Using Photorealistic RenderMan™ for High-Quality Direct Volume Rendering

Cyrus Jam
cjam@sdsc.edu

Mike Bailey
mjb@sdsc.edu

San Diego Supercomputer Center
University of California San Diego

Abstract

With the success of Pixar's recent feature films, everyone knows RenderMan to be the leading photorealistic renderer for animation and entertainment. What most people don't know is that it can also be used quite effectively as a direct volume renderer. Many of the same qualities that make it excellent for entertainment rendering also make it excellent for displaying and animating volumes. The fact that it is a commercial package that is well-supported and well-documented is an added bonus. This paper shows how to use RenderMan in this way and shows several example images.

Keywords: Computer Graphics, Scientific Visualization, Volume Rendering, Volumetric Imaging, Rendering Algorithms

Introduction

Direct volume rendering us a core tool in a visualizer's toolkit. By rendering the entire volume directly, the user can manipulate the transfer function to reveal key details in the volume dataset. Direct volume rendering has included image-based approaches such as ray tracing [1], and object-based approaches such as splatting [2]. The direct volume rendering methods have all approached the problem with the idea that a quality rendering is more important than reducing the time necessary to get it.

These methods, and others, are well-developed, but generally exist only in specialized research environments where they must be internally developed, maintained, and documented. Commercial direct volume rendering solutions are rare, and, where they exist, are quite costly.

This project investigated the use of the RenderMan photorealistic rendering system as a direct volume renderer. RenderMan is maintained and documented by Pixar [3,4]. It is stable. It has considerable functionality and flexibility. It is less expensive than commercial volume rendering packages. There is even a no-cost version available in the form of BMRT, the Blue Moon Rendering Tools [5].

Importing the Volume

The first step in importing the volume dataset was to extract slices from the voxel data and map them into TIFF images. Slice sets were taken throughout the entire dataset. Three sets of slices were taken, each parallel to a principle axis, x, y, and z. These TIFF images conserve the fundamental proprieties of the volume, specifically, color (R, G, B) and opacity (A). Because Prman (Pixar's RenderMan) requires that, "all textures be in a special, proprietary texture format" [4], it is then necessary to convert every TIFF file to a format that Prman can understand, which can be accomplished by using the API call *MakeTexture*.

Once the slices were pre-processed, they were mapped onto polygons. This was done with a RenderMan shader that maps the RGBA of the images and discards any properties of the polygon. Next, by positioning all of the slices of the volume in the correct relation to each other, the actual dimensions of the original dataset were recreated. This was easily done by confining the slices to a depth (first slice to last) equal to the depth value of the volume dataset. However, if this was not done properly, a considerable amount of image warping occured. This warping happened most when viewing the volume at angles approaching a parallel alignment with the slice planes. The images were compressed or widened. Therefore, it were important to use high precision when calculating the slice offsets. This composition of slices works as one object within the scene. The notion of slices then became completely transparent to the user.

Once the data was imported, a unique problem arose. When the viewing position became parallel with the slicing planes, the volume could no longer be seen, not looking through a group of composite slices but looking directly between the slices. To solve this problem we needed to track *exactly* where the viewer was located in relation to the volume and to decide which slicing plane is best suited for this location. The Threshold area, as shown in Figure 1., occurs at angles where the appropriate slicing plane to composite is disputable.

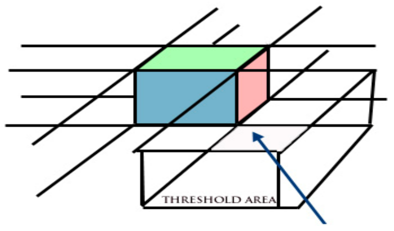


Figure 1: Threshold Viewing Direction

When the volume dataset was imported into a RenderMan scene, it then became possible to apply any of RenderMan's countless features to the volume. This was one of the major reasons to undertake this project. In this paper we have listed a few of these key features to produce real working examples.

Application: Arbitrary Viewing Angle, With or Without Perspective

Using the threshold viewing direction method shown above, the volume can be rendered from any location with no loss in quality. Also, because the volume is mapped onto RenderMan geometry, it can be displayed in either orthographic or perspective projections. The ability to display a direct-rendered volume in perspective is not easy for many volume rendering packages. But, because RenderMan pre-fractures the scene into the appropriate number of microfacets for the given viewing volume, here it comes automatically. Figures 2a-2d show a human head dataset viewed in perspective from 0°, 15°, 30°, and 45°.

Application: Arbitrary Resolution

RenderMan renders its scene at an arbitrary resolution specified by the user. When that scene includes texture maps, such as in this case, most display systems resort to bilinear interpolation to produce color values between the texels. RenderMan, instead, uses a bicubic interpolation. The result is a very consistent image, regardless of the resolution. Figures 3a-3c show the same human head dataset rendered at 500^2, 1000^2, and 2000^2 pixels. While the quality clearly increases, the smoothness of the 500^2 image is more than adequate.

Incidentally, the arbitrary resolution-ness of RenderMan extends to the size of the texture maps as well. This means that volume datasets of arbitrary size can be rendered with no change in the method, albeit with a cost in time.

Application: Shadows

"Shadows provide very important visual cues in the images we see. They clearly show the physical relationship and proximity of different objects. Rendering a 3D scene with shadows can provide almost as much information as a pair of stereo images" [6]

In general, direct volume rendering tends to leave out shadows because of the immense complexity of this task, even though shadows provide many useful applications in scientific visualization.

Creating shadows from volumes using RenderMan requires the help of both Prman and BMRT. The Blue Moon Rendering Tools, or BMRT for short [5], is a RenderMan-compatible renderer that provides extra features such as ray tracing and radiosity. Ray traced shadows from BMRT enable the volume to produce correct shadows by taking into account the opacity values throughout the volume slices. Prman primary handles shadows using Shadow Maps. Though this approach is much faster than that of BMRT, this technique can only take into account the geometry of the object. Therefore this would produce shadows of only the polygon slices.

By setting the shadow attributes to “on” in a BMRT attribute call, the user can automatically cast shadows in the final rendered scene. However, as stated above, this process alone can be very time consuming. A better approach is to use Prman for the bulk of the scene computations and BMRT for the ray tracing. This can be conveniently done using what is called a “ray server”. Prman can send queries that need to be executed to the “ray server”(BMRT) through *stdin*. The “ray server” will then compute the result and return back to Prman through *stdout*. With this, the best features of both renderers are used to efficiently produce accurate shadows of volumes. Figure 4 shows an example of a volume dataset that has a spotlight placed in front and casting a shadow onto the back wall.

Application: Motion Blur

Motion blur can be easily applied to any volume using RenderMan. This effect introduces another aspect of realism, which is traditionally not applied to volume datasets. Using RenderMan’s unique set of API calls, it is simple for the user to create an animated sequence of a volume using the same properties as a live-action camera. Using an arbitrary time scale (user defined), specify when the camera should open and close its shutter. Depending on the transformations applied during these shutter intervals RenderMan will automatically produce the correct, realistic effect. Figure 5 shows an example of motion blur applied to a volume dataset.

At first glance this looks like a gratuitous use of fun graphics, but in fact it is actually useful. Motion blur is basically a smeared interpolation from one dataset and transformation to another dataset and transformation. Thus, when working with a time-based set of volumes, this would be a way to show the transition across volume sequences in a single output image, or a way to smoothly transition in an animation.

Application: Depth of Field

Similar to motion blur, depth of field is another feature of RenderMan which can be applied to a volume dataset to achieve images that traditionally could not be easy created. By using the API call *DepthfField*, one can simulate the features of a live-action camera: length of the lens, the distance at which the camera is focused, and the diameter of the aperture (f-stop). Figure 7a-d shows a series of images with depth of field applied at various distances.

Application: Procedural Shading of Volumes

Another feature we gain by importing volumes into RenderMan comes out of one of RenderMan’s most important features, the shader. A shader is written in the RenderMan Shading Language and is a way to procedurally define the interactions of lights and surfaces.

There are five basic types of shaders. These are: surface, displacement, light, volume, and imager shaders. The general way a shader works is as follows. First there is an assortment of global values that are passed into the main shader routine. These values specify attributes such as point surface color and surface opacity. In our case the only values that we immediately find important are the texture maps color and opacity values. Then within the shader we are able to manipulate these values in any way. The final goal of each pass of the shader is to assign this input color and opacity to an output color and opacity.

In the most basic case this occurs when a texture is placed onto a surface with no alterations. The input color and opacity for each point of the texture is procedurally placed onto the polygons surface with the shader.

A more creative approach would be to apply transformations to the input color and opacity to produce interesting images. For example, you could write an unlimited number of shaders, from glass to wood surfaces, etc.

Speed

One problem that exists for volume rendering, and rendering in general, is the amount of time it might take to compute one single image. It would be very useful to have a method of approximating images initially to preview what the final frame will look like. With RenderMan this functionality is built in and can be easily used.

There are four options in particular that are useful for our purposes; pixel filter, shading rate, number of samples and final image resolution. The pixel filter takes the rendered image and smoothes out the 2D frame. The larger the pixel filter the softer the result becomes. Shading rate can be described as the "number of shading calculations per primitive." [3] Because imported volume data has such a large reliance on shading, increasing the shading rate significantly speeds up the time it takes to render a frame, although, a noticeable increase of texture artifacts can be noticed. The number of samples is the "effective sampling rate in the horizontal and vertical directions." [3] RenderMan will take slightly jittered pixel samples and average there values together. Using a call to *Format* one can control the output image resolution. The time to render an entire image is closely related to the number of pixels in the scene. "Rendering a 256x256 image will take approximately one quarter the time required to compute a 512x512 equivalent." [7]

For low-quality images suitable for previewing a scene's geometry, we found that a pixel filter width (1, 1) a shading rate of 16 and sampling rate of (1, 1) are sufficient enough to achieve a relative fast output. For high-quality renderings we used approximately a pixel filter of (6, 6) a shading rate of 1 and a sample rate of (10, 10).

Conclusion

In this paper we have presented the general strategy of importing volume datasets into RenderMan and a description of some of the most useful tools this empowers us with. The advantages in this are:

- Arbitrary input voxel resolution
- Arbitrary pixel image resolution
- Arbitrary viewing parameters, including perspective projection
- Able to combine a volume and hard geometry in the same scene
- Shadows
- Motion blur
- Depth of field

By harnessing the features and stability of RenderMan, we have been able to achieve high-quality images of volume datasets without the need for custom software. On top of this, we have also gamed the ability to produce images with features that were not generally available for volume graphics such as shadows and motion blur.

References

[1] J. Genetti and D. Gordon and G. Williams, "Adaptive Supersampling in Object Space Using Pyramidal Rays", *Computer Graphics Forum*, Vol 17, Number 1, pp. 29-54, 1998.

[2] Lee Westover, "Footprint Evaluation for Volume Rendering", *Computer Graphics* (Proceedings of SIGGRAPH 90), Volume 24, Number 4, pp. 367-376.

[3] Steve Upstill, *The RenderMan Companion, A Programmer's Guide to Realistic Computer Graphics*, Addison-Wesley, 1990

[4] Anthony Apodaca and Larry Gritz, *Advanced RenderMan: Creating CGI for Motion Pictures*, Morgan Kaufmann, 2000

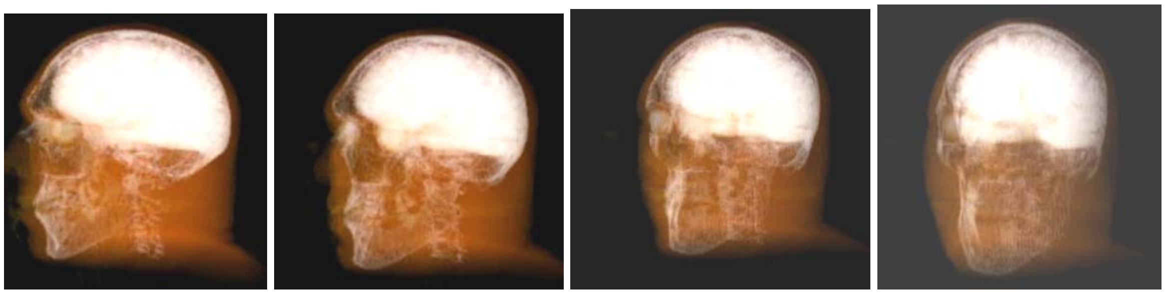
[5] Larry Gritz. *Blue Moon Rendering Tools, User Manual, Release 2.6*, 2000. Web site:

http://www.exluna.com/bmrt/bmrtdoc/2.6/index.html

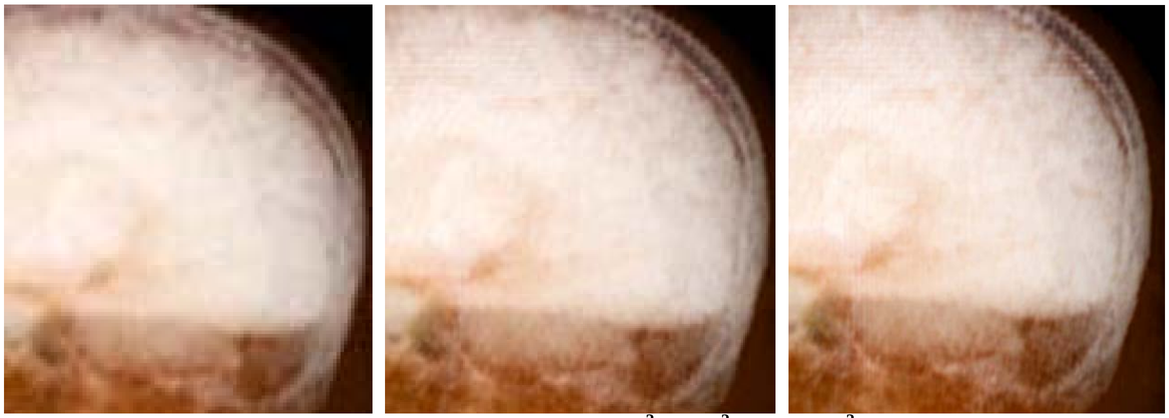
[6] P. Haeberli. Grafica Obscura, Collected Computer Graphics Hacks: A Note on Shadows, 2000. Web Site: http://www.sgi.com/grafica/shadows/index.html

[7] Pixar. Photorealistic RenderMan 3.8 Users Manual, 2000. Web Site: http://www.pixar.com/products/RenderMandocs/toolkit/Toolkit/user.html

Note to reviewers: We are currently working on an animation using these methods. It is looking very promising! If this paper is accepted, we will show the videotape in the presentation at the conference.



Figures 2a-d: Volume Dataset Viewed from 0°, 15°, 30°, and 45°



Figures 3a-c: Detail of Head Rendering at 500^2, 1000^2, and 2000^2 Pixel Resolution

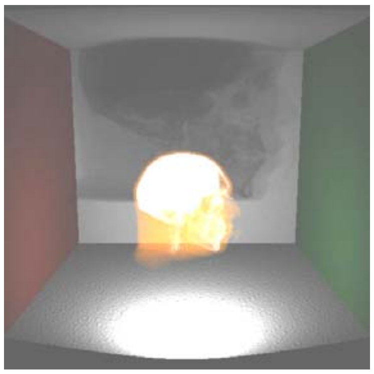


Figure 4:
Volume Shadows

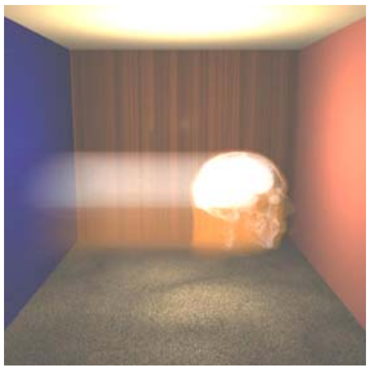


Figure 5:
Volume Motion Blur

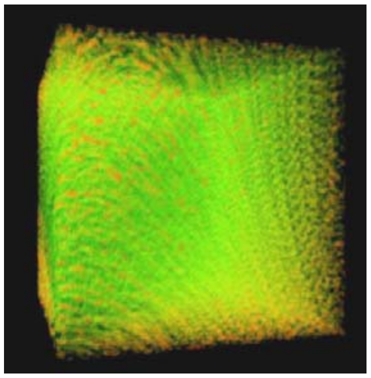
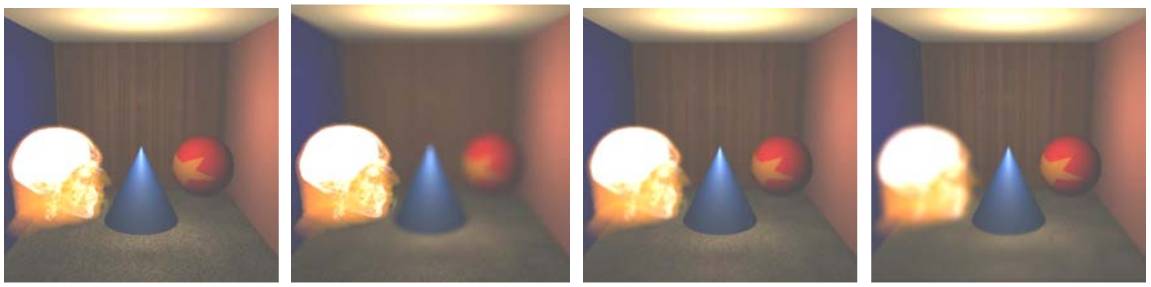


Figure 6:
3D LIC Volume



Figures 7a-d: Combining Volume with Hard Geometry and Displaying with Depth-of-Field