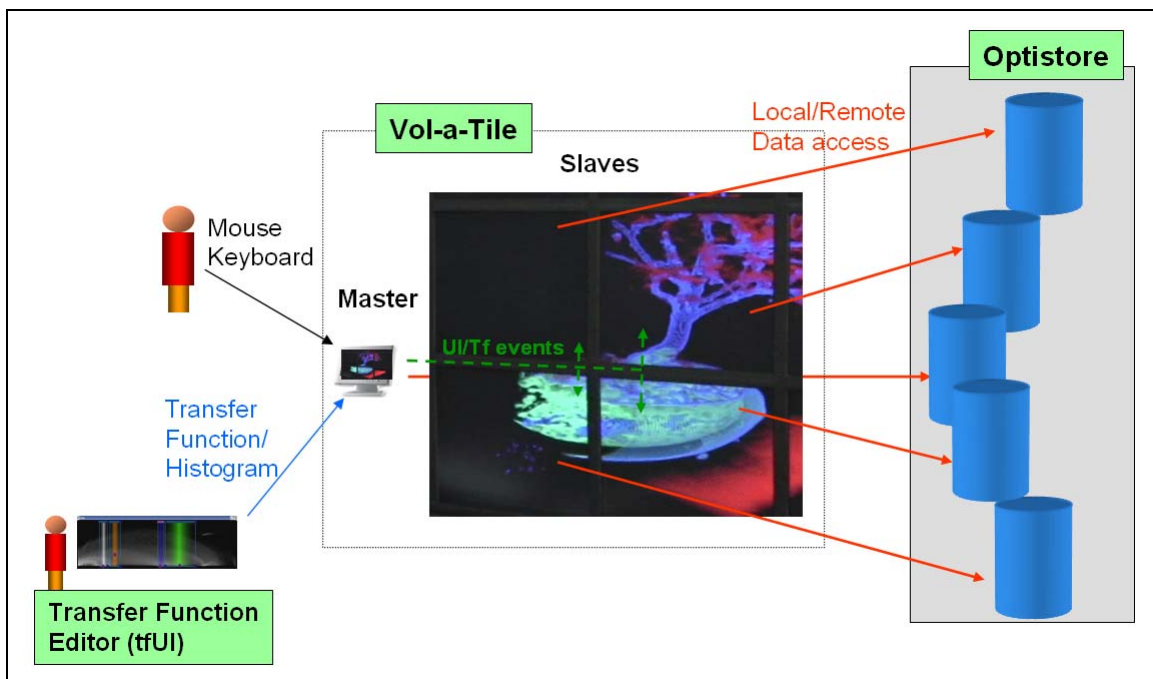
that the I/O time for loading data need not be taken into account; and displayed on a single monitor. Plate [11] has developed a hierarchical approach for paging and prediction strategies using frame-to-frame coherence to handle roaming through very large volumetric data (16GB). Again, local data storage and a single display resolution are assumed which does not deal with fill rate limitations. Nevertheless, in future, we hope to incorporate these multi-resolution methods into our distributed visualization framework.

### 3. VISUALIZATION APPROACH

In this section, we conceptually describe the functionality of the 3 components of our system

- Data access and processing - Optistore,
- Rendering, interaction and display – Vol-a-Tile
- Transfer function specification - tfUI.

The next section deals with the implementation specific details. Figure 8 shows a schematic of the system. Each stage is an independent process capable of running in a distributed manner.

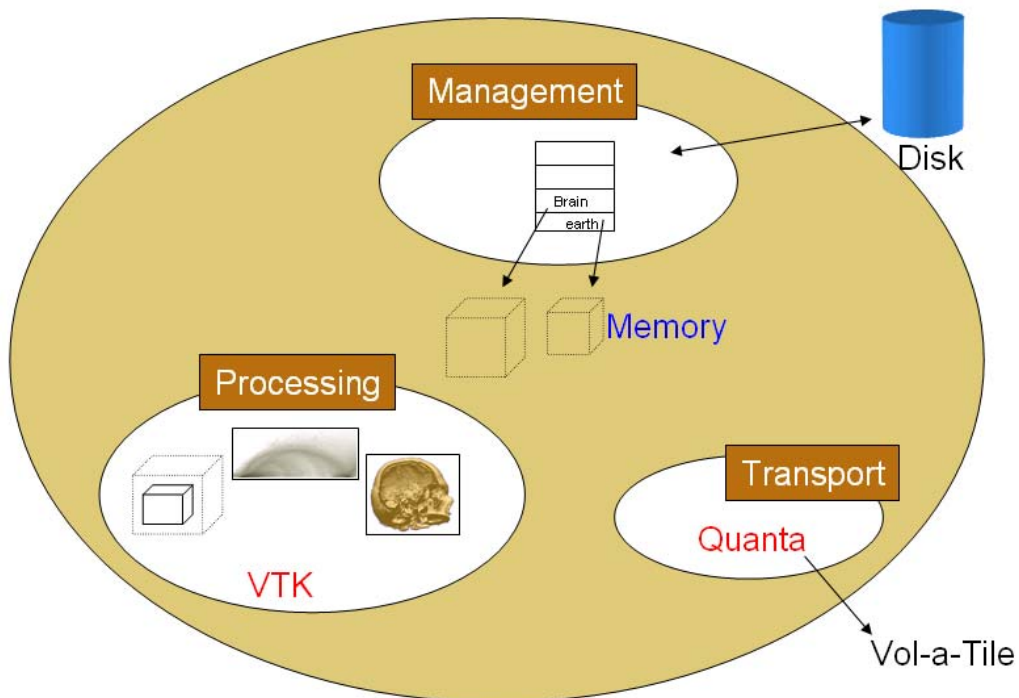


**Figure 8 : Schematic of our visualization framework. There is a link between each of the individual nodes in the storage (Optistore) and the visualization (Vol-a-Tile) cluster.**



### 3.1 Data service - Optistore

The first stage of data visualization involves reading a time step from disk. Optistore is the datasever that provides the high-performance I/O needed to stream time steps to the renderers memory and then to the graphics. Data objects are stored as 3D volumes or geometry. Optistore is designed to handle common visualization dataset operations, including data management, processing, representation and transport. A useful starting point for creating a parallel visualization system to support these needs is to build upon an existing visualization system - the visualization toolkit (VTK) [29] in our case. VTK contains many serial visualization, graphics and imaging algorithms and is portable to a variety of hardware platforms and operating systems. Optistore uses the existing filters in VTK like the marching cubes, gradient calculations, as well as volume cropping and down sampling functions. The functions of Optistore are shown in Figure 9 below.



**Figure 9 : Optistore components – Data Management, Processing and Transport**

### 3.1.1 Data Management

- **Loading** - Volume data can be loaded as uncompressed raw binary or compressed using Run Length Encoding or gzip. Geometry can be loaded as triangle strips.
- **Meta-information** – Optistore also maintains the meta-information of the datasets such as dimensions, voxel scaling, bytes per sample etc. A client can access the specific dataset using its symbolic name.
- **Storing** – Results of data operations such as resampling and cropping can be stored and assigned a symbolic name at run-time.

### 3.1.2 Processing

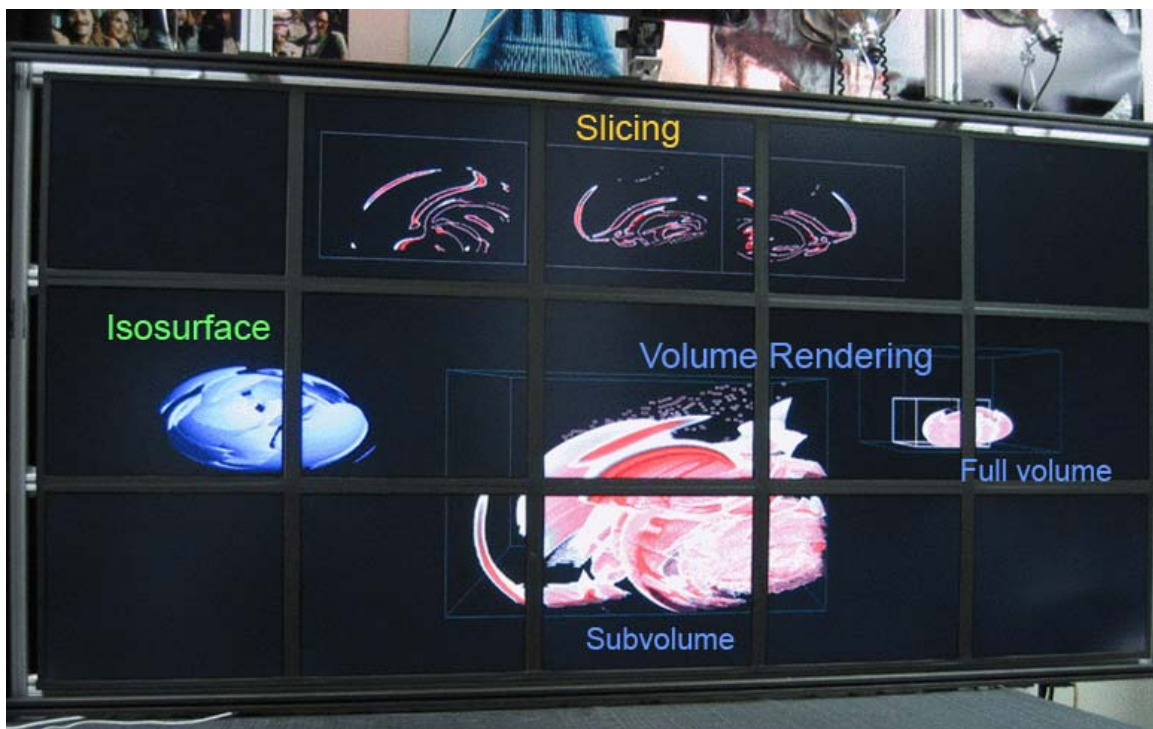
- **Gradient/Histogram** – Optistore does gradient computation of the volume on the fly as opposed to maintaining a separate copy of the gradient volume. The gradient volume is used to generate the 2D Histogram which is then used by the transfer function widget and in future for lighting and shading calculations. Optistore also maintains volume statistics such as mean, min and max.
- **Isosurface/Isopoint extraction** - Often an understanding of the distribution of density within the volume can be obtained by deriving an isosurface. That is, a 3D contour is created at a selected density. This often works well for volumes with strong and obvious internal structure, like for example, showing the bone from an MRI scan.
- **Sampling** – The data can be resampled to a specified resolution at run-time. This can be used when the user has picked a 3D region of interest and needs to zoom in to a higher resolution of 3D data.
- **Cropping** – This operation is used in roaming the data. A subvolume for a range specified by the user is extracted from the dataset.

### **3.1.3 Transport**

Optistore supports a variety of network protocols such as Photonic reservation, TCP and Reliable Blast UDP [30]. These protocols can be selected depending on the type of network connection.

### 3.2 Rendering

Once the data has been accessed and processed accordingly, the rendering process is the next stage. There are three main approaches to the visualization of volumetric data: isosurfacing, slicing and volume rendering. Figure 10 below illustrates these approaches on a snapshot of the visualization.



**Figure 10: Illustrates the three main approaches to volume visualization – Isosurface, Slicing and Volume rendering**

#### 3.2.1 Isosurface visualization

A surface shell is extracted from the data, containing points with a common value for the data functions. Assuming that  $F$  is a function that defines the 3D data, here we are visualizing the set of points  $(x,y,z)$  such that

$$F(x,y,z) = k, \text{ for some threshold value } k.$$

This surface is typically approximated as a triangular mesh which can then be passed as geometry to a rendering process. If time is used as an additional display dimension, then we can visualize the entire dataset by sweeping the threshold through the range of data

### **3.2.2 Slicing**

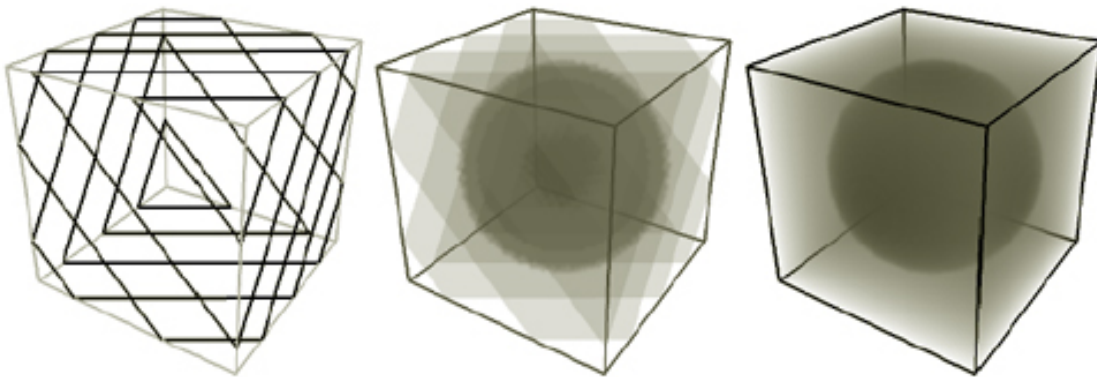
A two-dimensional slice is taken through the data (often parallel to the co-ordinate planes) or in user-controlled arbitrary orientation as in a cut-plane allowing one to probe the voxel in 2D space. Again, using time as an additional display dimension, we can sweep the slice plane through the data in order to gain a visualization of the full datasets. Although slicing reduces the problem to a 2D visualization space, it remains a very valuable technique and in many cases, users are better able to comprehend 2D information than 3D.

### **3.2.3 Texture-based volume rendering**

Volume rendering offers a more complete solution to volume visualization, in that it aims to picture the entire volume as is. It has been traditionally thought of as more computationally expensive than surface extraction, but this view has recently been challenged by recent hardware developments. The most attractive aspect of direct volume rendering is actually its defining characteristic: it maps directly from the dataset to a rendered image without any intermediate geometric calculations.

A 3D volumetric object is a collection of a large number of voxels analogous to pixels in a 2D image. This data is visualized by sending rays from the eye through each pixel of the image plane through the volume and sampling the data with a continuous sampling rate(ray-casting). This volume data can be defined by 3D textures in hardware and the parallel texturing hardware utilized to perform reconstruction and resampling on polygons embedded in the texture.[6,7]. The volume data itself is stored in a single 3D texture, where a single texel corresponds to a single

voxel. At the core of the algorithm multiple planes (determined by the sampling rate) parallel to the viewing plane are clipped against the volume bounding box. Compositing these individual planes can easily be done by exploiting hardware alpha blending to approximate the ray casting integral. The planes referred to as proxy geometry are rendered as texture-mapped quads), and compositing all the parts (slices) of this proxy geometry from back to front via alpha blending. Additionally, resampled data on each polygon can be transformed into color and opacity values using a transfer function and composited into the frame buffer. Since this whole process is supported by specialized graphics hardware, the time it consumes decreases considerably compared to a software implementation. Thus, interactive frame rates can be achieved. Figure 11 shows the process.



**Figure 11 : View-aligned texture slicing and blending**

As seen in the figure above, the number of slicing planes or the sampling rate directly determines the rendered image quality. There is a distinct trade-off between the sampling rate and rendering performance vs the rendered image quality. This gives users a degree of flexibility in choosing the sampling rate that best meets their needs. In our visualization we use 3D textures to

show both - a low-resolution thumbnail of the whole volume and a high-resolution volume of interest. As explained later, the user can manipulate the subvolume to see different parts of the dataset.

### 3.2.3.1 Opacity Corrected Texture Slicing

To interactively control the rendering time and quality of the image, we allow the user to vary the sampling rate to allow arbitrary number of slicing planes through the data. In this situation, it is important to properly scale the alpha values of incoming slices to maintain the overall look of the volume regardless of the sample rate. The relationship between the sample rate and scaled alpha values is not linear [9] and is modeled by

$$\alpha_{\text{new}} = 1 - [1 - \alpha_{\text{old}}]^{\text{sr-old}/\text{sr-new}}$$

where sr-old is the sample rate used with  $\alpha_{\text{old}}$  and sr-new is the new sample rate used with  $\alpha_{\text{new}}$ . Although this computation is expensive, it is compensated for because the texture lookup tables are relatively small, 256 elements when compared to the volume data's size.

### 3.2.4 Classification

Direct volume rendering is based on the premise that the data values in the volume are themselves a sufficient basis for creating an informative image. What makes this possible is a mapping from the numbers which comprise the dataset to the optical properties that compose a rendered image, such as opacity and color. This critical role is performed by the transfer function. Because the transfer function is central to the direct volume rendering process, picking a transfer function appropriate for the dataset is essential to creating an informative, high-quality rendering.

A texture lookup table encodes the transfer function which assigns color and opacity values to the scalar data elements. The lookup table is stored in hardware as a 1D texture or a 2D texture (also with respect to the gradient). The pixel texture[24] OpenGL extension is used to apply this 1D/2D texture to the 3D volume. Pixel texture and dependent texture are names for operations which use color fragments to generate texture coordinates, and replace those color fragments with

the corresponding entries from a texture. This operation essentially amounts to an arbitrary function evaluation via a lookup table. The GeForce3 series platform supports dependent texture reads on a per-fragment basis in hardware. Figure 12 shows the interaction between the 3D Volume texture and the transfer function texture.

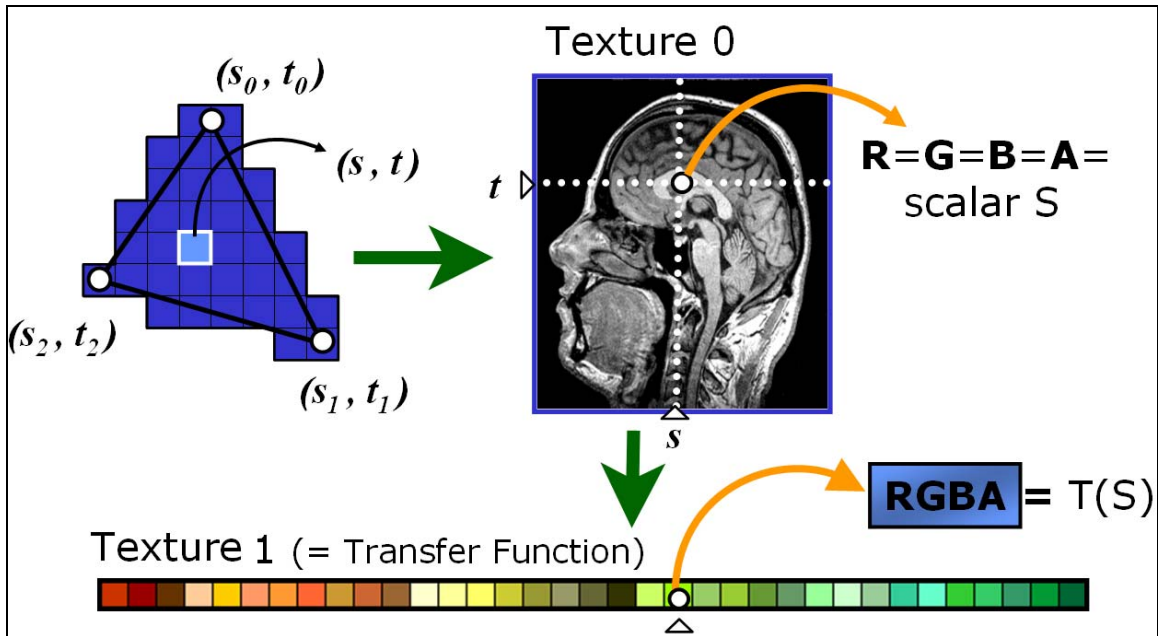
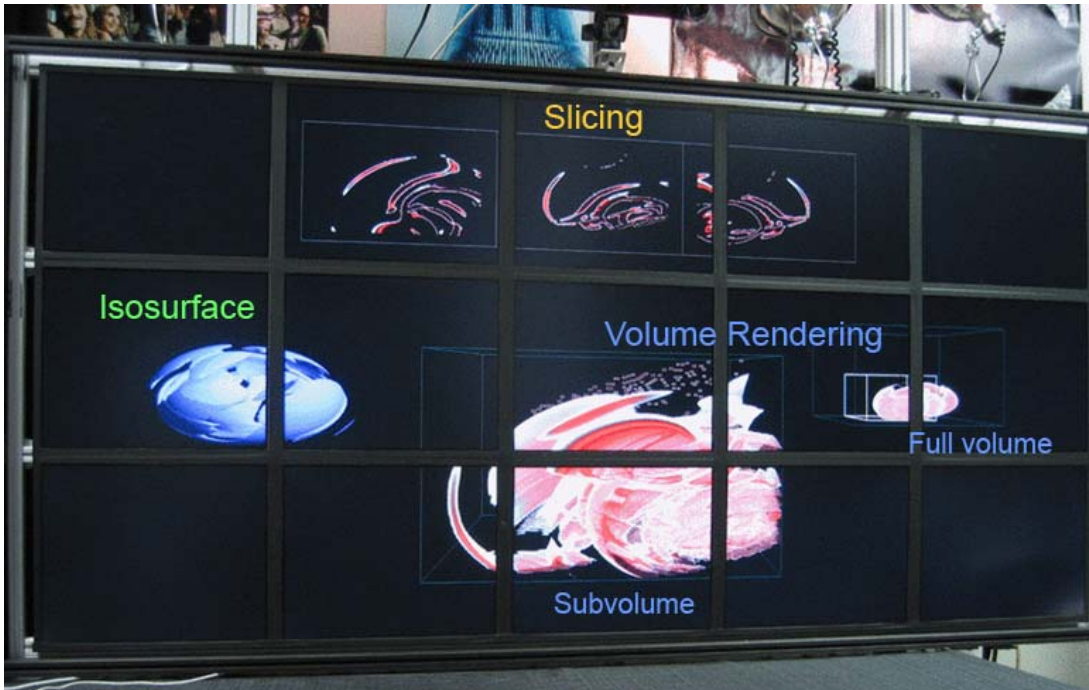


Figure 12 : The sequence of operations in the rasterization phase. The scalar  $S$  from the 3D texture for the texel is indexed into the transfer function (texture1) resulting in the final RGBA value.



### 3.3 User Interaction

Shown below is the snapshot of the application again so as to allow the user to correlate the interaction features to the visual interface.



Vol-a-Tile provides a “trackball” interface to interactively scale, rotate and pan any of the visual objects. The object to be manipulated is selected using a key-press. When the user is in interaction mode such as a pending rotation, the sampling rate is decreased to decrease the rendering time and allow interactivity. Once the interaction is complete ie when the user releases the mouse button, the sample rate is set higher allowing automatic progressive refinement.

### 3.3.1 Volume exploration

- **Roaming** – The subvolume serves as a magnifying lens to roam through the full volume. A rectangular subregion of the volume data for 3D inspection is selectable using the keyboard.
- **Probing** – The user can point to a region of the volume to automatically query the original data for the values at that location.
- **Slicing** is done in the axial, coronal and sagittal direction in the three synchronized panels. Selecting a volume position on one slice panel will show the correspondent slice in the other two panels.

### 3.3.2 Clipping

The internal structure of the volumetric data is often obscured in the volume rendering process. Clipping planes can be used to remove regions that occlude and reveal more information. The user can position and orient an arbitrary clipping plane in the scene. The clipping plane is drawn as a polygon texture mapped to the volumetric 3D texture.

### 3.3.3 Interactive isosurface/point extraction

The value from the probe could be used as a threshold to generate isosurfaces or isopoints from within the application at run-time.

### 3.4 Display

Our approach emphasizes rendering to a gamut of displays – from a single monitor to a high-resolution tiled display without the user actually having to change the application. Vol-a-tile supports any arbitrary mapping between the rendering and the display nodes. This is useful for twinhead cards where 1 graphics board renders to multiple viewports or configurations like the IBM berth where multiple machines render to one node. Moreover, side-by-side image rendering for passive stereoscopic systems is also supported. A configuration file is used to specify the mapping and the display options.

#### 3.4.1 AGAVE – passive stereoscopic system

The AGAVE-Geowall, as shown in Figure 13 below is a low-cost passive stereoscopic projection system designed for geoscientific research and education.

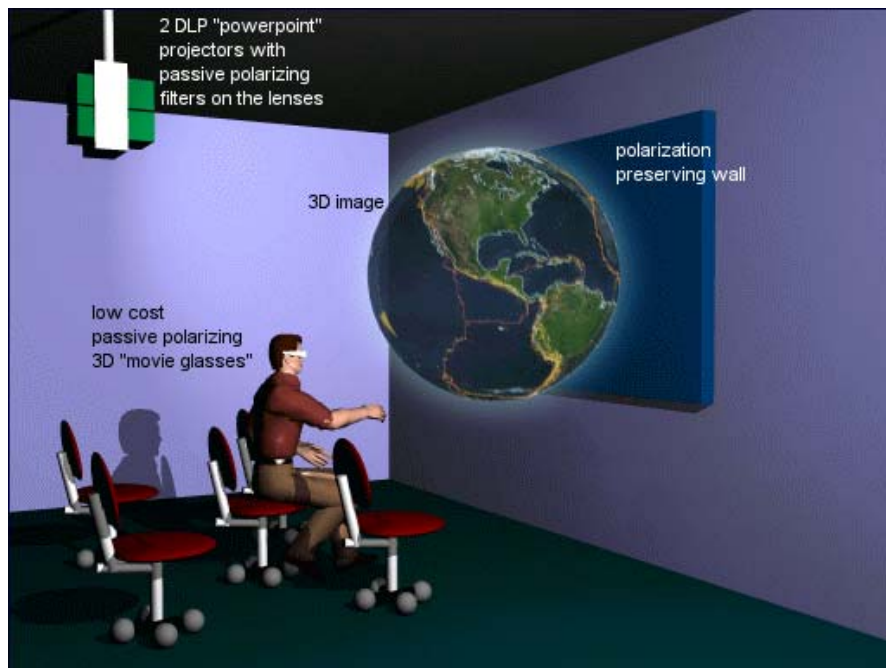
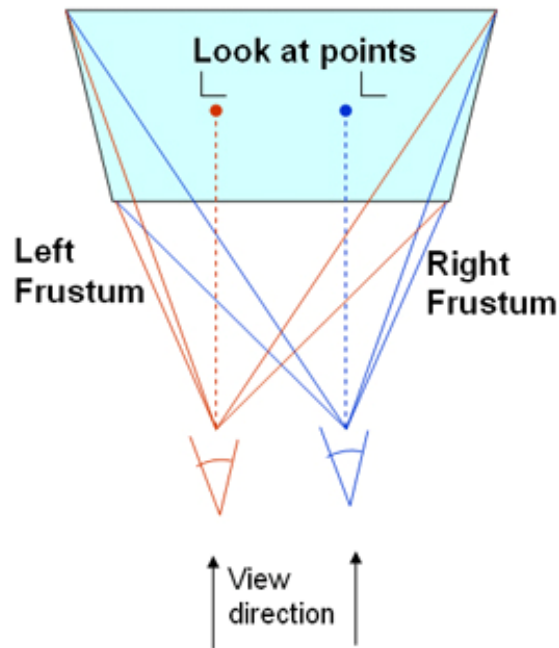
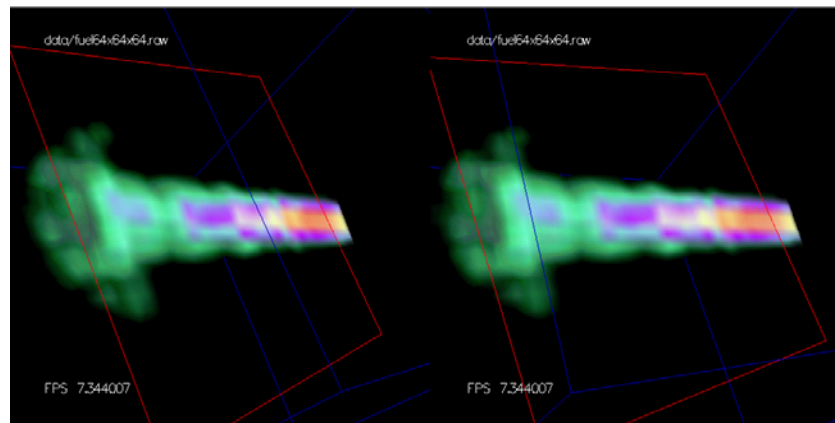


Figure 13 : The AGAVE/Geowall system

The left eye view and the right eye view are generated side by side on the display and delivered to separate projectors that have passive polarizing filters on the lenses. Viewers use polarized glasses to view the images on the screen. To render to a Geowall, Vol-a-Tile generates viewpoints using parallel viewing directions with asymmetrical frustums as shown in Figure 14 below. Figure 15 shows the left-right rendered images on the display.



**Figure 14 : View-frustum calculations for the AGAVE/GEOWALL**



**Figure 15 : Side by Side left eye and right eye view rendering for the Geowall**

### 3.4.2 Perspectile – tiled display

Also dubbed as Geowall2, the PerspecTile is EVL's 8000x4000-pixel LCD tiled display, driven by a cluster of PCs. The view-frustum for each render node is calculated based on the aspect ratio of the whole display and the fraction of the tile extents. One also has take into consideration the image space obscured by the mullions (gaps between the screens). The tile extents are specified in the configuration file. Figure 16 shows the view frustum setup and the equations are given below.

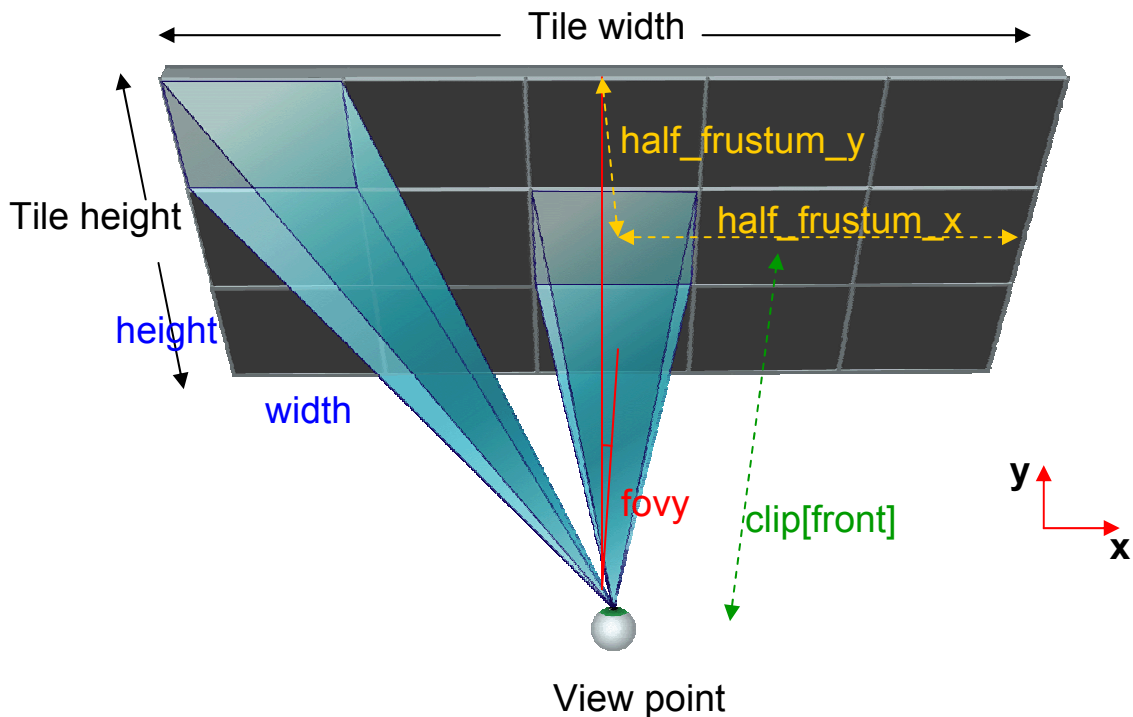


Figure 16: Perspective projection setup for the tiled display

Assuming that the vertical field of view,  $fovy = 45 \text{ deg}$ ,

$$\text{half\_frustum\_y} = \text{clip}[\text{front}] * \tan(\text{fovy}/2)$$

$\text{half\_frustum\_x} = \text{aspectRatio} * \text{half\_frustum\_y}$ , where  $\text{aspectRatio} = \text{TileWidth}/\text{TileHeight}$

Now, we just need to scale this total frustum by the individual tile extents. As an example, for the first tile that starts at (0,0) and has extents (width, height)

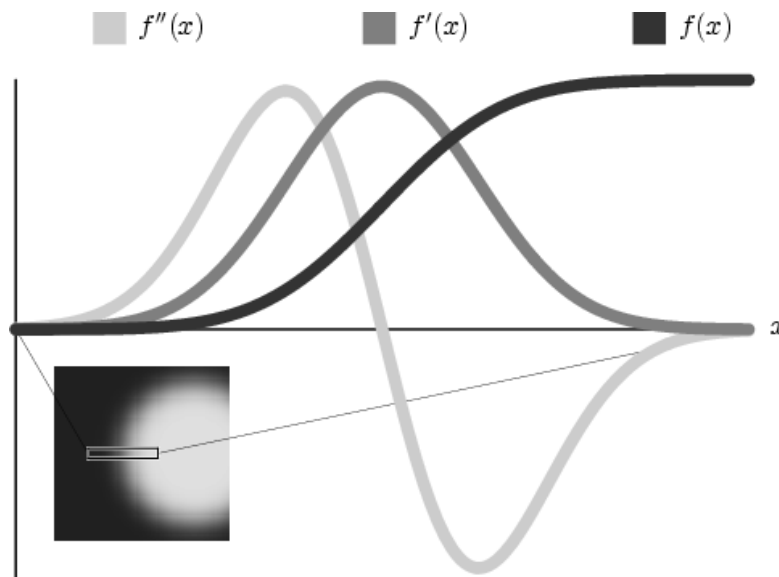
Frustum in X,  $\text{frustum}[x] = (\text{width}/\text{Tile Width}) * \text{half\_frustum\_x}$

Likewise, Frustum in Y,  $\text{frustum}[y] = (\text{height}/\text{Tile Height}) * \text{half\_frustum\_y}$

These frustum values can now be passed to OpenGL or any other graphics library to generate the correct perspective for a tile.

### 3.5 Transfer function specification

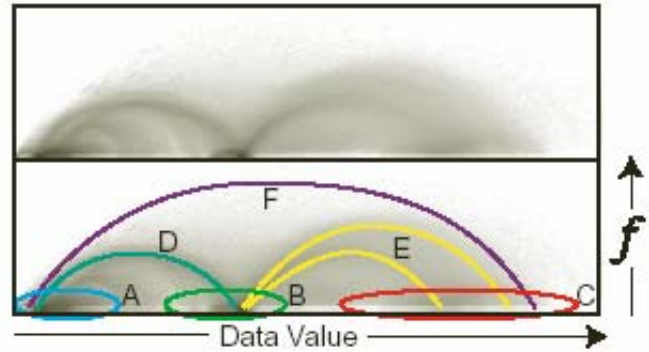
We draw upon the work done by Kniss et al [14] in using multi-dimensional transfer functions. The opacity transfer function will take as input certainly the data value, but also other information such as gradient estimates, and return an opacity value. The gradient value is used when interior structure (such as anatomical features in medical imaging) is to be highlighted: the opacity is scaled down in areas of low gradient, and up where the gradient is high - thus emphasizing boundaries between features. Color transfer functions do a similar job in assigning RGB values to data values. Figure 17 shows the relationship between the data, its first derivative and the boundaries in a dataset.



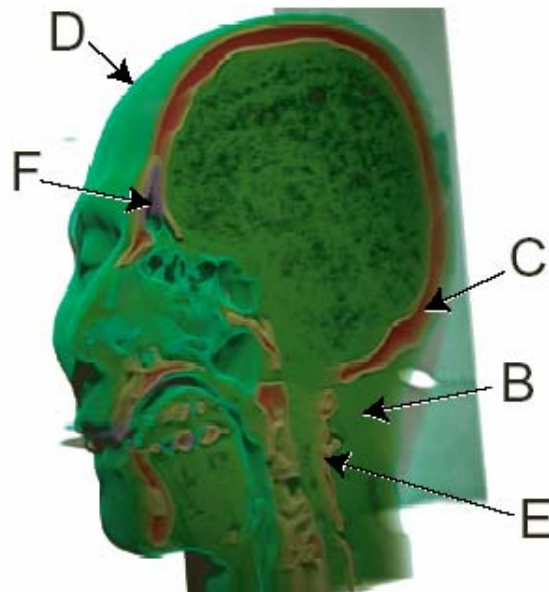
**Figure 17: Relationship between a data value, its first derivative (gradient) and second gradient. Note that at the boundary, the first derivative is at a maximum. This boundary information cannot be inferred from the data values alone.**

One unavoidable drawback to using multi-dimensional transfer function is the increased graphics memory consumption (order of 6!) needed to store all the data derivatives at each voxel

sample point. We tradeoff memory for rendering quality by just presenting the gradient value along with the data value in a 2D histogram to guide the user into selecting a 1D transfer function. Figure 18 and Figure 19 show how the values in 2D histogram are correlated with material boundaries and regions in the dataset using the well-known UNC Chapel Hill CT dataset.



**Figure 18 : A 2D histogram. Materials (A,B, C) and material boundaries (D,E,F) can be distinguished**



**Figure 19 : CT scan of a human head showing the materials and boundaries corresponding to the 2D histogram above.**



### 3.5.1 The User interface

Transfer functions are created by adding one or more classification widgets as shown in Figure 20. The opacity and color contributions from each classification widget sum together to form the transfer function. This rectangular classification widget has a top bar which translates the parameters along the data values. The balls located at the top right resize the widget. The rectangular region spanned by the widget defines the data values and gradient magnitudes which receive opacity and color. Color is constant, but the opacity can be constant or fall off linearly as a ramp.

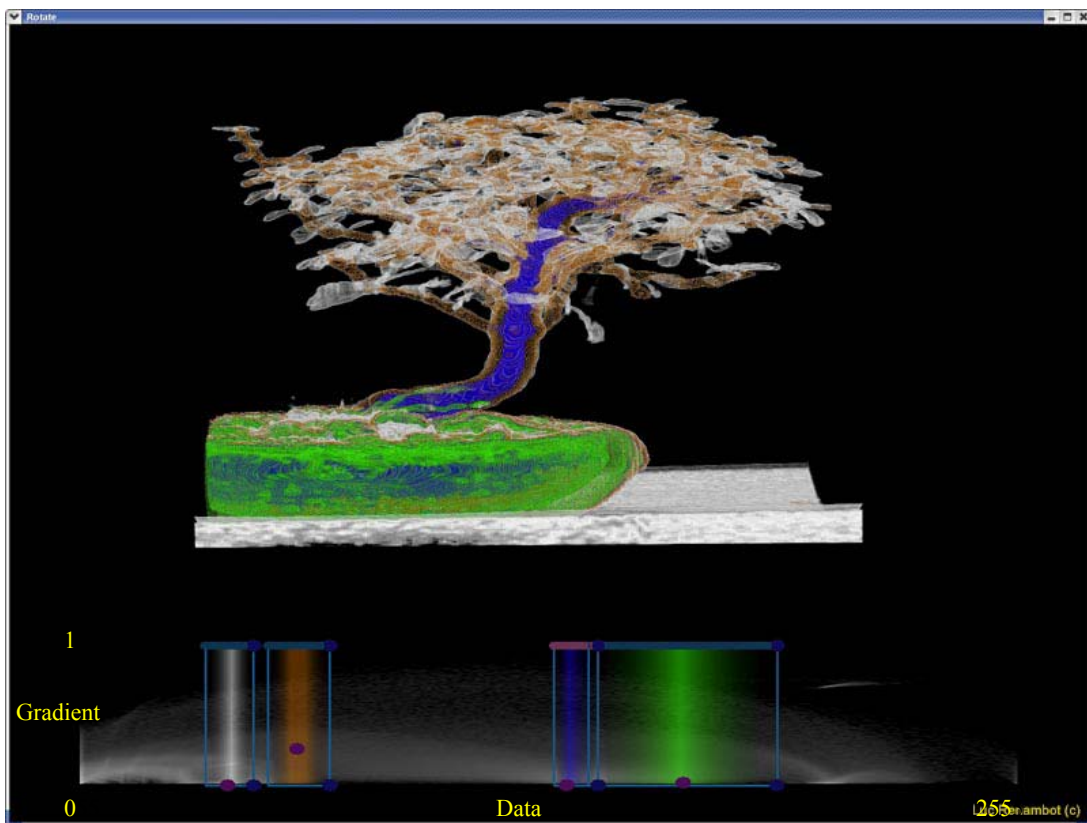


Figure 20 : The transfer function widget overlayed with the volume rendering. This widget was adapted from the Simian project at University of Utah.

The user specifies a color simply by clicking on the rectangle, then moving the mouse horizontally and vertically until the desired hue and lightness are visible. To select a hue, the user moves the mouse horizontally. Vertical mouse motion shifts the color towards white or black. Saturation and opacity are selected independently using different mouse buttons with vertical motion. In most cases, the desired color can be selected with a single mouse click and gesture. The resulting transfer function can be saved and reloaded for future use. In future, we hope to research into non-linear ways of representing color and opacity as many datasets eg seismic are not emphasized by boundaries like tomographic data.

#### 4. SYSTEM DESIGN AND IMPLEMENTATION

This section deals with implementation issues and the interaction between Optistore, Vol-a-Tile and tfUI. All the three components are implemented using C++ on Linux operating system. An implementation of the Message Passing Interface (MPI) - mpich is used for communication within the cluster between the Vol-a-Tile master and slaves. The QUANTA toolkit [25] is used for communication between the 3 components. QUANTA is a cross-platform adaptive networking toolkit developed at EVL for supporting data delivery requirements of interactive and bandwidth intensive applications. The OpenGL Utility toolkit is used to handle the display and user-interaction. The interaction between the components and software libraries used are shown in Figure 21.

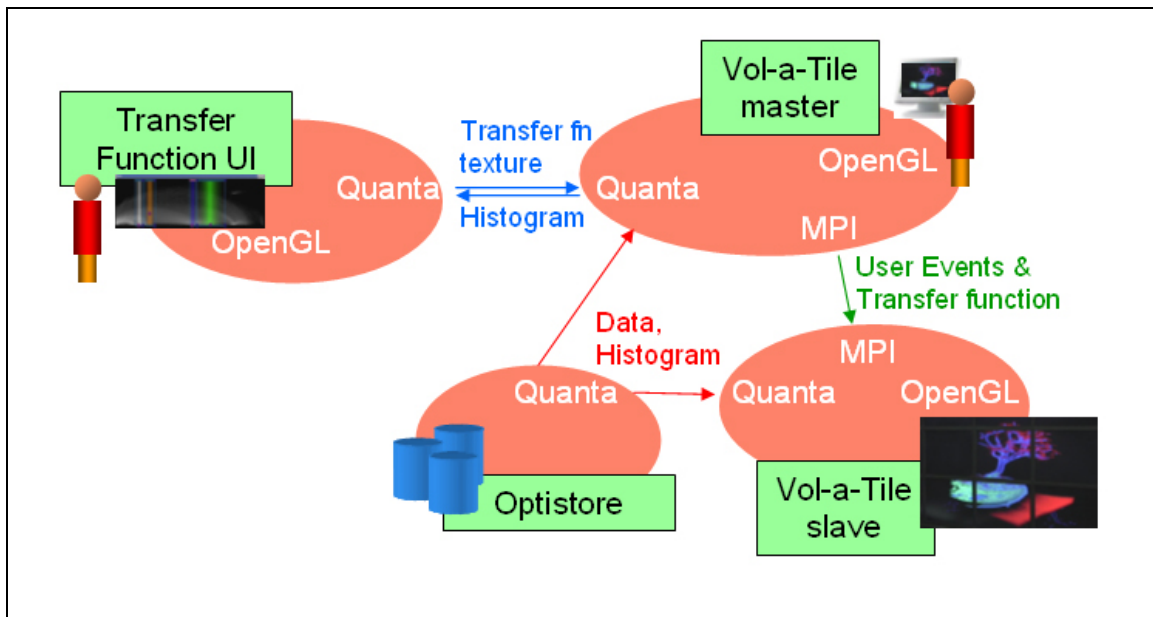


Figure 21: Interaction between Optistore, Vol-a-Tile and tfUI and the libraries used for communication.

#### **4.1 Optistore**

Optistore is executed on each machine that acts as the dataserver and communicates with the Vol-a-Tile renderer using QUANTA TCP. Optistore is capable of handling multiple clients and does this by having a separate thread for each client. Each thread is responsible for handling its clients' requests and then invoking the appropriate VTK objects to do the processing.

#### **4.2 Vol-a-Tile**

Vol-a-tile handles the rendering, scalable displaying and user interaction. All the nodes in Vol-a-tile (master and slaves) run a copy of the application and have a dedicated link to Optistore to retrieve the datasets. The master handles user interaction and any transfer function updates from the transfer function editor and broadcasts it to the slaves using MPI – the standard message passing interface for communication. Whenever the user presses a key or moves the mouse, this interaction is recorded, broadcast, and then evaluated by each rendering node. To compose the global picture, each rendering node renders only a partial image using a reduced frustum. The rendering nodes synchronize before the frame swap to minimize image tearing. Vol-a-Tile uses OpenGL for the actual rendering and the NVidia CG libraries to apply the transfer functions.

#### **4.3 A roam example**

To better understand the interaction between Vol-a-Tile and Optistore, an example of a roam operation is presented. This example walks through all the steps that need to be performed right from when a user presses a key to ask for the new subvolume to when the requested volume is displayed. Chapter 5 will analyze these processes quantitatively. Figure 22 shows all the steps and the interaction between the components for this operation. The description of each process is given below:

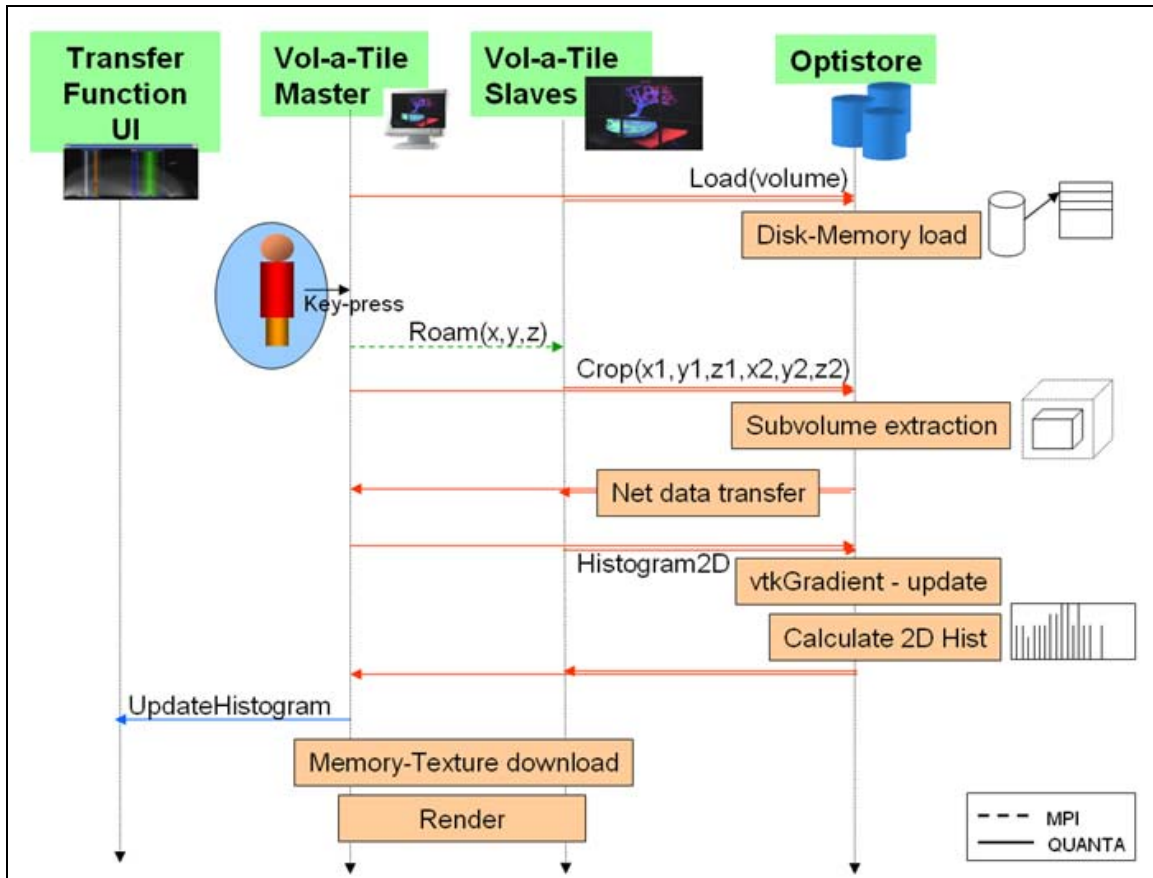


Figure 22 : The steps that occur for a roam operations from when a user presses a key to when the volume is displayed.

#### 4.3.1 Disk-Memory load

This occurs at the beginning of the program execution. The whole dataset corresponding to the symbolic name specified by Vol-a-Tile is loaded into memory from disk. The data can be read from disk as compressed (RLE or gzip) or as raw bytes.

#### 4.3.2 Subvolume extraction

When the user presses a key, the message is broadcast to the rendering slaves. The master and the slaves then request the subvolume for the given extents from their respective Optistore server. The subvolume size is selected so as to fit into texture memory.

### **4.3.3 Network data transfer**

The cropped subvolume is then transferred by each Optistore server to its respective Vol-a-Tile renderer using TCP.

### **4.3.4 Histogram operations**

Optionally, the renderer can request a 2D histogram from the dataserver. This histogram is computed based on the data and gradient values and is eventually used by the transfer function editor as a guide for transfer-function selection. If the tfUI is not used and previously saved transfer functions are loaded instead, this compute-intensive step can be omitted. There are 2 steps in the calculation

#### **4.3.4.1 vtkGradient - update**

We simply maintain 1 copy of the volume data and use the vtkGradient estimator to calculate the gradient values on the fly. This expensive process can be avoided if we precompute and store the gradient volume in addition to the data (which increases main memory and disk storage requirements).

#### **4.3.4.2 Calculate 2D Histogram**

This step calculates a 2D logarithmic histogram using the voxel and the gradient values.

### **4.3.5 Memory-Texture download**

When the renderer receives a new subvolume from the dataserver, it has to be downloaded to the graphics memory to be visualized as 3D textures. At this point, if the tfUI is connected, Vol-a-Tile also updates the tfUI with the new histogram that was just received.

### **4.3.6 Render**

Once the volume is downloaded into texture memory, it is rendered depending on the display configuration.

#### 4.4 Transfer function editor

. tfUI is an optional component of the system and is generally executed remotely, perhaps on a user's laptop. When the tfUI is not invoked, the Vol-a-Tile renderer simply uses a default or a previously saved transfer function. Figure 23 below shows the interaction between Vol-a-Tile and tfUI.

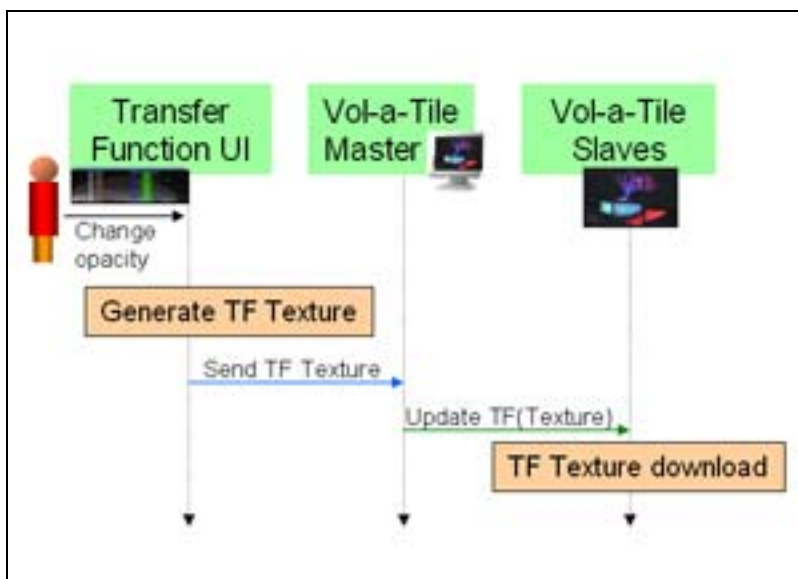
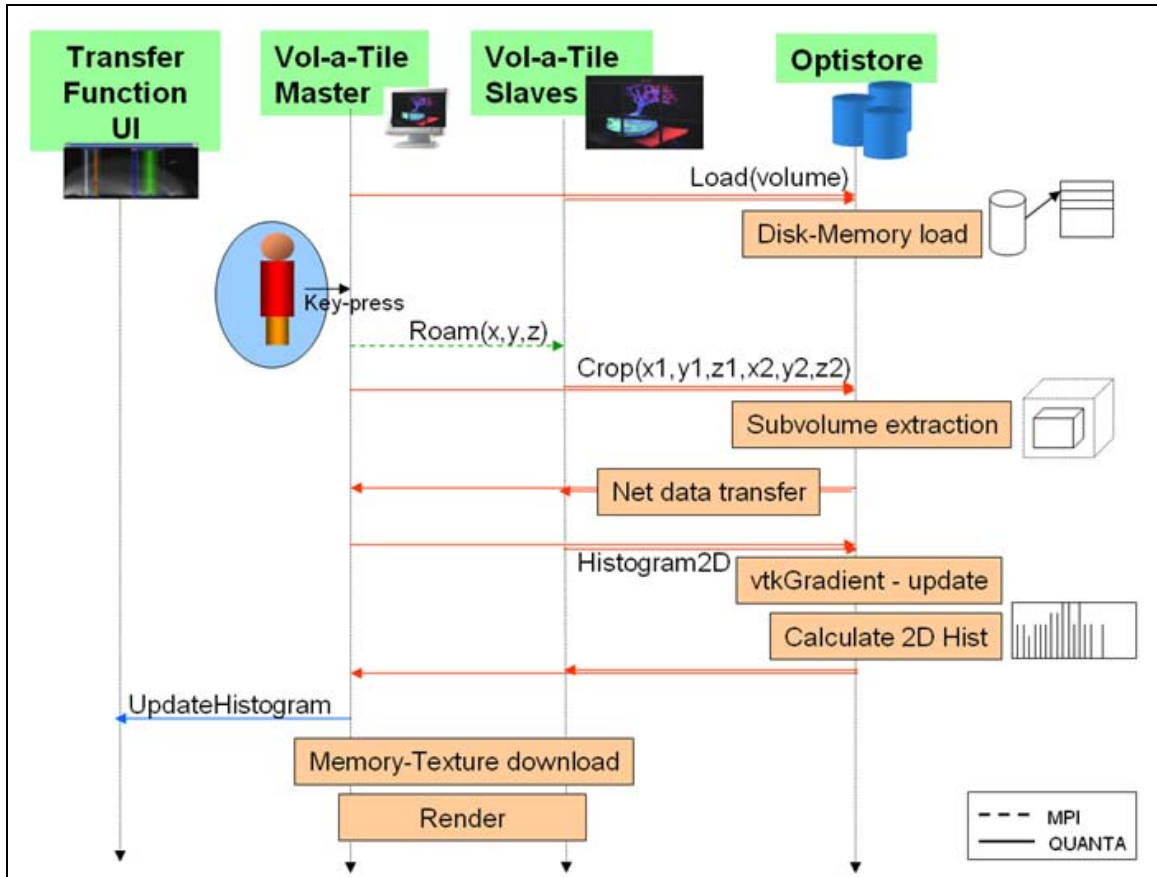


Figure 23 : Interaction between the Transfer function editor and the Vol-a-Tile renderer

As is shown in the roam example in Figure 22, when a new dataset is to be visualized, its histogram is sent to the transfer function editor from Vol-a-Tile. This histogram serves to summarize the data and any interesting features it has (boundaries etc). The results of user-interaction with the various classification widgets can be overlaid and then rasterized to a 2D texture which is then sent to Vol-a-Tile. These texture values are sent using QUANTA TCP. The Vol-a-Tile renderer will then have to download this texture to the graphics hardware prior to rendering a volume.

## 5. RESULTS AND BENCHMARKS

This section presents some quantitative results from a typical volume roaming operation presented in Chapter 4 that is also reproduced below. We will attempt to benchmarks the various steps.



System configuration for the tests-

- Scylla cluster at EVL - 15-node Dual Proc Xeon CPU 1.80GHz
- GPU - NVidia Quadro Fx 3000 ;1600x1200 resolution
- Main Memory - 1.5GB
- Network - gigE interface



## 5.1 Disk-Memory load

Figure 24 below compares loading time for the 3 datasets used with the different compression schemes - Raw binary, Run-length Encoded (RLE) and gzipped. These datasets are described in detail in Chapter 6. It can be seen that the compression schemes fail for data that is unstructured and noisy (eg ARAD).

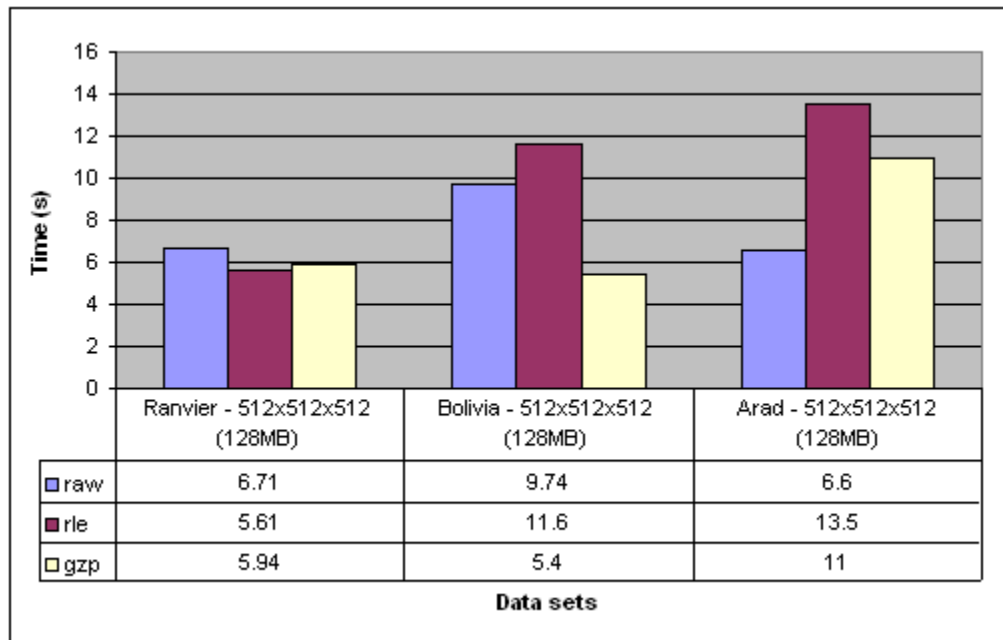


Figure 24: Disk to memory load for the different compression schemes

## 5.2 Subvolume extraction

The time taken to crop the full volume for the different subvolume sizes is shown below in Figure 25.

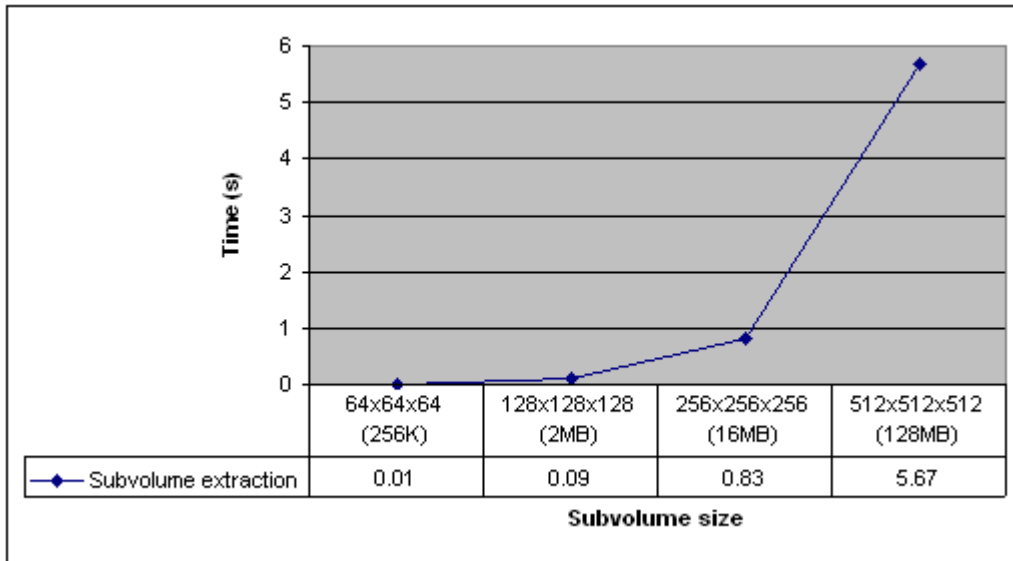


Figure 25 : Time taken for subvolume extraction in Optistore

### 5.3 Network data transfer

The graphs below shows the time for network data transfer and the total roam time. Three possible cases have been tested - local, cluster and remote

#### 5.3.1 Local

In this case, the server and the renderer are collocated ie the same machine runs Optistore and Vol-a-tile. There is no actual network communication involved here.

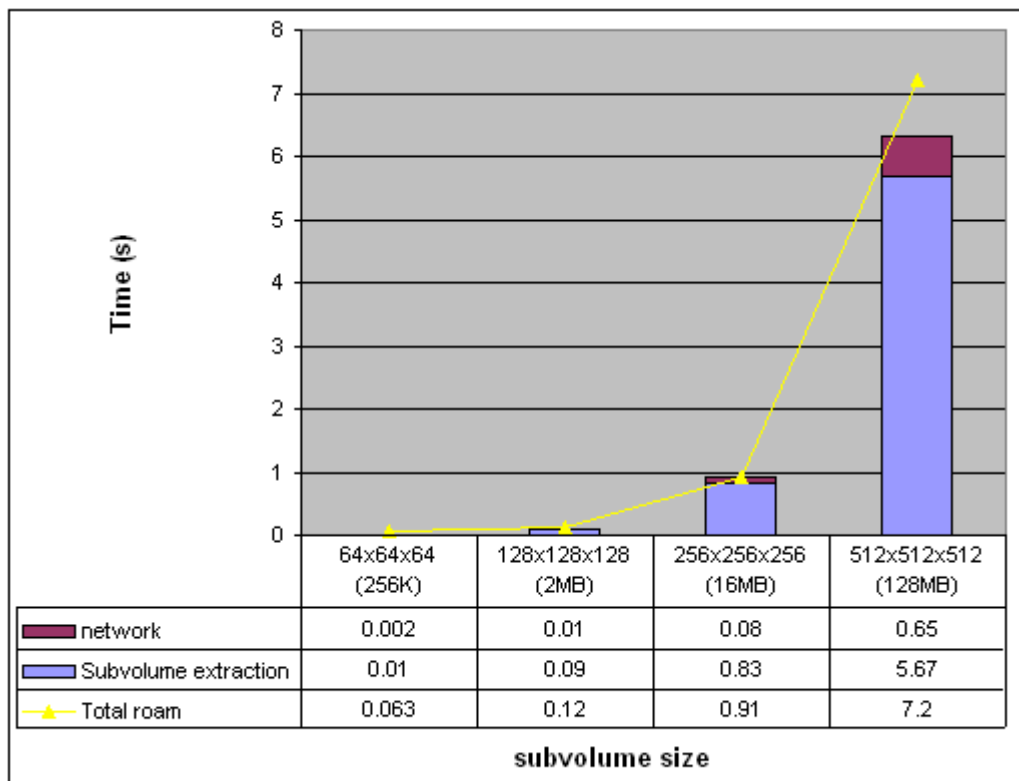


Figure 26 : Time taken for Roaming in the local case

### 5.3.2 Cluster

The server and the renderer are running on different nodes but within the same cluster. So, half the nodes of the cluster run Optistore and the other half Vol-a-tile. Network data transfer is over GigE.

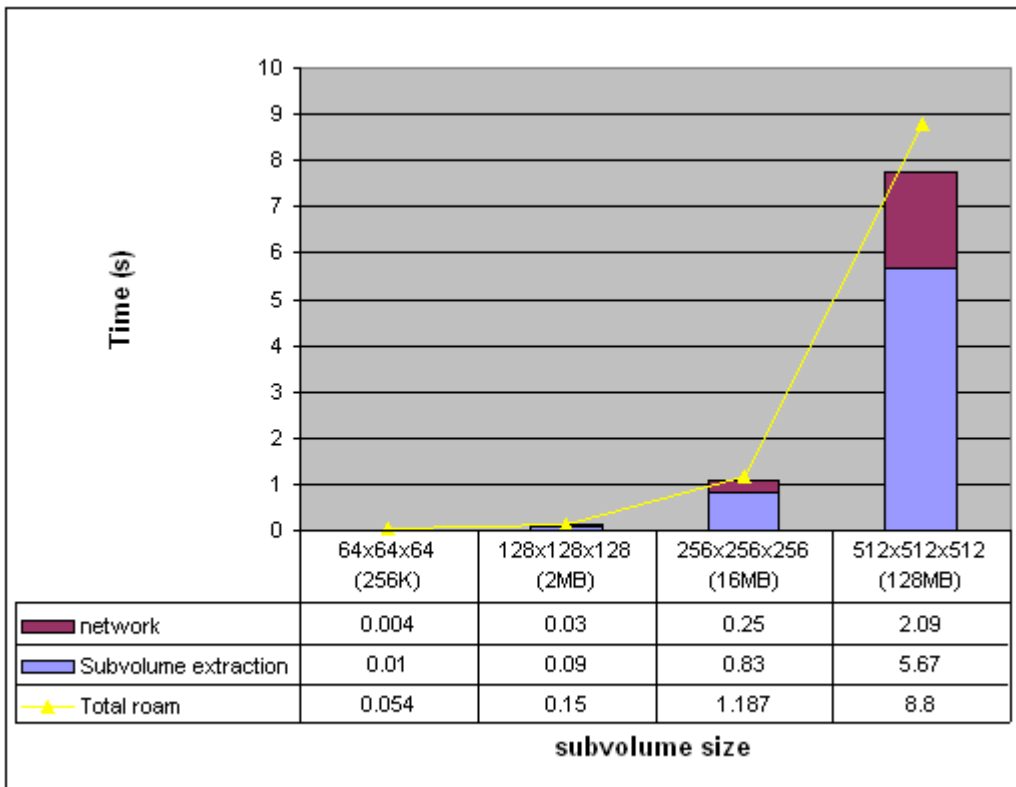


Figure 27 : Time taken for the roam within the cluster

### 5.3.3 Remote

The data server is remote, in this case at IGPP/SIO in San Diego and the rendering on scylla cluster at EVL. Data transfer is over regular internet. As seen in Figure 28, the network transfer times are high as expected.

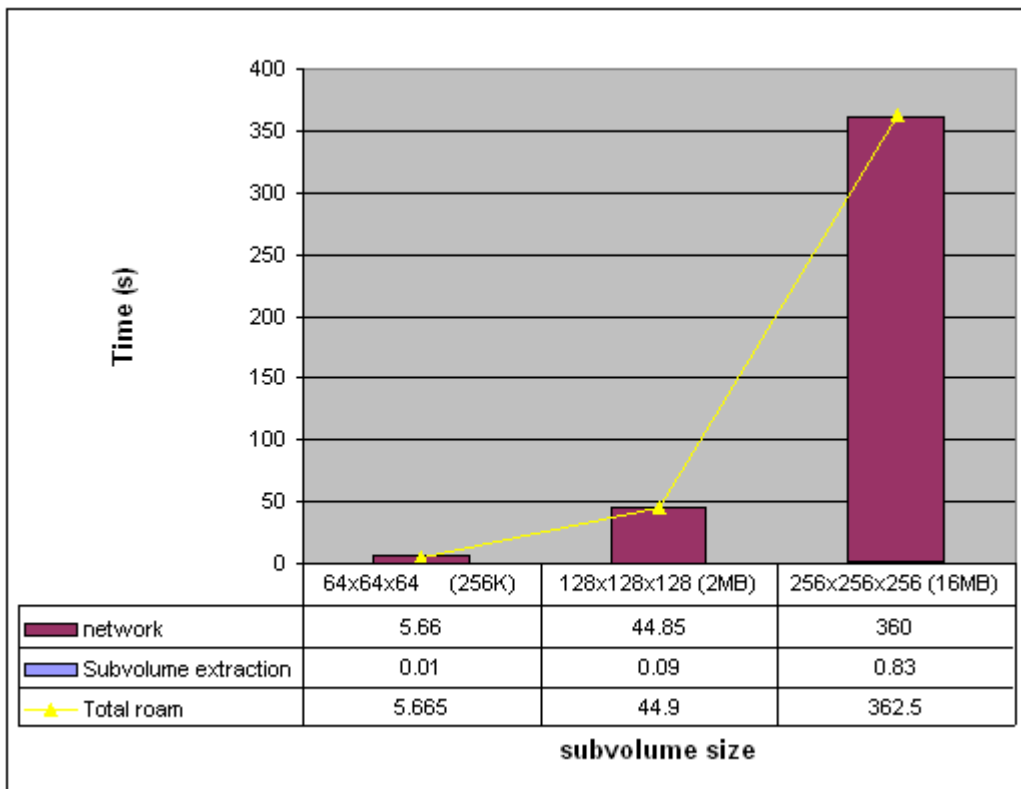


Figure 28 : Roam between Chicago and San Diego

## 5.4 Histogram operations

The graph below in Figure 29 compares the time taken for the vtkGradient-update and the 2D histogram calculation. The bulk of the time is spent simply updating the gradient volumes. This step can be bypassed if a precomputed volume for the gradients is stored.

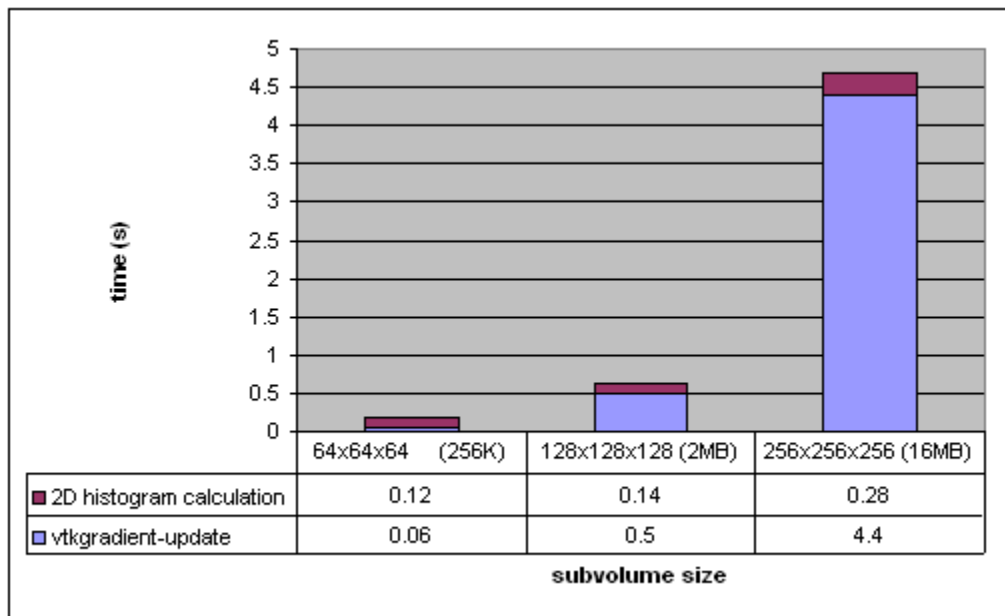
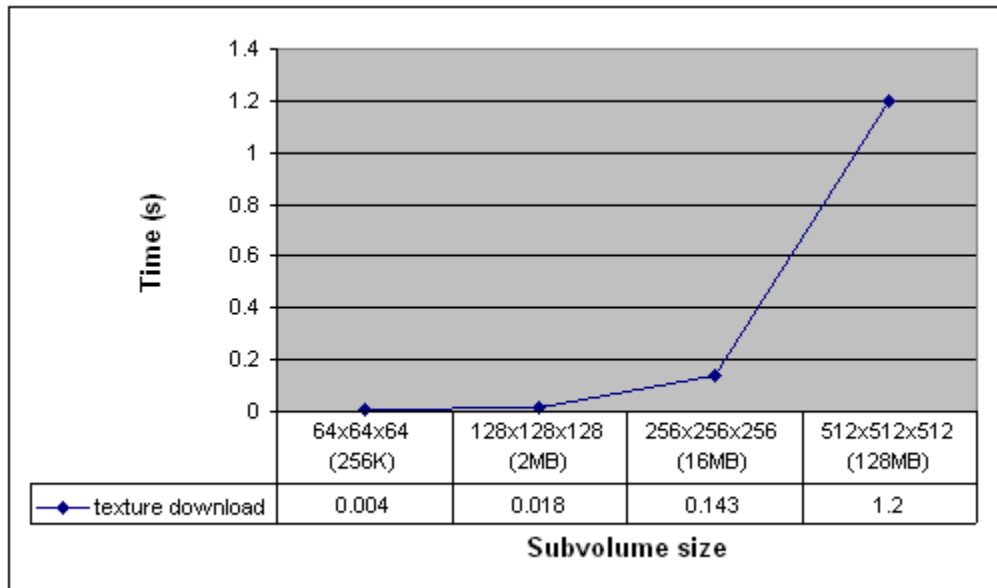


Figure 29 : Time taken for histogram calculations

## 5.5 Memory-Texture download

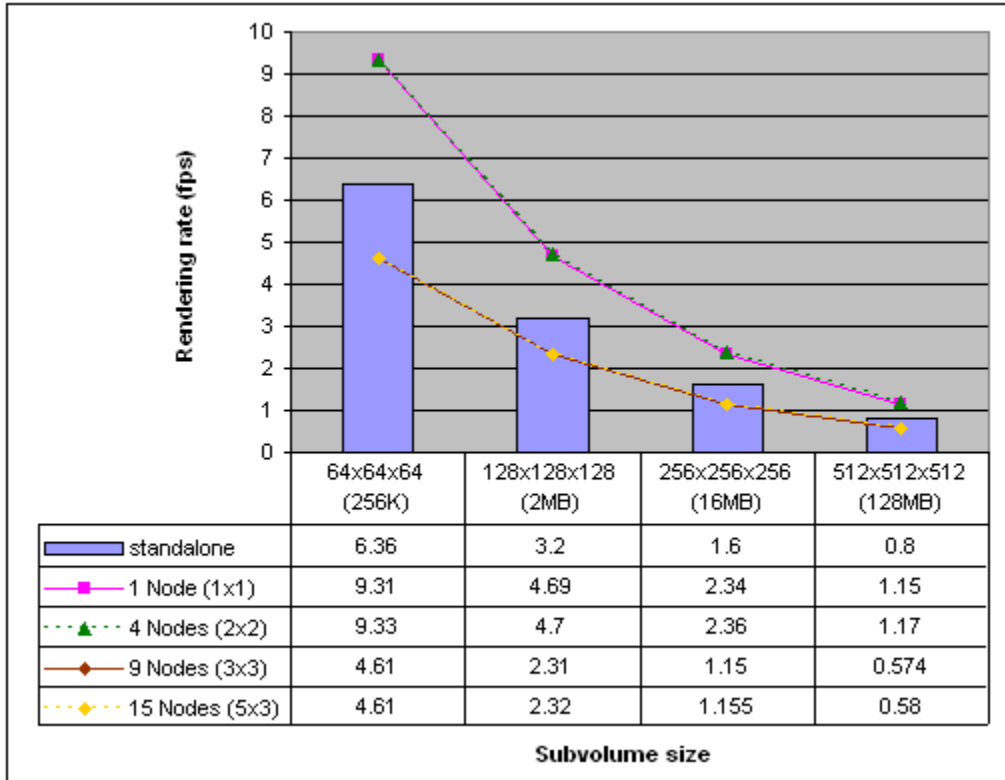
The time taken for the volume to be loaded from memory to graphics is shown below in Figure 30.



**Figure 30 : Time taken for the texture download from memory to graphics**

## 5.6 Render

The graph below in Figure 31 compares frame-rates for rendering with different tile-configurations. The number of texture slices is constant throughout and all nodes render to full screen (1600x1200). The performance of standalone version of Vol-a-Tile is also shown which is lower than the corresponding distributed version. The reason being the glutIdleFunc which does the continuous redraw. In the distributed version, MPI does this for us.



**Figure 31: Graph that compares the rendering rate for different data sizes and different tile configurations or display resolution.**

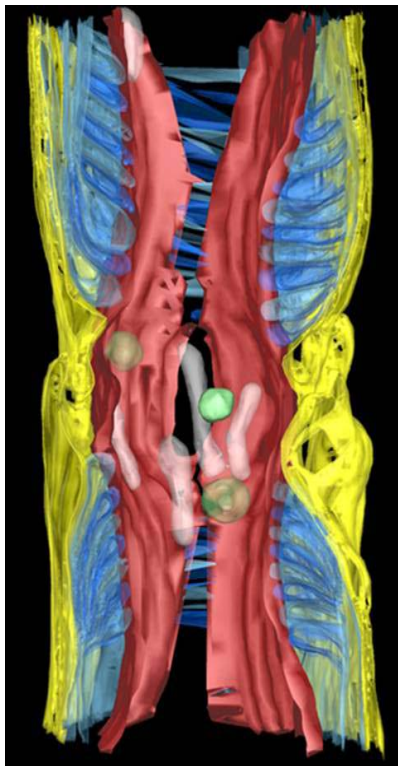


## 6. APPLICATIONS

This section presents the results of work with our research partners in the Optiputer project - The Scripps Institute of Oceanography and the Biomedical Informatics Research Network in the application areas of Neuroscience and Earth Sciences respectively.

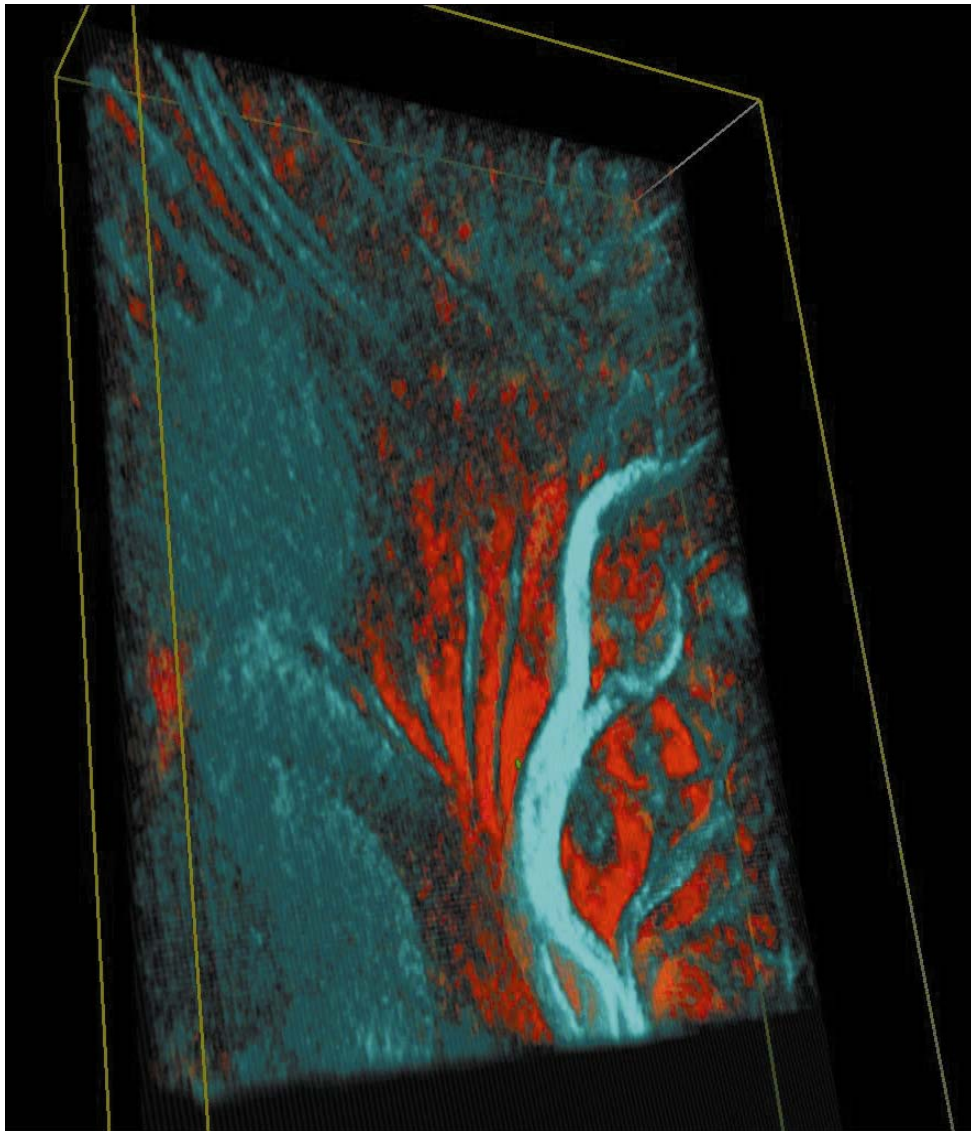
### 6.1 Neuroscience

Biological tomography is one of the driving applications for the Optiputer. The computational challenges in biological tomography are the typically large data set sizes (1-4GB) and the low contrast, single color data. Hence, selecting an effective transfer function is of paramount importance.



**Figure 32: A high-quality reconstruction of the Node of Ranvier that was generated offline**

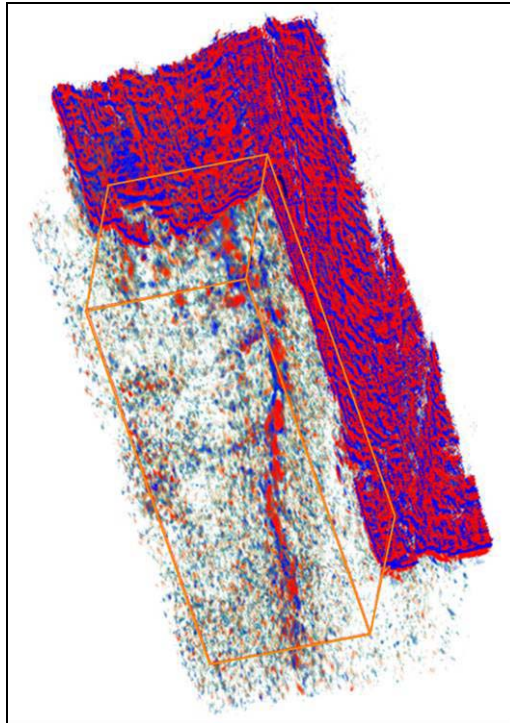
Shown above in Figure 32 is an electron microscopic and tomographic reconstruction of a node of Ranvier about 1.5 micrometers wide. These nodes are small, specialized sites found at intervals along the thin nerve processes of all vertebrate animals that serve to reamplify nerve impulses. Figure 33 below shows the same dataset rendered at interactive frame-rates using Vol-a-Tile.



**Figure 33 : The Node of Ranvier rendered using Vol-a-Tile. The transfer function is set to highlight the axon which is green. The red hues represent myelin**

## 6.2 Seismic Reflectivity experiments

In collaboration with the Scripps Institute of Oceanography (one of the collaborators in the OptIPuter project), Vol-a-Tile has been used to visualize 3-D reflectivity volumes that are now commonplace in both the oil industry. Seismic images from the *ARAD*-Anatomy of Ridge-Axis discontinuity 3-D experiment [26] were used to demonstrate volume visualization on the OptIPuter.



**Figure 34 : ARAD image from the batch volume renderer used at SIO to generate imagery**

This test volume, as shown in Figure 34 above, highlights the fine-scale structure and is the first 3D reflection image of a crustal magma underlying the East Pacific spreading center located at 9°N, off-shore Mexico where the ridge is misaligned. The dataset is 20 km by 20 km in map view, with the migrated data volume comprising of 1000x1000x800 float values, or roughly 3.2

gigabytes. Figure 35 below shows the same dataset visualized on the level-0 OpIPuter - the first prototype of the Optiputer at the Scripps Institution of Oceanography.

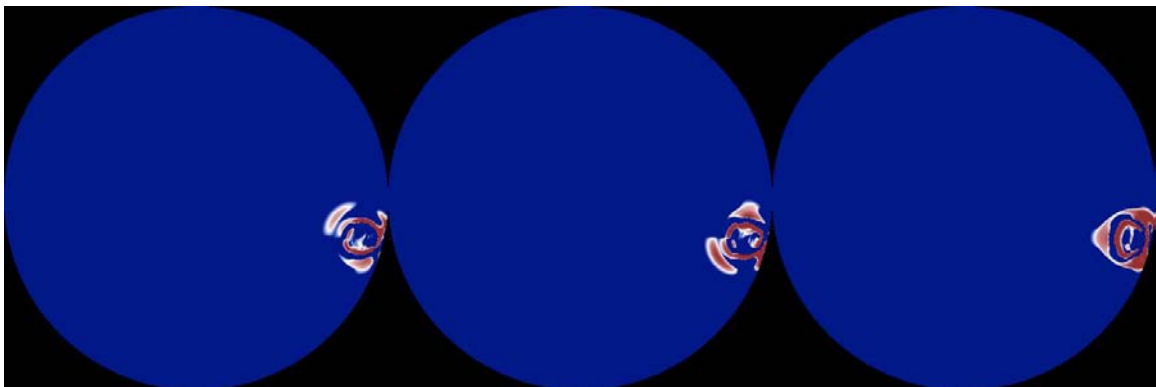


**Figure 35 : SIO's (IGPP, Scripps Institution of Oceanography) high-resolution display system (two IBM 9M pixels monitors), part of the OptIPuter project. Dataset: Arad3D, courtesy of Graham Kent / IGPP, Scripps Institution of Oceanography.**

### 6.3 Seismic wave propagation

Earth Scientists at any level want to gain an accurate understanding of the propagation of seismic waves in the Earth. Wave propagation is generally difficult to understand due to the spherical geometry and strong compositional layering in the Earth. Since there is no way to determine wave motion after an earthquake except where we have seismographs at the Earth's surface, we have to model it computationally.

Here, we are trying to visualize the results of a method devised to solve the full wave equation in a 3D spherical Earth[28]. The known source (think energy) of the earthquake is used to initiate the wave motion. Starting from that initial condition the wave equation is solved in a spherical model for the Earth that is quite realistic in the sense that it represents the known radial velocity variations, compositional variations, anisotropy of wave motion in crust and inner core, etc. The accuracy of the model is validated by comparing the predicted surface motion with the observed seismograms of this particular event. This necessitates a high enough spatial resolution in the model (which requires decent size PC clusters).



**Figure 36 : 2D slice visualization of the seismic wave propagation**

Figure 36 above shows a previous method visualizing the 2D slices of the Earth at various latitudes. The behavior we are interested in is not immediately apparent in the 2D slices. The high spatial and temporal resolution of this simulation presents difficulties when attempting to visualize the wave propagation since the presence of high frequency information requires high spatial resolution in the visualization.

We have developed various approaches to visualizing the realistic wave propagation, using both 2D slices and 3D volumes as well as isosurfaces. Figure 37 shows the simulations for the earthquake in Bolivia (1994). The data size is  $512^3$  with 169 time steps. This time-series visualization shows that waves propagate and reverberate from their origin around the surface of the Earth.



**Figure 37 : Earthquake simulation of Bolivia-1994 showing wave velocity, courtesy of Jeroen Tromp, Dimitri Komatitsch, and Peter van Keken, University of Michigan**

## 7. CONCLUSION AND FUTURE WORK

In this project, we have attempted to incorporate parallelization and distribution at all levels of the visualization pipeline – data access, processing, rendering and display. This is a viable approach in terms of performance since we assume high-speed interconnects between all the distributed processes. Results were also presented for the various possible configurations of the system – local, cluster and remote. We have also tried to incorporate various current researches in hardware-based volume visualization techniques such as dependant textures.

As future work, we would like to implement a seamless and adaptive way of selecting the visualization techniques. Vol-a-Tile gives us interactive frame rates but compromises on image-quality. In future, we hope to incorporate a software batch-renderer which will use Vol-a-Tile to set the user-specified render parameters and render movies offline at very high resolution for playback.

To allow seamless access to large pools of scalable memory, Optistore needs to be integrated with a networked and intelligent shared memory system like LambdaRAM [31] to provide predictive paging of 3D data for visualization. LambdaRAM is a system that uses the OptIPuter concept to create large memory pools to serve as data caches for overcoming access latency over long distance networks.

Multiresolution and hierarchical approaches could be used to store volume data in the limited texture memory of graphics cards. This is possible and effective since volume data generally has lot of “black space”. We also hope to incorporate lighting and shading techniques using the NVidia CGGL since we have access to the gradient volume in addition to the data.

Finally, some consideration has to be given to the specification of non-linear transfer functions to visualize data where boundaries are not emphasized eg seismic data.

## CITED LITERATURE

1. NVidia CG Shader. <http://www.cgshaders.org>
2. The OptIPuter : [www.evl.uic.edu/cavern/optiputer](http://www.evl.uic.edu/cavern/optiputer)
3. J. Leigh, G. Dawe, J. Talandis, E. He, S. Venkataraman, J. Ge, D. Sandin, T.A. DeFanti, AGAVE: access grid augmented virtual environment, in: Proceedings of the AccessGrid Retreat, Argonne, IL, January 2001.
4. K. Moreland, B. Wylie, C. Pavlakos, Sort-last Parallel Rendering for Viewing Extremely Large Data Sets on Tile Displays. Proc. IEEE 2001 Symp. Parallel and Large-Data Visualization and Graphics, San Diego, California, 2001.
5. Leigh, J., Renambot, L., DeFanti, T., Brown, M., He, E., Krishnaprasad, N., Meerasa, J., Nayak, A., Park, K., Singh, R., Venkataraman, S., Zhang, C., Livingston, D. and McLaughlin, M., An Experimental OptiPuter Architecture for Data-Intensive Collaborative Visualization , In Workshop on Advanced Collaboration Environments, June 22-24, 2003, Seattle, WA.
6. T.J. Cullip and U. Neumann. Accelerating Volume Reconstruction with 3D Texture Hardware. Technical Report TR93-027, University of North Carolina, Chapel Hill N.C., 1993.
7. Brian Cabral, Nancy Cam, and Jim Foran. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. 1994 Symposium on Volume Visualization, 91-98, October 1994.
8. Eric LaMar, Bernd Hamann, and Kenneth I. Joy. Multiresolution Techniques for Interactive Hardware Texturing-based Volume Visualization. In IEEE Visualization 99, 355-361, November 1999.
9. M. Weiler, R. Westermann, C. Hansen, K. Zimmerman, Th. Ertl: Level-Of-Detail Volume Rendering via 3D Textures in Proc. IEEE Volume Visualization and Graphics Symposium 2000.
10. Xinyue Li , Han-Wei Shen, Time-critical multiresolution volume rendering using 3D texture mapping hardware, Proceedings of the 2002 IEEE symposium on Volume visualization and graphics, October 28-29, 2002, Boston, Massachusetts
11. John Plate , Michael Tirtasana , Rhadamés Carmona , Bernd Fröhlich, Octreemizer: a hierarchical approach for interactive roaming through very large volumes, Proceedings of the symposium on Data Visualisation 2002, May 27-29, 2002, Barcelona, Spain.
12. J. Ahrens, C. Law, W. Schroeder, K. Martin, and M. Papka. A parallel approach for efficiently visualizing extremely large, timevarying datasets. Technical Report LAUR-00-1630, Los Alamos National Laboratory, 2000.
13. K. Engel , T. Ertl , P. Hastreiter , B. Tomandl , K. Eberhardt, Combining local and remote visualization techniques for interactive volume rendering in medical applications,



Proceedings of the conference on Visualization '00, p.449-452, October 2000, Salt Lake City, Utah, United States.

14. Joe Kniss , Patrick McCormick , Allen McPherson , James Ahrens , Jamie Painter , Alan Keahey , Charles Hansen, Interactive Texture-Based Volume Rendering for Large Data Sets, IEEE Computer Graphics and Applications, v.21 n.4, p.52-61, July 2001.
15. C. Bajaj, S. Park, A. G. Thane Parallel Multi-PC Volume Rendering System. CS & ICES Technical Report, University of Texas at Austin, 2002
16. Pfister, H., Hardenbergh, J., Knittel, J., Lauer, H., and Seiler, L. 1999. The VolumePro Real-Time Ray-Casting System. Proceedings of SIGGRAPH 99 (August), 251–260.
17. Lombeyda, Santiago, Laurent Moll, Mark Shand, David Breen and Alan Heirich, Scalable interactive volume rendering using off-the-shelf components, Proceedings of the 2001 Symposium on Parallel and Large-Data Visualization and Graphics, October 2001, pp. 115-121.
18. Greg Humphreys, Mike Houston, Ren Ng, Sean Ahern, Randall Frank, Peter Kirchner, and James T. Klosowski. Chromium: A Stream Processing Framework for Interactive Graphics on Clusters of Workstations. ACM Transactions on Graphics (Proceedings of SIGGRAPH 2002), pp 693-702, July 2002.
19. T. van der Schaaf, L. Renambot, D. Germans, H. Spoelder, and H. Bal. Retained Mode Parallel Rendering for Scalable Tiled Displays. Immersive Projection Technologies Symposium, 2002.
20. M. Thiebaut, H. Tangmunarunkit, K. Czajkowski, C. Kesselman. Scalable Grid-Based Visualization Framework. Submitted to IEEE symposium on High Performance Distributed Computing, 2004.
21. I. Foster and C. Kesselman, eds., The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999.
22. <http://www.grid.org/home.htm> P. Bhaniramka and Y. Demange. Opendgl volumizer: A toolkit for high quality volume rendering of large data sets. In Proceedings of 2002 Workshop on Volume Visualization, pages 45-54, 2002.
23. S. Guthe, M. Wand, J. Gonser, and W. Strafler. Interactive rendering of large volume data sets. In IEEE Visualization 2002, pages 53-60, 2002.
24. Michael Meissner, Ulrich Hoffmann, and Wolfgang Strasser. Enabling Classification and Shading for 3D Texture Mapping based Volume Rendering using OpenGL and Extensions. In IEEE Visualization 1999, pages 207-214, 1999
25. Eric He, Javid Alimohideen, Josh Eliason, Naveen Krishnaprasad, Jason Leigh, Oliver Yu, Thomas A. DeFanti, "Quanta: A Toolkit for High Performance Data Delivery over Photonic Networks," Journal of Future Generation Computer Systems (FGCS), Elsevier Science Press, Volume 19, Issue 6, August 2003, pp. 919-934.

26. Kent, G., Singh, S., Harding, A., Sinha, M., Orcutt, J., Barton, P., White, R., Bazin, S., Hobbs, R., Tong, C., and Pye, J. Evidence from three-dimensional seismic reflectivity images for enhanced melt supply beneath mid-ocean-ridge discontinuities. *Nature* 406 (Aug. 10, 2000), 614-618.
27. G. M. Kent, J. Orcutt, L. Smarr, J. Leigh, A. Nayak, D. Kilb, L. Renambot, S. Venkataraman, T. DeFanti, Y. Fialko, P. Papadopoulos, G. Hidley, D. Hutches, M. Brown, The OptIPuter: a new approach to volume visualization of large seismic datasets, *Ocean Technology Conference*, May 2004.
28. Van Keken, P., Tromp, J., Komatitsch, D., Venkataraman, S., Schwarz, N., Renambot, L., Leigh, J., .Visualizing seismic wave propagation., *Eos Trans. American Geophysical Union*, 84(46), Fall Meet. Suppl., Abstract ED31E-01, 2003.
29. W. Schroeder, K. Martin, and B. Lorensen. *The Visualization Toolkit: An Object Oriented Approach to 3D Graphics*. Prentice Hall, 2nd edition, 1998.
30. Eric He , Jason Leigh , Oliver Yu , Thomas A. DeFanti, Reliable Blast UDP: Predictable High Performance Bulk Data Transfer, *Proceedings of the IEEE International Conference on Cluster Computing*, p.317, September 23-26, 2002.
31. C. Zhang, J. Leigh, T. A. DeFanti, TeraScope: Distributed Visual Data Mining of Terascale Data Sets over Photonic Networks, .In *Future Generation Computer Systems*, Elsevier Science Press. 2003.