Vision, Modeling, and Visualization (2010)

Tuvok, an Architecture for Large Scale Volume Rendering

T. Fogal[1] and J. Krüger[1,2]

[1]Scientific Computing and Imaging Institute, University of Utah, USA
[2]IVDA, DFKI Saarbrücken, Germany

Abstract

In this paper we present the Tuvok *architecture, a cross-platform open-source volume rendering system that delivers high quality, state of the art renderings at production level code quality. Due to its progressive rendering algorithm,* Tuvok *can interactively visualize arbitrarily large data sets even on low-end 32bit systems, though it can also take full advantage of high-end workstations with large amounts of memory and modern GPUs. To achieve this* Tuvok *uses an optimized out-of-core, bricked, level of detail data representation. From a software development perspective,* Tuvok *is composed of three independent components, a UI subsystem based on Qt, a rendering subsystem based on OpenGL and DirectX, and an IO subsystem. The IO subsystem not only handles the out-of-core data processing and paging but also includes support for many widely used file formats such as DICOM and ITK volumes. For rendering,* Tuvok *implements a wide variety of different rendering methods, ranging from 2D texture stack based approaches for low end hardware, to 3D slice based implementations and GPU based ray casters. All of these modes work with one- or multi-dimensional transfer functions, isosurface, and ClearView rendering modes. We also present* ImageVis3D*, a volume rendering application that uses the* Tuvok *subsystems. While these features may be found individually in other volume rendering packages, to our best knowledge this is the first open source system to deliver all of these capabilities at once.*

Categories and Subject Descriptors (according to ACM CCS): Computing Methodologies [I.3.2]: COMPUTER GRAPHICSGraphics Systems

1. Introduction and Related Work

In the past decade texture-based volume rendering on graphics hardware has positioned itself as a powerful tool for interactive visual analysis of modestly sized data sets. In earlier years slice-based approaches [CN93, CCF94] were utilized due to the limited capabilities of older graphics hardware, with the drawback of distracting visual artifacts. Later, GPU-based ray casting became possible on consumer GPUs, producing superior image quality and allowing for the integration of various acceleration strategies [KW03]. In addition to improvements in volume traversal methods, various approaches have been presented to efficiently render data larger than the video- or even the system's main memory.

As data sizes grow, however, an efficient rendering system only solves part of the problem in visualization. Along a different line of research, novel methods have been proposed to effectively interrogate, search, highlight and present data with an increasing number of high resolution features. In the course of this research multi dimensional- [KKH05], spatialized- [RBS05], size-based- [CM08], motion controlled- [CS05], topology-based [WDC*07], and style transfer functions [BG07], as well as other focus and context enhancing techniques [VGH*05, WZMK05, KSW06] have been developed. For a complete and detailed survey on volume rendering we refer the reader to the state of the art report and courses by Engel et al. [Eng02, EHK*04] as well as the text book by Hadwiger et al. [HKRs*06].

Due to this vast body of research a large variety of different volume rendering systems and prototypes exist both in academia as well as in industry. Yet researchers and developers often find themselves implementing the same basic fundamentals for their new volume rendering application. It may seem that there are many different good reasons for not reusing existing, proven code, but one can usually categorize the decision into one of three cases:

- **System**: Often, the integration of new ideas and methods into large monolithic rendering systems proves to be a bigger issue than re-implementing the entire environment from scratch.
- **Software Environment**: The existing code may be implemented in the wrong environment, such as the programming environment or graphics APIs. For instance, a Di-

DOI: 10.2312/PE/VMV/VMV10/139-146

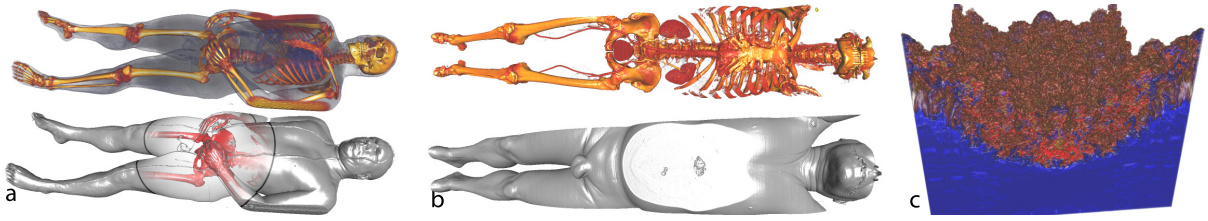


Figure 1: *Large data sets rendered with the* Tuvok *framework. The Visible human CT scan (a), the Wholebody data set (b) and the Richtmyer-Meshkov instability RMI (c).*

rectX implementation will not be suitable for a cross platform project. Further, many research prototypes are tailor made for one system due to the lack of time and need for a more general implementation.

- **Licensing**: while largely irrelevant in the academic environment, license issues often prevent developers in commercial environments from reusing existing code. Even code that is released under Open Source conditions may come with untenable requirements, such as the GPL's stipulation that related yet non-derivative code be released under the GNU license.

Research groups and companies often release their work and thus a number of systems for volume rendering structured data exist as free or open source programs. One of the earliest examples of such an open source volume rendering system is Stanford's VolPack software [Com95]. Unfortunately it has not been under development for more than a decade. A more recent example is the Simian system developed by Kniss et al. [KKH02]. Released under a very liberal open source license, it has a very polished user interface, and allows the application of multi dimensional transfer functions. Unfortunately it falls short as far as data import is concerned and development ceased years ago; therefore no novel render modes are implemented. Other such discontinued frameworks and toolkits are the OGLE [Col02] system, optimized for large data, and OpenQVis [RSEH02], optimized for fast GPU rendering. A program tailored for 3D Microscopy, Voxx [Ind09], has been released by Indiana University; while it has very promising features, including support for 4D data, it is only published in binary form. While Bruckner and Gröller's "volumeshop" [BG05] implements unique GPU accelerated illustrative render options, its development ceased in 2005 and no current version is available. Further, it only supported their proprietary volume format and the current license disallows the use of the code in commercial environments.

For medical applications the MITK [TXD*08] toolkit delivers many interesting features, including support for large data sets and data manipulation routines, but it offers only basic transfer function support and slow performance compared to highly optimized out-of-core GPU volume rendering systems. Solely on the Apple Mac OS X platform, OsiriX [Ros09] offers unmatched DICOM support in an open source application, but as the tool is tied closely to Apple's Cocoa framework and implemented in Apple-extended Objective-C, it is nigh-impossible to port to any other platform.

Instead of using a specialized volume rendering application, existing visualization toolkits can be utilized to render volumetric data. The most prominent examples are the VTK [SML06] and ITK [YAL*02] systems, which allow for extremely versatile and flexible rendering and modification of many types of data sets. The major drawback is the lack of support for out-of-core processing, forcing application developers to concoct external strategies to handle large data sets. The ParaView application [AGL05], built on top of VTK, addresses this issue and extends the support to large data sets but—like the underlying toolkit—does not efficiently utilize the capabilities of current graphics cards, resulting in interactive performance only at very low quality even for modestly sized data sets. Recently, the VisCG at the Universität Münster developed the Voreen system [MSRMH09], a prototyping environment for volume visualization. The interface provided exposes the underlying data flow network and many visualizations require knowledge as to how they are technically realized, which we found was not suitable for a large segment of our user base. Other non commercial visualization toolkits are the OpenDX3 [IBM06] system which is no longer under active development, and finally the SCIRun [Ins09] and VisIt [CBB*05] systems. As these systems suffered some of the problems of previously mentioned solutions (e.g. outdated render modes, slow performance, or limited support for large data sets) *Tuvok* is currently being integrated into these solutions. Besides these free & open source solutions, a number of commercial products exist such as AVS2, Amira, Ensight, syngo, VGStudio Max, or AltaViewer. As these systems are closed source, obtaining detailed information on their operation is difficult; the possibility of integrating *Tuvok* into these systems is intriguing, but we do not discuss them in detail for this work.

In order to address the aforementioned three issues and to overcome the limitations of existing systems, we present *Tuvok*, a system built of cleanly separated components that can be used together, such as in the *ImageVis3D* application, or stand-alone. The entire system is implemented in C++ with OpenGL and DirectX graphics bindings and is designed to be completely platform independent. When necessary, *Tuvok*'s components can be compiled into a shared library and accessed from another programming language. *Tuvok* is also released with a modest open source license that allows unrestricted academic and commercial use of the code. Specifically, *Tuvok* offers the following benefits:

1. **Large Data Support**

Given sufficient storage space, the system can theoretically handle data sets of up to 16 Exabytes in size.
2. **Modular Design**
While the application *ImageVis3D* presents itself to the end-user as a single application, it is composed of a collection of independent *Tuvok* frameworks.
3. **Self contained**
While *ImageVis3D* requires Nokia's Qt [Nok09] library as an external dependency, *Tuvok* itself does not rely on external libraries at all.
4. **Cross platform support**
Tuvok and also *ImageVis3D* support all major platforms, including various versions of Microsoft Windows, Apple Mac OS X, and many Linux variants.
5. **Legacy hardware support**
Tuvok has been extensively tested to work even with the very limited GPU capabilities of older or less capable systems.
6. **Up To Date Rendering algorithms**
Besides its support for 2D and 3D texture based slice based volume rendering—mostly for older graphics hardware—*Tuvok* features GPU based ray casting to interactively render images of the highest quality.
7. **Provenance Support**
Tuvok and *ImageVis3D* provide provenance hooks, with provenance recording and playback realized via VisTrails [BCS*05].
8. **Open Source**
Tuvok and *ImageVis3D* are released under the very liberal MIT license, which means that practically no usage restrictions exist—including the use of *ImageVis3D* or its components in commercial applications.

The remainder of this paper is organized as follows. In section 2 we discuss the design of *Tuvok*, focusing on the ways in which the library handles large data. To demonstrate the versatility of *Tuvok* and *ImageVis3D*, we describe extensions to the system in section 3, and projects that have incorporated *Tuvok* in section 4. We conclude the paper with a summary of the presented system and future research directions.

2. Design

The *ImageVis3D* system is composed of three major components, the *Tuvok* Volume rendering library, the *Tuvok* IO library, and the Qt based UI toolkit. Note that these components are designed to work well together but can also be used separately or replaced by other external libraries (see Section 4 for examples). In fact, during the compilation process of *Tuvok* the subcomponents are compiled as separate libraries that are simply linked together. During the design of these components care has been taken to create flexible and simple interfaces between the subcomponents. As an example of this decoupled design, the communication from the UI to the rendering and IO systems happens through a single entity, named the `MasterController`. This concept makes it easy to intercept all the communication to and from the UI (see Section 3.2) and is also the heart of the scripting interface built into *ImageVis3D*, which allows programmatic control over the application.

2.1. The Volume Rendering Library

The *Tuvok* volume rendering library contains the core graphics algorithms to render volumetric data. Currently, a slice based volume renderer as well as GPU based ray casting renderer are available in OpenGL and DirectX 10. For pure software based rendering the system currently relies on the Mesa library [Pau].

2.1.1. Interactivity and Quality

One of the primary design goals of *Tuvok* is that it should be able to visualize data sets of incredible size on almost any commodity system. At present, we have verified the renderer works with data sizes greater than 2 terabytes [FCS*10]. This is achieved using a streaming, progressive rendering system guaranteeing interactive frame rates with adaptive quality. The generation of full quality imagery is also guaranteed on all configurations, with any data set, but may not happen interactively.

To achieve this goal *Tuvok* utilizes a multiresolution level of detail (LoD) data representation. It queries the volume parameters from the *Tuvok* IO Library—or an external IO framework through a documented API if the standard IO is not used—and uses that information together with the current viewing parameters and system performance history to compute a work order for the current render task. More details are available in section 2.2.

To achieve goals 4-6 in the list above, renderers contain a variety of extra code paths for compatibility settings, as a means to address a number of issues discovered in various OpenGL drivers. *Tuvok* contains multiple renderers, based on ray casting, 3D slicing, and 2D slicing, which span a large range of quality versus portability across GPUs and drivers. This has been important to support a breadth of collaborations, as less technical users tend to have integrated graphics chips which lack support for even 3D textures. Another feature driven by this requirement is the ability to select the bit width of the framebuffer object (FBO) used for rendering, because we found that some drivers would switch to a software path when rendering into a 32-bit FBO.

Table 1 gives timings for multiple data sets on different systems, demonstrating the system's compatibility and scalability. For these timings the progressive rendering has been disabled: only the time to render the maximum quality image for the given view was measured. With the progressive rendering turned on all data sets render at the chosen refresh rates on all systems. Note that the systems used in the test cover chipset integrated GPUs as well as also high end PC configurations. Timings are presented for small data sets as well as reasonably sized CT scans and simulations. Using even larger data sets does not significantly impact the performance of the system, as the amount of data accessed is bounded by the screen resolution.

2.2. Large Scale Data Handling

While *Tuvok* can take advantage of recent advances in hardware capabilities, it is still true that data are growing and have been growing faster than modern hardware. Thus, while

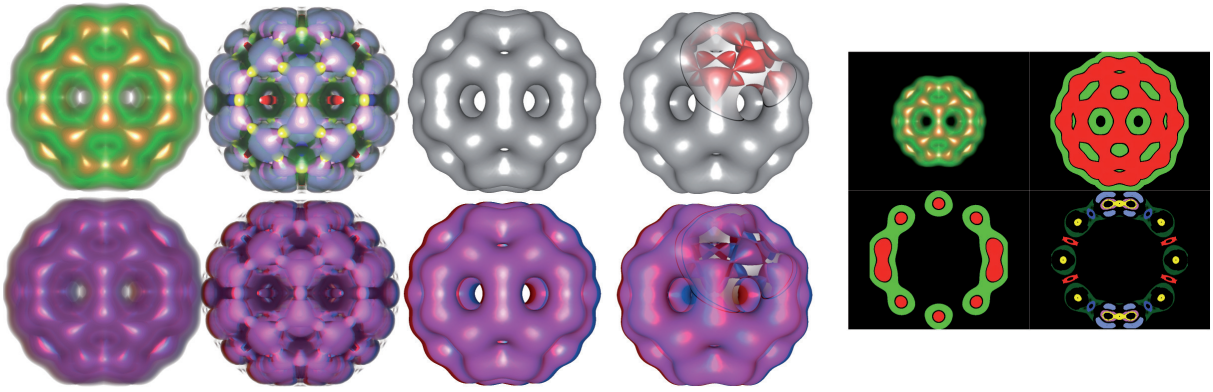


Figure 2: *Various render modes applied to the C60 dataset. In the top row 1D and 2D transfer functions, isosurface extraction, and ClearView are shown. The bottom row shows the same views in anaglyph stereo mode. On the right is two by two mode featuring a 3D view, a MIP view (top right), and two slice views (bottom).*

data set	Air	Pro	Vista
C60 Molecule $128 \cdot 128 \cdot 128 \quad 8bit = \mathbf{2\ MB}$ See Figure 2	110/184	80/124	12/14
VH Male CT $512 \cdot 512 \cdot 1884 \quad 8bit = \mathbf{471\ MB}$ See Figure 1a	380/500	526/744	48/76
Wholebody $512 \cdot 512 \cdot 3172 \quad 16bit = \mathbf{1586\ MB}$ See Figure 1b	680/700	587/984	126/301
RM Instability $2048 \cdot 2048 \cdot 1920 \quad 8bit = \mathbf{7680\ MB}$ See Figure 1c	5523/6112	3112/3520	196/321

Table 1: Tuvok *Timings in **milliseconds** for various data sets and configurations. "Air": MacBook Air, 2GB RAM, Onboard Geforce 9400, "Pro": MacBook Pro, 4GB RAM, Geforce 9600, "Vista": PC running Windows Vista, 24GB RAM, Quadro 5800. All tests were performed in isosurface-mode (first value) and in 1D transfer function mode (second value), using the ray casting renderer sampling twice per voxel, into a* $1024 \cdot 1024$ *viewport. The camera was zoomed such that the data set covered the entire viewport, and the data were divided into bricks of size* 256^3.

the size of data sets that we can interactively render is increasing with each hardware revision, we still find that a larger percentage of our data sets cannot be rendered interactively. It would be unreasonable to assume this trend would reverse in the coming years. Therefore, it is critical that interactive visualization systems incorporate progressive renderers.

Tuvok's progressive renderer is based on overloaded concepts of *frames* and *subframes*. In the context of *Tuvok*, a frame is a single, complete rendering of the data at native screen and data resolution. A subframe is an intermediate state between no rendering and a frame, which includes the full spatial range of the data and any annotations present in the visualization. The quality of successive subframes monotonically increases. A sequence of these subframes are rendered before the final frame is displayed, detailing different approximations of the complete rendering much more quickly than a frame can be displayed. We guarantee that there is always at least one subframe which can be displayed interactively (within a couple hundred milliseconds). The system turns to such a subframe when the user is actively interacting with the data.

To model the concepts of frames and subframes, *Tuvok* uses a multiresolution, level of detail representation of data. For the most part, a subframe corresponds to the data at a particular level of detail. At the coarsest level of detail, the data are small enough that they can easily be read from disk under our real time requirements. However, we found that older GPUs could not always render such data quickly enough for our needs. Therefore *Tuvok* always makes available up to three additional subframes. These are generated by lowering the screen resolution of the rendering (and upscaling before display to the user), lowering the sampling rate used by the renderer, or both. Lowering resolution and sample rate significantly reduces the strain on the fragment processing stage of the graphics pipeline, allowing *Tuvok* to respond quickly even on low end hardware. We do not know any OpenGL 2.0-capable GPU which *Tuvok* does not perform acceptably on, and (through extensions) *Tuvok* can render even on some cards that do not report OpenGL 2.0 capabilities.

2.2.1. Preprocessing

Most data are not fed to visualization software with multiple levels of detail included. To accommodate such data, *Tuvok*'s IO subsystem implements a preprocess which generates a multiresolution hierarchy. The data at their native resolution form the finest level of detail, and we subsample by two recursively until a level of detail exists which is less than or equal to a predefined user-configured limit. We also use this opportunity to perform other operations on the data, for example by converting the endianness so that, in most cases, it will need no transformations when used for rendering later.

The primary issues we face when loading large data are

32-bit address spaces, limitations on GPU 3D texture sizes, and managing the IO in an efficient manner. The address space limits us to only handling two gigabytes of data at any one time. Limitations on texture sizes prevent us from 'simply' loading the data into a single, large 3D texture. Typical IO performance on desktop-class and predicted future hardware informs our strategy for how we access and consume data.

To tackle these issues, the preprocess divides each level of detail into a set of bricks, with each brick small enough to fit into the texture memory of any modern GPU. The rendering core will render each level of detail in an out-of-core fashion: a brick will be loaded, rendered, and discarded in an atomically. This allows the renderer to load data of virtually unlimited size with very little available memory, as the required amount of memory is independent of the data set size. To achieve the IO performance we require, the IO library uses large reads (by default, 16 megabytes) that make seek times virtually irrelevant.

A simple survey of modern disk drives finds reported seek times ranging from 3.75 up to 8.9 milliseconds. Sustained transfer rate capabilities can be as low as 65 MB/s; see table 2. While there are of course differences across drives and manufacturers, multi-megabyte reads very quickly overtake seek times as the predominant factor in disk transfers. At 65 MB/s, it takes just under a quarter of a second to read 16 megabytes of data, yet only 8 milliseconds to seek to the position of that block. Even as one gets into the higher end drives, the story is the same; a Cheetah 15K.5 would take 0.12 seconds to read a 16 megabyte chunk of data, and only 3.75 milliseconds to seek to the appropriate location on disk. In relative terms, seek time makes up approximately 3% of the time required to read the data block. Based on these simple calculations, it is clear that transfer rates will have to improve drastically before seek times become a relevant parameter.

Drive Name	Seek time (ms)	Sustained Transfer (MB/s)
Cheetah 15K.5 SAS	3.75	73 to 125
WD Caviar RE2-GP	8.9	84
Barracuda 7200.8	8	65
WD 740GD	5.2	72

Table 2: *Relevant disk performance characteristics for disks ranging from high-end server drives (Cheetah 15K.5) to an aging model released 6 years ago (WD 740GD)*

We have also benchmarked our I/O subsystem using solid state drives. Table 3 shows the time spent on I/O when loading a 648-brick data set via *Tuvok*. The SSD boasts vastly better seek times, on the order of microseconds instead of the normal milliseconds for mechanical drives, and a factor of two to three improvement in bandwidth. Using large reads, the seek time matters little in this case, but as shown in Table 3 *Tuvok* benefits from the improved transfer times offered by SSDs.

3-Disk SATA RAID5	Solid State Drive
64.8704	27.6723

Table 3: *I/O times (seconds) to render a 9 gigabyte timestep from a simulation of a Richtmyer-Meshkov instability.*

2.2.2. Paging Strategy

Transfer time forms the majority of our pipeline execution time when using high end GPUs. Therefore, by maintaining a cache for individual bricks, we can improve the overall rendering time by obviating the need to pay the transfer cost for every brick rendered.

A straightforward paging strategy for such a cache would be Least Recently Used (LRU), however this strategy delivers poor performance in many situations. Consider a dataset with 10 bricks, and a brick cache capable of storing 9 bricks. In the first frame, all ten bricks must be paged. Further, loading the final brick of the first frame will evict the first brick of that frame. Assuming any reasonable amount of frame-to-frame coherence, the next frame will again need the same 10 bricks, and they are likely to require a similar depth ordering. Thus, in the second frame, the first brick we will need is the brick we just evicted at the end of the last frame; further, the second brick we need will be evicted while loading the first brick of the second frame, and so on throughout the entire frame.

We have implemented a custom paging strategy which takes into account our progressive rendering system. In this strategy, we evict bricks *within* a frame using the Most Recently Used (MRU) strategy; we evict bricks *between* frames using a LRU strategy. The rationale for the former is that once we have used a brick in a subframe, it will not be used in the rendering of that frame again until the progressive renderer starts over, and we may service a large number of bricks in the interim. However, if we do start the frame from its earliest subframe again, particularly before finishing the frame, we are likely to need the oldest bricks which are present in the cache. Between frames, we rely on frame-to-frame coherence. If a brick was not used in the previous frame, and is not used in the current frame, it is likely to not be required in subsequent frames as well; a common example is if the user has enabled a clip plane: any viewing transform will not effect which bricks are clipped away by the plane. Therefore the LRU strategy will tend to evict bricks which are not visible under the current transfer function, isosurface, or viewing parameters.

2.3. UI and Networking Library

To facilitate rapid development of other visualization applications, all those components built on top of Qt which are not specific to the application level were separated, allowing them to be shared and reused in future applications. These components can be roughly categorized as the UI and networking components. The independent networking components include the bug reporting, update checking, and data set sharing subsystems, while the UI components include the base classes that define the look and feel of *ImageVis3D*,

such as dialogs, tool widgets, user interaction, and persistence.

3. Extensions to *Tuvok* and *ImageVis3D*

In this section we will present a couple of examples to demonstrate how simple it is to add new features or extend existing functionality. We present examples from research projects implementing a prototypic environment to experiment with new methods (see Section 3.1) as well as new features to *ImageVis3D* to use it for other research.

Due to the modular design, the scripting interface, and the MasterController concept, integration with external software is simple. As the UI and 'execution layer' communicate strictly through a single class, the `MasterController`, any type of external communication channel can simply attach itself to this class and track changes. Control of the library can also happen through the `MasterController` via script commands that allow programmatic modification of all of *Tuvok*'s features.

3.1. Extensions to the Rendering system

ImageVis3D has been extended to provide domain specific visualization capabilities. In some domains, it is necessary to visualize multiple data sets simultaneously. A student has modified *ImageVis3D* to render multiple data sets that live in overlapping space, and added domain-specific widgets for ease of use in a particular scientific domain. One such example is a dialog to automatically create transfer functions, based on external knowledge of characteristic data distributions within data sets common to that field. A second example is repurposing the 2D transfer function editor to utilize different metadata along each axis.

3.2. Extensions to *Tuvok*'s controller

For provenance tracking, we have integrated VisTrails, a production provenance framework with well-developed APIs for integration with external systems. The integration of VisTrail's provenance tracking features required a two way communication from and to *Tuvok*. Interactions made by the user need to be communicated to VisTrails to track the provenance, but also VisTrails needs to be able to control *Tuvok* to perform undo/redo operations. Thus, this example is prototypic for any type of recording or remote control of *Tuvok*, such as cluster extensions or connections to novel input devices.

4. Use Cases of *Tuvok*

In the following we present examples where *Tuvok*—or only some of it components—have been integrated into rendering environments other than *ImageVis3D*. Figures 4 and 3 demonstrate the integrations presented here.

4.1. SCIRun

SCIRun is a problem solving environment for modeling, simulation, and visualization of scientific data. It is an example of what we refer to as a *legacy application*, in that it was developed without the ideas implemented by *Tuvok* in mind. In particular, this means that the system must work with in-core, "unbricked" data sets of a single resolution.

To support such an environment, *Tuvok* has a simplified API for existing systems which do not include level of detail or bricking concepts. The information flows one way from the controlling application to *Tuvok*, and includes a reference counted smart pointer to the data, as well as metadata and rendering parameters. For small changes in rendering parameters, data shared from previous frames is retained and simply re-rendered. When changing or passing a new data set to *Tuvok*, the old data set is removed and replaced by a new reference counted smart pointer. This scheme allows us to avoid data copying between the host application and rendering library. In these kinds of systems, *Tuvok* does not have access to a multiresolution form of the data, and thus cannot guarantee interactive performance.

4.2. VisIt

VisIt is a data visualization and analysis application which is well-suited to large scale data processing on leadership computing platforms. We have integrated the underlying rendering core as an option alongside VisIt's existing volume renderers. Since VisIt already supported domain-based data set decomposition, it can easily take advantage of an additional *Tuvok* feature: bricking. This allows VisIt to volume render data of arbitrary size on the GPU, whereas it was previously limited to resampling the data or utilizing software rendering.

Though data do not come directly from a data file in this and other integration work, the abstraction provided by *Tuvok*'s IO layer allows the rendering core to remain ignorant about the source of the data. The metadata which must be supplied to *Tuvok* scales with the complexity of the application: in the unbricked, SCIRun case, *Tuvok* can be told only the brick size (assuming the brick lies centered on the origin); with decomposed data, *Tuvok* must be informed of the world space location of the bricks; for progressive rendering applications, such as *ImageVis3D*, the LoD that a brick belongs to must also be given. Should an application choose, it can also supply additional metadata to allow advanced rendering optimizations.

An issue that arose specifically in the VisIt integration was state management in large, established software systems. The OpenGL API is a global state machine, and VisIt has many sub-libraries which can and will change the global state in ways we cannot predict. *Tuvok* therefore makes very few assumptions about OpenGL state. During the 'setup' stage, *Tuvok* takes state information – camera and viewing reference points, data, etc. – and simply stores it locally. A single method then uses all that information to configure OpenGL state once before moving on to per-brick rendering. For efficiency reasons, the system leaves the OpenGL state "as-is" when finished rendering, much like other VisIt subsystems and libraries do. Until OpenGL establishes an object model, we have found this to be the best method for managing OpenGL state.

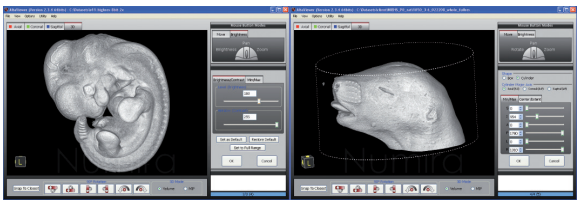


Figure 3: *The left image shows an E11 mouse embryo (2.4GB) while the right image depicts a P0 newborn rat (7.6GB). Both specimens were stained using Numira's custom protocol and scanned using microCT. Images courtesy of Numira Biosciences. Copyright ©2009 Numira Biosciences. All rights reserved. AltaViewer Software available at* `http://www.numirabio.com/`

4.3. AltaViewer

Finally, we demonstrate the usability of *Tuvok*'s components in a commercial environment. Numira Biosciences is a specialty contract research organization (CRO) which focuses on high-resolution imaging and analysis of small animal specimens, provides researchers with quantifiable, visible evidence of disease progression, as well as drug efficacy and drug side effects in their animal models. For the next generation of their visualization suite "AltaViewer" (see Figure 3) they have chosen to replace their proprietary IO library in part by *Tuvok*'s IO components to achieve significantly better performance.

5. Conclusion and Future Work

In this paper we have presented the *Tuvok* framework as well as *ImageVis3D*, an application built with *Tuvok* . We gave insight into large data support in a production volume renderer. We also gave a couple of examples of research projects and commercial use of components of *Tuvok*.

We are currently working on three major extensions to *Tuvok*. First, the support of time dependent data sets, in particular we are working to extend the progressive rendering concept to this data as well. Secondly, we are extending *Tuvok* to render multiple data sets in overlapping 3D space; due to the out-of-core nature of the system an efficient implementation of this feature is non-trivial. Finally, we also plan to add purely software based as well as OpenCL based ray casters to allow for fast rendering of ultra large data sets on headless clusters with and without GPUs.

Acknowledgements

This research was made possible in part by the Cluster of Excellence "Multimodal Computing and Interaction" at the Saarland University, by the NIH/NCRR Center for Integrative Biomedical Computing, P41-RR12553-10 and by Award Number R01EB007688 from the National Institute Of Biomedical Imaging And Bioengineering, and by the Office of Advanced Scientific Computing Research, Office of Science, of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231 through the Scientific Discovery through Advanced Computing (SciDAC) program's Visualization and Analytics Center for Enabling Technologies (VACET). The work presented in this paper has been cofinanced by the Intel Visual Computing Institute. The content is under sole responsibility of the authors. We thank John Blondin for some of the data pictured in Figure 4. Furthermore we thank Numira Bioscience for the AltaViewer screen shots, the visible human project for the visible human scan and Siemens Corporate Research for the Wholebody data set.

References

[AGL05] AHRENS J., GEVECI B., LAW C.: *The Visualization Handbook*. ., 2005, ch. ParaView: An End-User Tool for Large Data Visualization.

[BCS*05] BAVOIL L., CALLAHAN S. P., SCHEIDEGGER C. E., VO H. T., CROSSNO P. J., SILVA C. T., FREIRE J.: Vistrails: Enabling interactive multiple-view visualizations. *Visualization Conference, IEEE 0* (2005), 18. http://doi.ieeecomputersociety.org/10.1109/VIS.2005.113.

[BG05] BRUCKNER S., GRÖLLER M. E.: Volumeshop: An interactive system for direct volume illustration. In *Proceedings of IEEE Visualization* (2005), C. T. Silva E. Gröller H. R., (Ed.), pp. 671–678. http://dx.doi.org/10.1109/VIS.2005.135.

[BG07] BRUCKNER S., GRÖLLER M. E.: Style transfer functions for illustrative volume rendering. *Computer Graphics Forum 26*, 3 (Sept. 2007), 715–724. was awarded the 3rd Best Paper Award at Eurographics 2007.

[CBB*05] CHILDS H., BRUGGER E., BONNELL K., MEREDITH J., MILLER M., WHITLOCK B., MAX N.: A contract based system for large data visualization. *Visualization Conference, IEEE 0* (2005), 25.

[CCF94] CABRAL B., CAM N., FORAN J.: Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In *Proceedings ACM Symposium on Volume Vis* (1994), pp. 91–98.

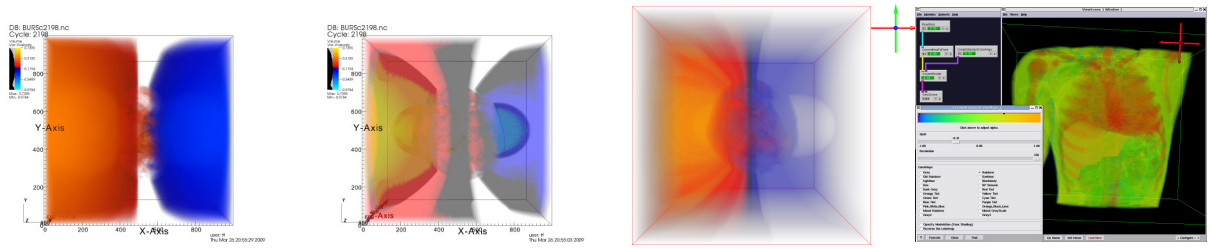
[CM08] CORREA C., MA K.-L.: Size-based transfer functions: A new volume exploration technique. *IEEE Transactions on Visualization and Computer Graphics 14*, 6 (2008), 1380–1387. http://doi.ieeecomputersociety.org/10.1109/TVCG.2008.162.

[CN93] CULLIP T., NEUMANN U.: *Accelerating Volume Reconstruction with 3D Texture Hardware*. Tech. Rep. TR93-027, University of North Carolina, Chapel Hill N.C., 1993.

[Col02] COLORADO RESEARCH ASSOCIATES DIVISION OF NORTHWEST RESEARCH ASSOCIATES INC.: OGLE Large-Scale Scientific Data Visualizer, 2002. http://www.nephrology.iupui.edu/imaging/voxx/.

[Com95] COMPUTER GRAPHICS AT STANFORD UNIVERSITY: The VolPack Volume Rendering Library, 1995. http://graphics.stanford.edu/software/volpack/.

[CS05] CORREA C. D., SILVER D.: Dataset traversal with motion-controlled transfer functions. In *Proceedings of IEEE Visualization* (Oct. 2005), pp. 359 – 366.



(a) (b) (c) (d)

Figure 4: *3D Texture (a), SLIVR (b), and* Tuvok *(c) volume renderers in VisIt;* Tuvok *rendering a torso in SCIRun (d).*

[EHK*04] ENGEL K., HADWIGER M., KNISS J. M., LEFOHN A. E., SALAMA C. R., WEISKOPF D.: Real-time volume graphics. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Course Notes* (2004), ACM, p. 29.

[Eng02] ENGEL K.: STAR: Interactive High-Quality Volume Rendering with Flexible Consumer Graphics Hardware. In *State of the Art Reports* (2002), Fellner D. W., Scopigno R., (Eds.), Eurographics.

[FCS*10] FOGAL T., CHILDS H., SHANKAR S., KRÜGER J., BERGERON R. D., HATCHER P.: Large Data Visualization on Distributed Memory Multi-GPU Clusters. In *Proceedings of High Performance Graphics 2010* (2010).

[HKRs*06] HADWIGER M., KNISS J. M., REZK-SALAMA C., WEISKOPF D., ENGEL K.: *Real-time Volume Graphics*. A. K. Peters, Ltd., 2006.

[IBM06] IBM: OpenDX: Open Visualization Data Explorer, 2006. http://www.opendx.org/.

[Ind09] INDIANA UNIVERSITY: Voxx: A Volume Rendering Program for 3D Microscopy, 2009. http://www.nephrology.iupui.edu/imaging/voxx/.

[Ins09] INSTITUTE S.:, 2009. SCIRun: A Scientific Computing Problem Solving Environment, Scientific Computing and Imaging Institute (SCI), Download from: http://www.scirun.org.

[KKH02] KNISS J., KINDLMANN G., HANSEN C.: simian, 2002. http://www.cs.utah.edu/~jmk/simian/.

[KKH05] KNISS J., KINDLMANN G., HANSEN. C.: *Multidimensional Transfer Functions for Volume Rendering*. Elsevier, 2005, pp. 189–210.

[KSW06] KRÜGER J., SCHNEIDER J., WESTERMANN R.: ClearView: An interactive context preserving hotspot visualization technique. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization / Information Visualization 2006) 12*, 5 (2006).

[KW03] KRÜGER J., WESTERMANN R.: Acceleration techniques for GPU-based Volume Rendering. In *Proceedings IEEE Visualization* (2003).

[MSRMH09] MEYER-SPRADOW J., ROPINSKI T., MENSMANN J., HINRICHS K.: Voreen: A rapid-prototyping environment for ray-casting-based volume visualizations. *IEEE Computer Graphics and Applications 29* (2009), 6–13.

[Nok09] NOKIA: Qt, 2009. http://qt.nokia.com/.

[Pau] PAUL B.: The Mesa 3D Graphics Library. http://www.mesa3d.org/.

[RBS05] RÖTTGER S., BAUER M., STAMMINGER M.: Spatialized transfer functions. In *EuroVis* (2005), pp. 271–278. http://dx.doi.org/10.2312/VisSym/EuroVis05/271-278.

[Ros09] ROSSET A.: Osirix, 2009. http://www.osirix-viewer.com/.

[RSEH02] REZK-SALAMA C., ENGEL K., HIGUERA F. V.: The OpenQVis project, 2002. http://openqvis.sourceforge.net/.

[SML06] SCHROEDER W., MARTIN K., LORENSEN B.: *The Visualization Toolkit An Object-Oriented Approach To 3D Graphics*, 4th ed. Kitware Inc., 2006.

[TXD*08] TIAN J., XUE J., DAI Y., CHEN J., ZHENG J.: A novel software platform for medical image processing and analyzing. *Information Technology in Biomedicine, IEEE Transactions on 12*, 6 (2008), 800–812.

[VGH*05] VIOLA I., GRÖLLER E., HADWIGER M., BÜHLER K., PREIM B., SOUSA M., EBERT D., STREDNEY D.: Illustrative visualization. IEEE Vis 2005, Tutorial #4, 2005.

[WDC*07] WEBER G. H., DILLARD S. E., CARR H., PASCUCCI V., HAMANN B.: Topology-controlled volume rendering. *IEEE Transactions on Visualization and Computer Graphics 13*, 2 (2007), 330–341.

[WZMK05] WANG L., ZHAO Y., MUELLER K., KAUFMAN A.: The magic volume lens: An interactive focus+context technique for volume rendering. In *Proceedings of IEEE Visualization* (2005), pp. 47–54.

[YAL*02] YOO T., ACKERMAN M., LORENSEN W., SCHROEDER W., CHALANA V., AYLWARD S., METAXAS D., WHITAKER R.: Engineering and algorithm design for an image processing api: A technical report on itk - the insight toolkit. *Insight Toolkit* (01 2002).