EngineRoom:
A Comparative Development Environment For Volume Rendering Engines

Michael A. Steinberg Klaus Mueller Robert Kelly

Center for Visual Computing, Department of Computer Science, Stony Brook University

Abstract

This paper describes Engine Room, a unifying system architecture for comparative volume rendering, fast prototyping of system components, and teaching. Engine Room's manager-driven architecture is able to: (i) dynamically dock or link multiple third-party volume rendering engines to a common environment, (ii) allow each linked engine to utilize the common environment's core functions; (iii) concurrently load, display, and navigate multiple volume datasets in separate windows using one or more of the rendering engines currently docked to the common environment; (iv) perform real time volume pre-viewing within the common environment for quick navigation and transfer function setup; and (v) share volume viewing parameters across renderers, allowing users to compare the relative quality of different rendering techniques in unbiased side-by-side window comparisons.

1 INTRODUCTION

Increases in computational power in concert with technological advances in volume rendering techniques foster an environment favorable to the practical application of volume rendering in industry and academia. As the technology required to support volume rendering develops and new volume rendering techniques continue to emerge, practical applications of volume rendering technology will continue to pervade industry and scientific research. Volume rendering currently serves as an essential analytical tool in medical imaging, computational fluid dynamics, engineering, and education.

Today's volume rendering approaches can be broadly classified into raycasting [7], splatting [16], shear-warp factorization [11], cell-projection [15], texture-mapping hardware assisted [6], and custom hardware-based [13]. Research following existing and future volume rendering approaches is matched by the concurrent need to create practical implementations capable of utilizing these concepts. In fact, each of these volume rendering approaches could be merged into a common environment. This would facilitate: (i) the unbiased comparison of new techniques with present techniques, (ii) the utilization of new techniques to assist in actual visualization tasks, and (iii) the quick integration of existing and new techniques to augment a comprehensive visualization system.

While a new volume renderer, or *rendering engine*, can consist of less than one-hundred lines of source code, the required support functions, such as user interface, image display, volume navigation, and transfer function editor, can easily exceed several thousands lines of source code. Since most volume rendering techniques are constructed on the same scientific premise, the support functions required for these volume rendering engines are essentially identical. Therefore, the one-time development and continuous reuse of such support functions makes practical and economic sense. Applying the same support functions to all volume rendering techniques will also result in a common look-and-feel across all volume renderers, flattening the learning curve for users new to a specific volume renderer.

In recent years, volume rendering researchers have displayed great interest in developing frameworks and methodologies for the comparative study of new and existing volume rendering algorithms [10][17][8][12]. These efforts have focused on the comparison of volume rendering speed and quality. A popular approach for the latter is the definition of a suitable error metric for comparing rendered images (or partial results within the rendering pipeline) produced by contending renderers. However, as was shown by Gaddipatti, Machiraju, and Yagel [8], numerical metrics that do not consider the human observer within the loop are prone to fail, and they consequently introduced a framework that seeks to model the human perceptional system. Nevertheless, the simplest and perhaps most reliable form of comparison may be the presentation of images to a statistically relevant group of users, perhaps domain experts, for side-by-side comparisons. Although this approach may seem simple in principle, the infrastructure required to accomplish this task can be tedious, particularly if the user has full control over the rendering parameters. If each volume renderer is implemented in a separate package (which is plausible if the renderers are sufficiently complex and therefore hard to replicate) then (i) the viewing parameters and transfer functions used may be different for each renderer due to heterogeneous implementations, (ii) the user may have to learn a number of different interfaces, and (iii) side-by-side comparisons may be cumbersome if the display interfaces of the individual renderers do not tile or overlap well on the screen. The latter may necessitate the printing of images on a tiled comparison sheet viewable by the user, which is quite cumbersome to produce.

Engine Room is a current research project aiming to establish guidelines for good design of volume rendering system architectures. Engine Room contains all of the support functions needed by most volume rendering applications and enables users to easily add new components via a straightforward plug-in mechanism. The application's ability to spawn identical and cooperating user interfaces for each volume renderer also facilitates unbiased visual and runtime comparisons.

Our paper is organized as follows: First, in Section 2, we provide a summary of related work. Next, Section 3 illustrates the basic required components of a general volume rendering applications. Section 4 provides a detailed description of all components of Engine Room's architecture. Section 5 discusses several key applications for which Engine Room would be an ideal user environment, and Section 6 concludes with final remarks.

2 RELATED WORK

The idea of docking an application to a generic support mechanism is standard practice in application development. For example, in the field of graphics, OpenInventor [3] nodes can be implemented as DSOs (Dynamic Shared Objects), allowing the docking of an active rendering module, compiled into the DSO, to a visual interface such as SceneViewer [3]. Toolkits, such as VTK [14], use Tcl/TK or other scripts to link modules together into a working application. A more visually oriented interface for this linking task is provided by packages that exploit the visual datapath paradigm. Popular

systems in this group include Khoros [2], AVS [1], and OpenDataExplorer [4].

In our work, we are not suggesting to create yet another application builder or library toolbox. In fact, our application could probably be, with modest effort, implemented with any of the existing frameworks. (We chose a more "down-to-the roots" approach, using C++ for programming and FLTK [5] for constructing the GUI, simply to maintain a higher level of flexibility.) Rather, our goal is to design and demonstrate an effective system configuration, architecture, and communication flow optimized specifically for comparative volume rendering (and possibly general graphics) with the capacity for component prototyping, system evolution, production use, and teaching.

3 GENERIC ARCHITECTURE

Our development process starts by evaluating the design of a generic volume rendering system architecture, as illustrated in Figure 1. Using this architecture, the user can select and render an arbitrary volume file using the rendering engine available to the system. The GUI is equipped with a number of global and local support functions, including interfaces for navigation, editing of transfer functions, and configuring renderer-specific parameters. A display window is provided to view the rendered image.

This generic architecture is limited to exploring a single volume dataset, during which time the volume's transfer function can be modified. Although this architecture may be sufficient for many applications, there are many scenarios (some of them stated in the introduction) that call for a more elaborate architecture. For example, the user may wish to:

- view a volume from several different viewpoints simultaneously,
- view several different volumes (perhaps time-varying) from the same viewpoints simultaneously,
- compare the rendering results of two or more engines, designating one engine for navigation, transfer function specification, etc., while the remaining engines follow in tandem,
- perform a quick pre-viewing using a dedicated low-cost engine and another higher-quality engine for image generation,
- incrementally develop a new engine, or any other support function, in a well equipped support test bed.

A number of important issues ensue from this wish list:

- How are volume and other data shared among rendering engines.
- How are state and support function data, such as viewing parameters and transfer functions, communicated to and among rendering engines.
- How is the system state maintained and saved.
- How are associations among the individual system components maintained and created, and how are components and engines quickly integrated into the existing system.

Of course, not all functionality will be required by all users. For example, a general user will not need to develop a new rendering engine, but may acquire a new renderer or other component (fitted to EngineRoom's specifications) and may want to quickly incorporate these into the system in a plug-and-play fashion.

In the following section, we describe the solution that Engine Room provides to address these issues.

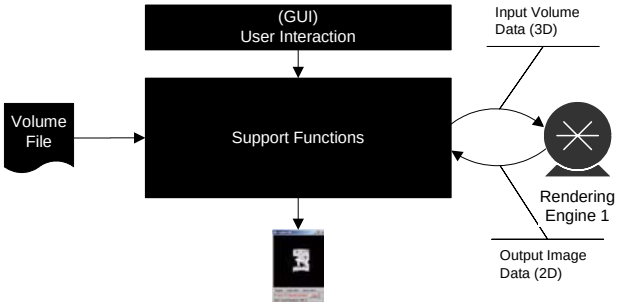

Figure 1: Generic Volume Rendering Architecture

4 ENGINE ROOM ARCHITECTURE

The core of Engine Room is comprised of a series of C++ objects called *managers.* Each manager is responsible for controlling a specific subset of Engine Room's functionality. The managers currently incorporated in Engine Room include: *Window Manager*, *Engine Manager*, *Volume Manager*, *View Manager*, *Transfer Function Manager*, and *Preferences Manager*.

The relationship between each of these managers is illustrated in Figure 2. In essence, the manager-driven architecture fills the Support Functions box in Figure 1. Every operation performed by Engine Room is delegated to a specific manager or subset of managers based on related functionality. Inter-manager communication is employed for tasks requiring the services of multiple managers.

The manager-driven-architecture employed in Engine Room has the advantage of maximizing Engine Room's extensibility. Any of Engine Room's managers can easily be updated or swapped with new managers that provide similar services. We shall now describe the functionality of each manager in turn.

4.1.1 Window Manager

Engine Room provides the capability to render into multiple display windows, whereby each window is attached to the same or a different rendering engine. Window Manager plays the most critical role in Engine Room's architecture. It is responsible for managing all window creation, destruction, and volume renderings performed by Engine Room. Window manager is also responsible for routing manager-specific requests to other Engine Room managers, such as the selection of a volume rendering engine.

Each window is associated with a *Win* object. Window Manager maintains an array of Win objects, where one Win object is assigned to each newly opened volume instance, associated with a newly opened rendering window. Each Win object contains a pointer to a volume object, a display window, and a transfer function data structure and transfer function editor object.

4.1.2 Engine Manager

Importing Rendering Engines: Engine Manager is responsible for linking all available third-party rendering engines to Engine Room. This manager also owns each window's rendering engine selection menu used to request an engine change, in which case a

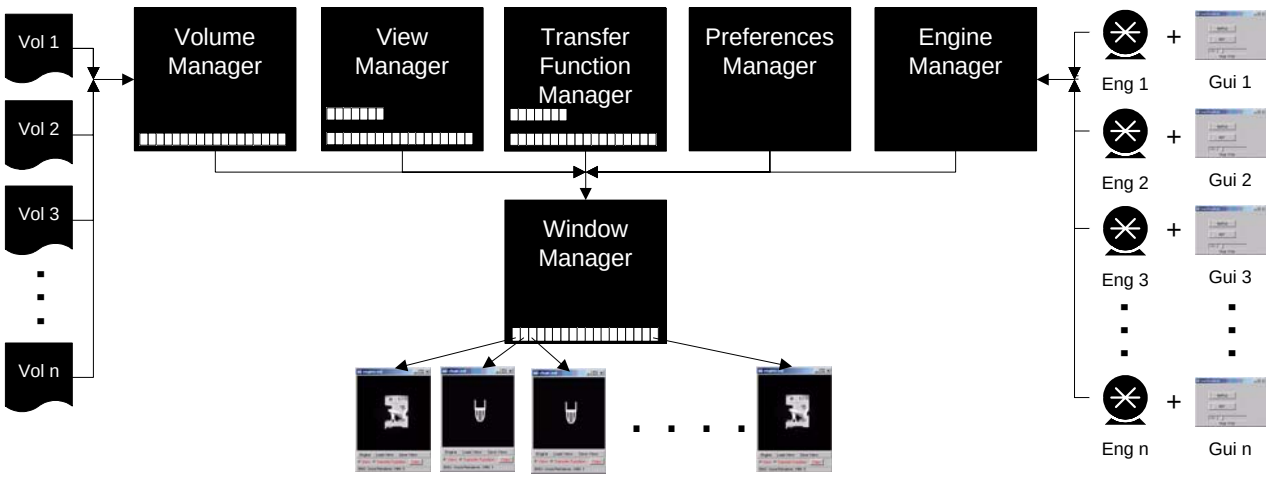

Figure 2: Engine Room Core Functions

new association from window to selected engine is established. The engine can access the window's Win object.

Engine Manager detects new rendering engines using dynamic linked libraries (DLLs) or DSOs (on Unix and Linux), thus eliminating the need to recompile Engine Room when a new or updated rendering engine is added.

Rendering Engine Specific GUIs: Engine Room provides support for a unique GUI control panel for each linked third-party rendering engine. Custom GUIs can be easily designed using *Fluid*, the visual GUI editor of FLTK (Fast Light Toolkit) [5], used to design all of Engine Room's GUIs.

4.1.3 Volume Manager

Volume Manager is responsible for storing and managing each volumetric dataset loaded into memory. Storing volume data independently from other parameters allows multiple open windows to render the same volumetric data set in memory using different rendering engines for side-by-side comparisons.

4.1.4 View Manager

View Manager is responsible for storing the state (view parameters) of each window. State variables include: pointers to each volume data structure, image data structure, volume and image dimensions, rotation in X-Y-Z space, current translation in X-Y space, magnification, perspective, and the like.

The view parameters for a window are stored in a *View* object. View Manager maintains an array of View objects, and one View object is assigned to each newly opened window.

4.1.5 Transfer Function Manager

The Transfer Function Manager is responsible for storing and retrieving the transfer function for each volume instance and rendering window. This manager owns the linked transfer function editors, selectable by a window menu, and all related transfer function methods. A DLL mechanism is employed to link available transfer function editors into Engine Room.

The transfer function for a single open volume file is stored in a *TransferFunctionEditor* object. Transfer Function Manager maintains an array of TransferFunctionEditor objects, where one TransferFunctionEditor object is assigned to each newly opened rendering window.

4.1.6 Preference Manager

Preferences Manager contains functions that can globally adjust the look-and-feel and functionality of any existing manager in Engine Room, for example its size or tile color.

4.1.7 Inter-Window Communication

Rendering windows communicate via global clipboards which can be employed to store, retrieve, and transfer viewing parameters, transfer functions, and other variables. To bypass the clipboard cut-and-paste mechanism and to provide a tighter coupling of rendering windows, a check box on each window is available to specify window associations. In this way, any state change (viewing, transfer function, or other) in one window will be automatically propagated to the other associated windows and their respective engines, and will trigger an immediate rendering based on the new state.

5 APPLICATIONS

We have identified several applications for which Engine Room would constitute a well suited environment.

Prototyping: Engine Room promotes easy and rapid prototyping of volume renderers by providing support for the simple docking of one or more experimental volume renderer(s) to the support architecture, granting immediate access to all interface functionality. In the (rather common) case that the docked experimental volume renderer is computationally intensive and non-interactive, resulting in a prohibitively long rendering time, the user may opt to first employ one of Engine Room's optimized real-time volume pre-viewers to select an interesting view or transfer function, and then switch to the experimental renderer to view the rendering result using these selected parameters. Prototype construction is completed by designing (an optional) engine GUI to provide controls unique to the experimental renderer, using FLTK's easy visual GUI builder *Fluid.*

Comparative studies: Engine Room's support for side-by-side display window tiling of multiple volumes, each rendered using a different rendering engine, coupled with the ability to share viewing parameters and transfer functions via a common clipboard, provides an effective environment for comparative studies. Since Engine Room's interface is identical for all renderers (except for private engine specific GUIs), the learning curve required to gain

proficiency in using all of the docked rendering engines is greatly reduced. Engine Room's ease-of-use expands the range of potential users to include non-programmers and novice-users. The inclusion of industry specific non-technical users in comparative studies, such as physicians, can be used to collect and generate more robust and representative statistical results.

Teaching: In an introductory visualization course, students often spend considerable time implementing user interface and support functionality, time that would be better spent focusing-on and implementing actual visualization algorithms. On the other hand, providing a skeleton program containing all of the required support functionality for a complete application often leads to considerable confusion and frustration due to the large amount of unfamiliar code that must be comprehended in order to finish a programming assignment. Engine Room's architecture hides all support code from the student programmer and only requires a minimal amount of interaction with the host architecture, preventing the problems associated with the traditional teaching approach, and resulting in a more-focused and structured learning environment for visualization. As a teaching tool, students can be provided with a pre-compiled model rendering engine for side-by-side comparisons with their own programming assignments. In a graduate level visualization course, Engine Room can be used as a platform to create and integrate other advanced visualization components and functionality.

6 CONCLUSIONS

Engine Room provides a proof-of-concept study of a fully extensible, dynamic, intuitive volume rendering environment. Engine Room's architecture gives rendering engine developers the ability to fully utilize newly developed rendering engines with minimal programming overhead. General users involved in volume rendering research can use Engine Room to easily access multiple rendering engines in a single application. Volume visualization researchers can use Engine Room as a vehicle to create new volume visualization tools and utilities.

The development of Engine Room is an on-going research project, and we plan to refine and expand all of engine room's support functionality as time progresses. A public open-source release of the software is planned. The development of Engine Room also spawned a number of other future research topics that are worthy of consideration. First on this list is research into the determination of how much data sharing is possible among active volume renderers. If multiple volume renderers require access to the raw data of a single volume file, the ideal solution would involve providing all renderers access to a single copy of the loaded raw data. However, depending on each rendering engine implementation, this type of ubiquitous data sharing is not always possible. Some rendering engines may write directly to the raw data, making the data set unusable by other rendering engines. Also, renderers often reformat the data into an own renderer-specific representation, useless for other renderers, and discard the original data. A second area of research is the expansion of Engine Room to operate on parallel architectures, possibly even over networks, where a single user could control an array of distributed volume renderers using Engine Room as a client application. A third area of potential research is the incorporation of numerical quality metrics to better gauge rendering quality using an objective user-independent methodology. Finally, Engine Room's architecture is not constrained strictly to volume rendering applications, but could be applicable to general graphics engines for the purposes of teaching, prototyping, or comparative studies.

ACKNOWLEDGEMENTS

We would like to thank the FLTK consortium for their continued effort in advancing the FLTK software.

REFERENCES

1. http://www.avs.com
2. http://www.khoral.com
3. http://www.sgi.com/software/inventor
4. http://www.opendx.org
5. http://www.fltk.org
6. B. Cabral, N. Cam, and J. Foran, "Accelerated volume rendering and tomographic reconstruction using texture mapping hardware", *1994 Symposium of Volume Visualization*, pp. 91-98, 1994.
7. M. Levoy, "Efficient ray tracing of volume data", *ACM Trans. Comp. Graph.*, vol. 9, no. 3, pp. 245-261, 1990.
8. A. Gaddipati, R. Machiraju, R. Yagel "Steering Image generation using Wavelet Based Perceptual Metric*", Computer Graphics Forum*, vol. 16, no. 3, pp. 241-251, 1997.
9. J. Kniss, G. Kindlmann, C. Hansen, "Interactive volume rendering using multi-dimensional transfer functions and direct manipulation widgets," *Proc. Visualization'01*, pp. 255-262, October 2001.
10. K. Kim, C. Wittenbrink, A. Pang, "Data level comparison of surface classification and gradient filters, " *Proc. Workshop on Volume Graphics 2001*, pp. 19-34, Stony Brook, NY, June 21 – 22, 2001.
11. P. Lacroute and M. Levoy, "Fast volume rendering using a shear-warp factorization of the viewing transformation", *Proc. SIGGRAPH `94*, pp. 451- 458, 1994.
12. M. Meissner, J. Huang, D. Bartz, K. Mueller, R. Crawfis, "A practical comparison of popular volume rendering algorithms," *2000 Symposium on Volume Rendering*, pp. 81-90, Salt-Lake City, October 2000.
13. H. Pfister, J. Hardenbergh, J. Knittel, H. Lauer, and L. Seiler, The VolumePro Real-Time Ray-Casting System, *Proceedings of SIGGRAPH 99*, p. 251-260, Los Angeles, CA, August 1999.
14. W. Schroeder, K. Martin, B. Lorensen. *The Visualization Toolkit: An Object-Oriented Approach To 3D Graphics, 2nd edition*. Prentice Hall: Saddle River, 1998.
15. P. Shirley and A. Tuchman. "A polygonal approximation to direct scalar volume rendering," *Computer Graphics*, vol. 24, no. 5, pp. 63-70, Workshop on Volume Visualization, 1990.
16. L. Westover, "Footprint evaluation for volume rendering", *Proc. SIGGRAPH'90*, pp. 367-376, 1990.
17. P.L. Williams and S.P. Uselton, "Metrics and generation specifications for comparing volume-rendered images," *The Journal of Visualization and Computer Animation, vol. 10, pp. 159-178, 1999.*

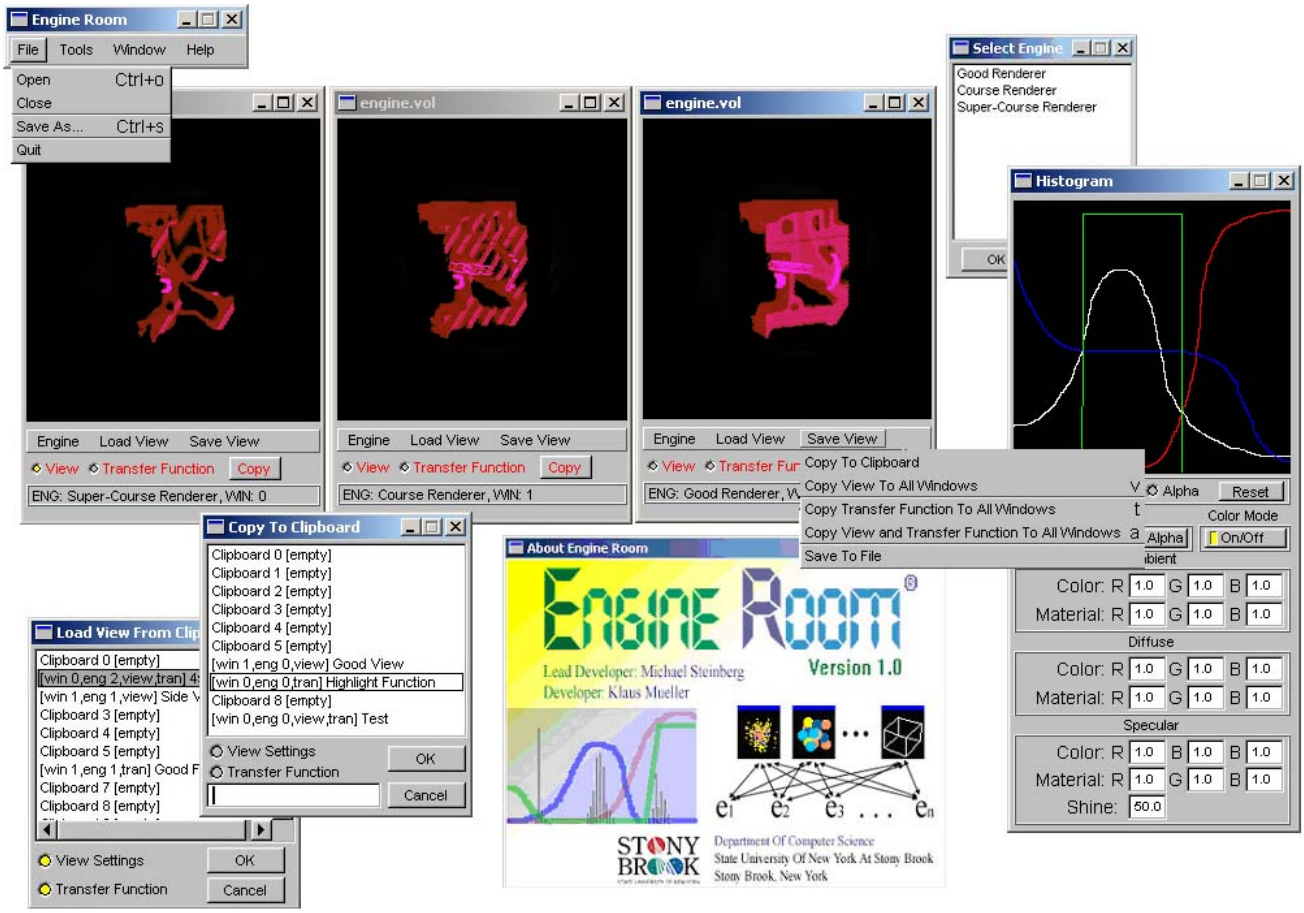

Figure 3: Mosaic of Engine Room Screenshots