PERGAMON

Computers & Graphics 27 (2003) 725–734

COMPUTERS & GRAPHICS

www.elsevier.com/locate/cag

Graphics hardware

Recent advances in hardware-accelerated volume rendering

Kwan-Liu Ma[a,*], Eric B. Lum[a], Shigeru Muraki[b]

[a] *Department of Computer Science, University of California at Davis, 2063 Engineering II, One Shields Avenue, Davis, CA 95616 8562, USA*

[b] *National Institute of Advanced Industrial Science and Technology (AIST), Aomi 2-41-6, Koto-Ku, Tokyo 135-0064, Japan*

Abstract

The programmability and texture support of consumer graphics accelerators have drawn a lot of attention from visualization researchers, resulting in some very important advances in interactive volume data visualization. For many applications, scientists can now perform routine data visualization and analysis tasks on their desktop PC with a consumer graphics card that was designed mainly for playing video games. This paper presents several representative hardware-accelerated algorithms that have been introduced recently to address the problems of classification, illumination, non-photorealistic rendering, decoding, and image compositing in volume data visualization.
© 2003 Elsevier Ltd. All rights reserved.

Keywords: 3D texture; Data compression; Data visualization; Graphics hardware; Image processing; Volume rendering

1. Introduction

Volume rendering is a powerful technique for visualizing sampled data describing physical phenomena or structures in 3-space, from molecular structures and dynamics, neuron structure, the anatomy of the human body, chemical reactions inside a furnace, air flow surrounding a vehicle, ocean temperature distribution, to the birth of our solar system. The advent of hardware support for real-time volume rendering has made this 3-D rendering technique even more attractive for a growing range of applications. Notable examples include SGI RealityEngine's support of texture mapping [1], the VolumePro volume rendering acceleration board [2], and reconfigurable volume rendering systems such as VIZARD II which uses FPGA for fast design changes [3]. More recently, driven by the video game industry, advances and innovations in consumer graphics hardware have been made at a rather fast pace. The low cost and high performance of the consumer graphics cards have led to many creative uses of several advanced features, such as high precision arithmetic and programmability, for volume graphics applications.

This paper gives an overview of the evolving commercial hardware support for volume rendering and samples of representative research results in advancing the art of hardware-accelerated volume rendering. Specifically, we describe how graphics accelerators can be used to assist volume classification, how illumination and non-photorealistic rendering (NPR) can be added to increase the clarity of the visualization, how to accelerate the rendering of time-varying data, and different hardware options for the construction of a cluster of PCs for interactive volume graphics applications. Finally, we suggest directions for further research.

2. Texture hardware features and volume rendering

Volume rendering involves resampling, classification, shading, and compositing. The latest consumer graphics cards designed mainly for playing video games can accelerate almost all the volume rendering calculations, making possible real-time rendering rates.

*Corresponding author. Tel.: +1-530-752-6958; fax: +1-530-752-4767.

E-mail address: ma@cs.ucdavis.edu (K.-L. Ma).

0097-8493/$ - see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S0097-8493(03)00146-8

With texture hardware support, volume rendering is done by drawing a set of view-aligned polygon slices that sample a 3-D texture containing the volume data, as shown in Fig. 1. These slices are composited using hardware alpha-blending to derive the final image. The resampling step of volume rendering uses a reconstruction filter kernel, and the shape of this filter can have a strong influence on the quality of the resulting visualization. Older graphics hardware was limited in supporting only 2-D textures and, thus, performing bilinear interpolation. Employing 3-D textures permits the use of tri-linear filtering which helps improves image quality. Hadwiger et al. [4] show how to employ the programmable features of the graphics cards to perform higher quality filtering such as bi-cubic filtering using a B-spline filter kernel.

In exploratory data visualization, it is desirable to freely change viewing and rendering parameters and immediately receive visual feedback. Since the performance of a texture hardware volume renderer is mainly limited by the fill rates and the data transfer rates between the main memory and video memory, in order to achieve maximum interactivity we must carefully utilize the available video memory space. Since the volumes being visualized are typically tens or hundreds of megabytes in size, we cannot afford to store classified volume (i.e., RGBA data for every voxel) whenever a new classification is done. Similarly, for lighting calculations storing the gradient vectors is also not feasible. Section 4 describes two techniques for deriving gradient values for lighting calculations.

Storage space can be saved by making use of lookup tables in the form of either dependent textures or paletted textures. Dependent textures treat the values in one texture as texture coordinates into a second lookup texture. By treating per voxel volume information as texture coordinates, and storing lighting or transfer function information in the lookup texture, it is possible to change the appearance of an entire volume through variations in a relatively small lookup texture. An alternative method is the use of paletted textures, where each texel stores an index into a color lookup table that is applied during rasterization. The primary difference between the two methods is that dependent textures require the use of two textures, and the filtering is done prior to the table lookup, which can be contrasted with paletted textures that require a single texture and are filtered during rasterization using color values after the lookup have occurred. Since filtering typically involves linear interpolation it is desirable to use paletted textures for quantities that have non-linear mappings between index and color such as normals stored using a single-index lookup table, or for temporal compression as will be described in a later section. Fig. 2 illustrates the difference between these two methods. The use of paletted textures is discussed further in Section 7.2.

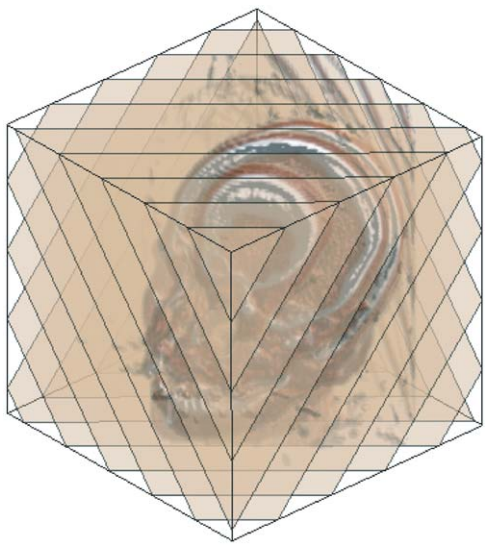

Fig. 1. Texture hardware volume rendering is done by drawing a set of view-aligned polygon slices that sample a 3-D texture volume containing the volume data.

Multi-texturing permits several textures to be combined on a single polygon during the rendering process. By utilizing several separate volumetric textures that store, for example, scalar data value, gradient magnitude, and gradient direction, it is possible to combine these contributions for enhanced visualizations. We describe how to mix different rendering styles with multi-texturing in Section 5. Furthermore, the programmable feature recently added to consumer graphics hardware provides far more flexibility in how the different texture data is combined during rendering. Specifically, fragment programs in OpenGL and pixel shaders in Direct-3D allow for the use of programs that are executed on a per pixel basis during rasterization. This functionality provides the developers with further control over how the texture data and linearly interpolated vertex attributes, like color and texture coordinates, are combined.

3. Classification

In volume rendering, each voxel must be first mapped to a color and an opacity value before the projection and compositing calculations are done. This mapping is equivalent to a classification of the volume. There has been a great deal of research devoted to the generation of transfer functions for volume classification [5].

A straightforward method for implementing a 1-D transfer function that maps each scalar value to a color and opacity is to use paletted textures, with the palette consisting of the transfer function, or 1-D dependent textures, with the 1-D lookup textures containing the

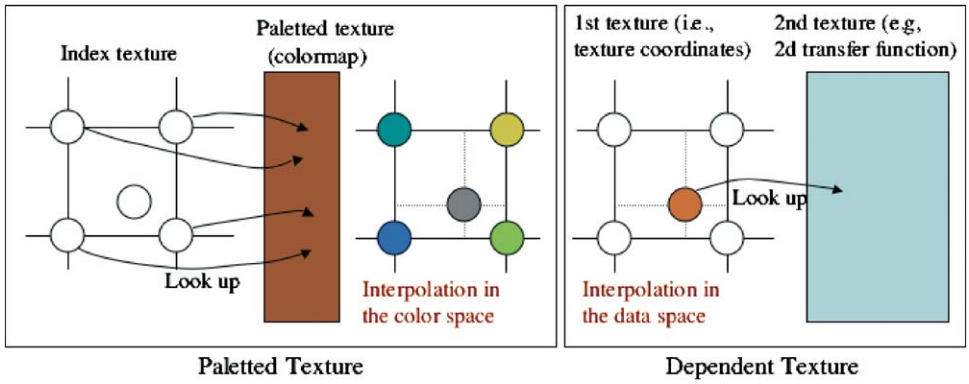

Fig. 2. Paletted texture versus dependent texture, as used for storing transfer functions.

opacity and color maps. In order to reduce the integration errors from the discrete slice sampling of a volume, Engel et al. [6] introduce a method that applies a 1-D transfer function classification using 2-D- rather than 1-D dependent textures. Their pre-integrated volume rendering technique reduces color and opacity integration errors by using two texture samples, one from each of a pair of neighboring slices, and a 2-D lookup table texture that maps the integration of a segment with those scalar values into a pre-computed color and opacity. Kniss et al. [7] describe an interface for specifying 3-D transfer functions, and how they can be applied in hardware using dependent textures. By storing each data value used for classification as a coordinate in a 3-D dependent texture, arbitrarily mapping between those factors and a color and opacity pair can be achieved. They also describe how 3-D transfer functions can be implemented as separable 2-D and 1-D transfer functions to reduce the size of the transfer function texture, with the trade-off of less control in the types of transfer function mappings available.

Tzeng et al. [8] present a new method for specifying higher-dimensional classification functions through an interactive paint-based interface. The user simply paints on a few slices from the volume to give hints about how the classification should be done. Abstracted from the user is the generation of a higher-dimensional classification function using artificial neural networks which are implemented using pixel shaders. The network uses the painted regions as training data to 'learn' a classification function to map voxels into uncertainty over whether the given voxel is part of the material of interest which can then be used as opacity. The classification function uses as inputs a voxel's scalar value, the values of its six neighbors, and its x, y, z location. The scalar value is fundamental information directly from that voxel, its neighboring values provide information that can be incorporated for gradient and texture, while position can be used to take into account a materials structural properties. Two materials might have similar scalar values, but they are much less likely to also have similar texture and location. Thus, by using higher-dimensional classification functions, it is possible to better differentiate the materials for visualization. Fig. 3 shows an example, which would not be made possible with 1-D or 2-D transfer function.

Fig. 3. Using a 10-D classification function it is possible to straightforwardly derive a visualization showing the brain and a part of the head with the skin, skull, and other soft tissues removed. The Image was generated by Fan-Yin Tzeng.

4. Lighting

Lighting can greatly increase the visual quality of volume rendered images by providing subtle depth cues and feature highlighting. To include lighting, however, normalized gradient direction of each voxel must be either pre-computed and stored or calculated on the fly.

Rather than saving the normal vector for every voxel which would significantly increase the storage requirements, we can use paletted textures and store for each texel an index into a normal lookup table, with each direction quantized to one of 240 vectors obtained from the faces of a subdivided version of a combined dodecahedron and icosahedron [9]. This approach only requires one byte per voxel to store normal directions.

If there is insufficient space to store precomputed normals, we can compute the normal vectors on the fly. To calculate the normal information on the fly, the newer hardware permits enough texture lookups in a single pass to derive gradient information based on neighboring texels. By sampling the six nearest neighbors, we can calculate the gradient of that voxel and apply lighting calculations accordingly. This yields high-precision normals without the extra storage and loss of precision due to the use of normal maps. Rendering performance can become bandwidth limited with respect to the graphics bus. Thus, when rendering a large volume dataset that does not fit into video memory, the use of dynamically calculated normals can improve performance. The cost of sending normal maps across the graphics bus for each frame is eliminated entirely and is replaced with a more expensive rasterization stage. If both the volume data and the normal maps could fit entirely in the video memory, then the faster rasterization from using pre-computed normals would result in higher performance.

5. Non-photorealistic rendering

NPR can be used to illustrate subtle spatial relationships that might not be visible with more realistic rendering techniques. NPR for volumetric data visualization has recently become an area of active research. Treavett and Chen [10] show how pen-and-ink rendering can be applied to volume visualization. Ebert and Rheigans [11] describe a number of NPR techniques that can be applied to volume rendering. They show that non-photorealistic methods can enhance features and improve depth perception. When NPR methods are utilized, interactivity becomes even more critical because each NPR technique adds its own set of additional parameters that must be specified. For example, if hue-valued shading is utilized, for best results the variation in color should be carefully specified with respect to both hue, saturation and value until the result desired by the user is met. Much like for the transfer function specification, there is no 'correct' set of parameters that can be utilized for all data sets since these parameters vary widely depending on what type of features the user would like to accentuate or de-emphasize. In fact, often the user does not know what type of rendering style is desired, only through experience and experimentation can parameters be found that are suited for their particular application. In addition, it might be desired to mix photorealistic and NPR styles when rendering a single volume. This would require multiple sets of transfer functions and lighting parameters, multiplying the number of parameters that must be set as well as the need for interactivity. Thus, when NPR is used in a volume rendering context, interactive response is essential with respect to viewpoint, transfer function, and NPR parameter space.

Unfortunately, many factors make interactive NPR difficult. First, the addition of NPR techniques only adds to the number of calculations required in the rendering process. As a simple example, silhouette edge rendering requires the additional calculations associated with silhouette edge detection. Furthermore, the standard technique of rendering lower resolution data or rendering to a lower resolution window to achieve interactivity is often not suited to the needs of the user when selecting NPR parameters. NPR can be used effectively to clarify fine structures in a volume. In order to specify rendering parameters optimized for viewing these structures, the volume must be rendered at high resolutions. For example if a user is trying to accentuate blood vessels in a data set by manipulating the parameters associated with silhouette and gradient, it is necessary that the volume be rendered at a sufficiently high resolution to a screen resolution high enough to view the vessels clearly.

These two requirements, interactivity and high resolution, can be met by using hardware accelerated rendering techniques and a PC cluster, respectively. Lum and Ma [12] show how to employ a number of features found in modern consumer graphics cards, including 3-D textures, multi-texturing, and paletted textures, to implement several NPR techniques in hardware such as hue-varied shading, silhouette illustration and depth-based color cues. By using multiple graphics cards spread across a PC cluster, they are able to render high-resolution volumes at frame rates interactive enough for the tuning of view, transfer function, and NPR parameters. The interactivity they achieve makes possible the creation of highly effective non-photorealistic visualizations which would not be possible with less interactive methods. Fig. 4 displays an NPR example in contrast to regular volume rendering.

In order to combine the different NPR styles in hardware, they make use of multiple rendering passes with variations in texture palette to implement the different techniques. Specifically, the original scalar data are stored in one paletted texture, while the quantized normals are stored in a second paletted texture. The first pass renders the volume with hue-varied shading which is accomplished by rendering polygons with the modulation of two textures. One is the scalar data texture which has a palette containing the transfer function,

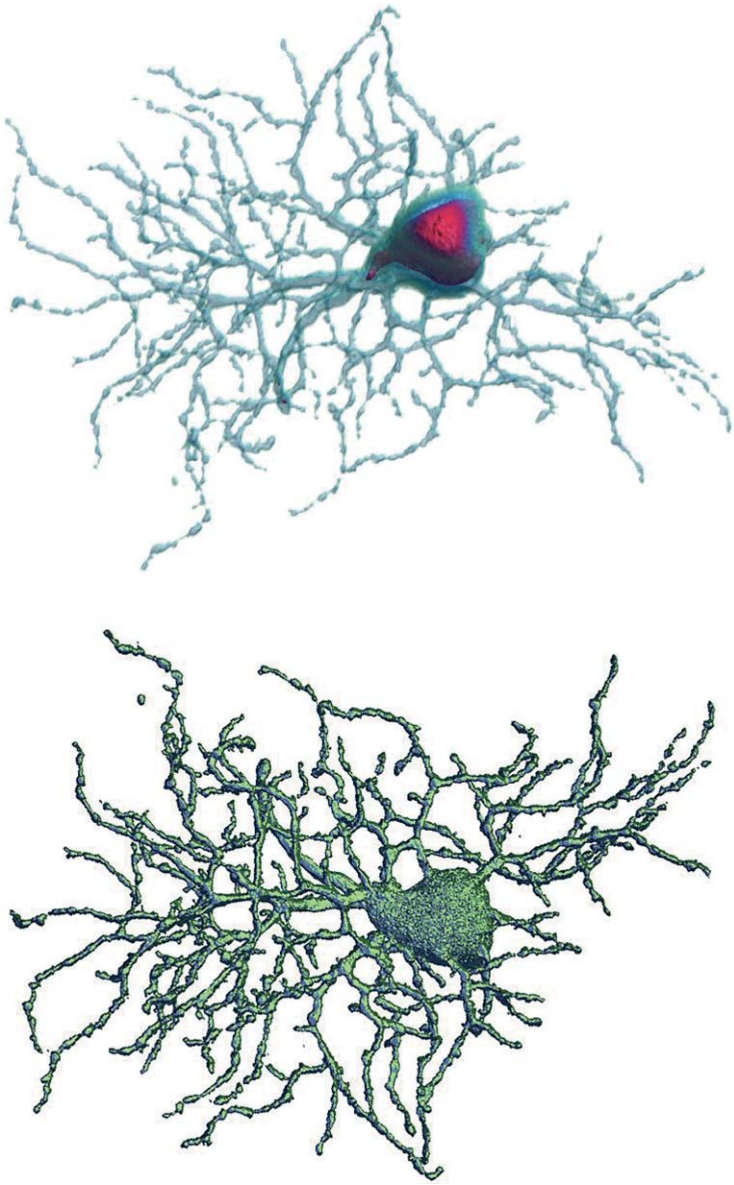

Fig. 4. Top: Direct volume rendering of confocal microscopic ganglion data. Bottom: NPR of the same volume data shows the ganglion structure more clearly.

while the other texture stores paletted normals, with a palette set such that the color entry for each indexed normal is set to the hue and intensity calculated for that normal direction. The second rendering pass applies specular highlights and silhouette edges. During this pass, opacity from the transfer function is utilized in a scalar data texture that is modulated with silhouette and specular contributions stored in a paletted normal texture. The palette for this second texture is set white and opaque for normal directions that yield specular highlights, black and opaque for normal directions that are perpendicular to the view direction and correspond to silhouette edges, and transparent for all other directions. Additional non-photorealistic techniques can be applied by modulating each pass with more textures, such as a 1-D hue-varied texture that is varied along the view direction to show aerial perspective (variation in color based on depth), or a gradient texture with opacity varied based on gradient value to better illustrate surfaces.

Stompel et al. [13] apply hardware-accelerated NPR to the visualization of multi-valued volume data and

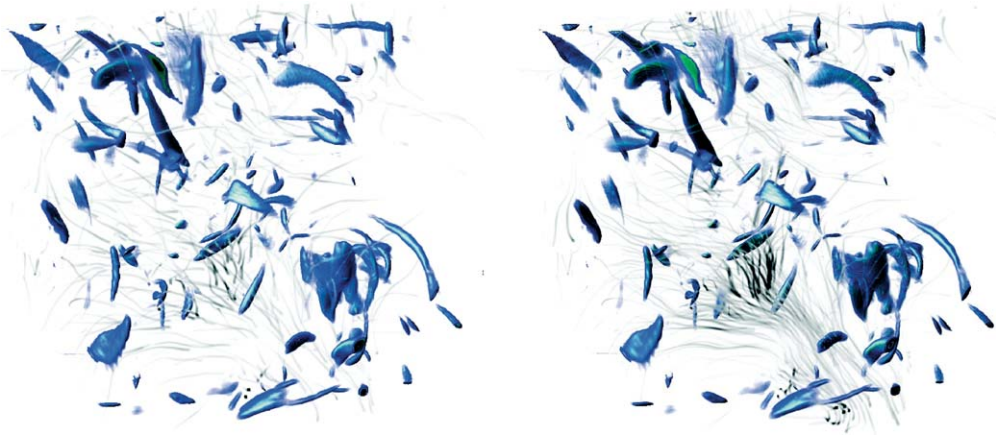

Fig. 5. Hardware-accelerated volume rendering of vorticity field and stroke-based rendering of velocity field. Stroke size and color can be changed interactively. Images were generated by Aleksander Stompel.

time-varying volume data. One or more variables are used to either highlight important features in another variable, or add contextual information to the visualization using different NPR styles. For time-varying data, rendering of each time step also takes into account the values at neighboring time steps to reinforce the perception of the changing features in the data over time. Similarly, hardware-accelerated rendering enables interactive tuning and selection of a set of rendering parameters and methods to derive effective visualizations. Fig. 5 shows rendering of stroke volumetric textures for the visualization of turbulent flow.

6. Rendering isosurfaces from 3-D texture

Isosurface visualization is widely used in many engineering and medical applications. Isosurfaces are mostly extracted in a view-independent manner and represented as triangular meshes that can be efficiently rendered with polygon graphics hardware. The amount of time required for isosurface extraction is heavily dependent on which isovalue is used, where some isovalues can result in extremely dense geometry and thus high polygon counts, hampering interactive rendering. Using texture hardware, it is possible to approximate the appearance of an isosurface in a view-dependent way. The rendering cost, however, is independent of the isovalue selected.

Engel et al. [6] extend the pre-integration method to render an isosurface using view aligned polygon slices that transverse a 3-D texture and achieve a 1 frame/s rendering rate for $256 \times 256 \times 256$ volume data using an nVidia GeForce3. Lum and Ma [14] also render isosurfaces from 3-D texture with lighting using normals from gradient directions that are calculated dynamically at the location of ray-isosurface intersections estimated with sub-voxel precision using a pixel shader. In order to reduce the number of pixels that must be rasterized using the computationally expensive gradient calculating pixel shader, they employ a two-pass rendering method where the first pass sets the z-buffer for those pixels where an isosurface intersection occurs, and the second pass renders the lit isosurface where there is z-buffer equality. Using an ATI Radeon 9700 Pro card, they can achieve 1.6 frames/s for $512 \times 512 \times 512$ volume data. Fig. 6 shows an isosurface visualization example using their technique.

7. Large volume data

The size of the volume that can be rendered interactively is limited by the amount of video memory the graphics card contains. The sheer size of a data set from a contemporary scientific application can easily overwhelm a commodity graphics card which typically has up to 256 MB. For data too large to completely fit in the video memory, the rendering performance is thus limited by how fast data can be transferred from the main memory to the video memory.

7.1. Rendering a single large-volume data set

The female anatomical images set of the visible human data set is about 39 GB, and the male one is over 60 GB. Such sizes overwhelm even the largest main memory possible for a single PC. Guthe et al. [15] develop a compressed hierarchical wavelet representation of the volume data that can be efficiently decompressed on-the-fly in software and transferred to the video memory, and rendered using hardware texture mapping. Level-of-detail rendering is done based on both the view and an error metric. According

Fig. 6. Hardware-assisted isosurface rendering of the CT Christmas tree data.

Fig. 7. Comparing image quality for interactive rendering of the visible human male data. Left: low quality, 7.1 fps. Middle: medium quality, 5.4 fps. Right: high quality, 2.8 fps. Images are provided by Stefan Guthe at WSI/GRIS, University of Tübingen.

to their experimental results, they can use a data compression ratio of 30:1 without introducing visually noticeable artifacts. With run-length Huffman coding, decompression is about 50 MB/s. Further performance improvement is achieved by caching the decompressed data for subsequent frames. They are thus able to achieve interactive rendering of the visible female data on a single 2 GHz Pentium 4 PC with an nVidia GeForce 4. For data larger than the main memory can hold, their technique can be generalized for out-of-core rendering. Fig. 7 compares the quality of images rendered at different level of details.

7.2. Rendering time-varying data

Many applications produce large-scale time-varying volume data sets. Scientists' ability to animate time-varying phenomena is absolutely essential to ensure correct interpretation and analysis of the data, to provoke insights and to communicate their findings with others. Hardware-accelerated volume rendering, while offering realtime rendering rates, requires the loading of the volume data into the texture memory of the video card prior to rendering. Even though a single time step of the data might fit in the video memory, the complete time-varying volume data set may consist of hundreds to thousands of time steps which would not fit in the video memory. Rendering performance is thus determined by how fast the time steps can be transferred into the video memory. One solution is to treat video memory, main memory, and disk as a three-level cache for volume rendering. By compressing the volume data we increase the amount of data that can fit in each level while decreasing the I/O costs of transferring data between these levels. In this way, the interactive volume rendering of very large data sets is possible using commodity PC hardware.

Lum et al. [16] develop a solution based on the temporal encoding of indexed volumetric data that can quickly be decoded in hardware. The method makes extensive use of support for the changing of color palettes without reloading volume textures. The cycling of color palettes can be used to create simple animations from static images. Similarly, the use of color palette manipulation allows a single scalar value to represent multiple time steps values. Even though this results in lossy compression, according to their test results, in most cases the differences between the compressed and uncompressed visualizations are hard to discern.

This approach actually compresses the index data, rather than color data, to avoid recompressing the volume when modifying the color or opacity transfer functions. The encoding process consists of mapping sequences of scalars into a single scalar index. The discrete cosine transform (DCT) is used for this process which by itself is reversible and does not compress the data. Rather, this transform is selected since it puts more energy into fewer coefficients, thus allowing the less important, lower energy, coefficients to be quantized more coarsely, thus using less data.

When the encoding is complete, the sequence of scalar values is calculated for each possible index. Rather than using the inverse DCT, a mapping for each index for each time frame is determined that minimizes the mean square error of the decoded value. The decoding is done during rendering and uses an extension of OpenGL to allow the palette of all textures to be changed at once. Fig. 8 shows the encoding process.

Using four times compression, on an AMD 1.2 GHz Athlon with 768 Mbytes of main memory and a GeForce 3 with 64 Mbytes of texture memory, 1492 time steps of a $256 \times 256 \times 256$ volumetric data set can be rendered in an out-of-core fashion at approximately 6.8 frames/s using 256 object aligned textured polygons. Since the main memory can hold 280 time steps of the data, if rendered in-core, 25.8 frames/s can be achieved. Without compression, the same 280 time steps no longer fit in main memory and would need to be swapped into main memory in an out-of-core manner. A memory-resident subset of this uncompressed data can be rendered at about 11.5 frames/s, compared to the 25.8 frames/s with compression. These results were obtained when rendering the volume to a 512×512 window, with the volume occupying more than half the window area.

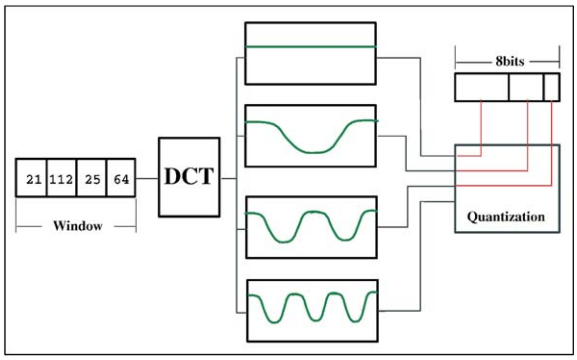

Fig. 8. DCT-based encoding. In this example, the window size is 4 and only the first three coefficients are stored into an 8-bit value.

Note that there is a distinct tradeoff between the compression ratio and rendering performance versus the quality of the compressed volume. This gives users a degree of flexibility in choosing the compression ratios that best meet their needs. For example, if a scientist is interested in viewing a short time sequence at high quality, since it lower compression ratio can be used. On the other hand, to view a very long sequence of data at high speeds, a higher compression rate can be selected. The scientist can combine compression ratios to preview a data set at a coarser temporal resolution and then view a specific time sequence of interest with less compression.

8. Volume rendering clusters and image compositing hardware

Parallel volume rendering offers a feasible solution to the large data visualization problem by distributing both the data and rendering calculations among multiple computers connected by a network. In sort-last parallel volume rendering, each processor creates an image of its assigned subvolume, which is blended together with other images to derive the final image. Improving the efficiency of this compositing step, which requires interprocessor communication, is the key to scalable, interactive rendering. Parallel software compositing algorithms generally are sufficiently efficient in previous parallel rendering systems where the rendering calculations dominate the cost. The increased use of hardware-accelerated volume rendering demands further acceleration of the image compositing step. An optimized algorithm introduced recently can support rendering rates at 5–20 frames/s [17]. However, to achieve real-time rendering rates, over 30 frames/s, we must seek hardware solutions.

Several specialized hardware architectures and devices have been developed to support real-time image compositing for demanding graphics applications using a cluster of graphics-enhanced PCs. Sepia [18], Lightning-2 [19] and Metabuffer [20] were developed for the construction of large display subsystems for distributed clusters. Sepia is a commodity-based architecture implemented by custom PCI cards connected to a high-speed network for image acquisition, compositing,

Fig. 9. Image compositing boards fabricated by Mitsubishi Precision Co., Ltd.

and display. It supports pipelined associative blending operations in a sort-last configuration. The second generation of Sepia [21] incorporates a high-speed network interface. Lightning-2 is a hardware system that employs scanline-based pixel mapping and provides a DVI-to-DVI interface which delivers pixel data from graphics accelerators to remote tiled displays. It scales in both the number of rendering nodes and the number of displays supported, and allows any pixel data generated from any node to be dynamically mapped to any location on any display. However, using DVI limits the compositing to RGB data so semitransparent volume rendering cannot be efficiently supported. Metabuffer based on a mesh interconnect is similar to Lightning-2 in supporting a rich set of viewport mappings but it also offers multi-resolution support.

Muraki et al. [21] describe a compositing hardware system designed specifically for the construction of volume graphics clusters. Their approach, based on binary-tree compositing, results in reduced circuitry, scalable performance and $\log(n)$ latency. Their compositing hardware system consists of PCI interface boards and an eight-way compositing device. The compositing hardware has a 21-stage pipeline with 36-bit bandwidth. Most of the hardware on the interface boards and the compositing device is implemented using FPGAs. Fig. 9 displays the compositing boards.

The advantage of this system is the scalability that comes from the octree connectivity of the compositing devices. A prototype 17-node cluster system connecting two 8-node clusters with consumer graphics cards (GeForce 4 Ti4600), is capable of delivering 115, 45 and 22 fps rendering rates, respectively, for 256×256, 512×512 and 768×768 image sizes.

While some of these hardware solutions have become commercially available, it is still prohibitively expensive to build a large system using these specialized compositing devices.

9. Conclusion

We have presented a variety of ways to exploit the high precision, programmable, texture mapping features of consumer graphics hardware for advancing the state of the art in volume data visualization. At the time of this writing, the representative consumer graphics cards on the market are nVidia GeForce FX and ATI Radeon 9800 Pro. New graphics cards which offer more video memory and programmable features are made available every year. It is thus difficult to exploit the full potential power of these graphics cards before they get replaced. Computer graphics researchers were never before given so much power and freedom in using hardware for creative design of visualization techniques. In this article, we limit our discussion to rectilinear-grid volume data. However, it is clear that the advanced features of the newer cards can be exploited for interactive visualization of unstructured volume data and vector data.

Acknowledgements

This work has been sponsored in part by the US National Science Foundation under Contracts ACI 9983641 (PECASE award) and ACI 9982251 (the LSSDSV program); the US Department of Energy under Memorandum Agreements No. DE-FC02-01ER41202 (SciDAC program) and No. B523578; the National Institute of Health through the Human Brain Project; and a United States Department of Education Government Assistance in Areas of National Need (DOE-GAANN) Grant P200A980307. The confocal microscopic data set was provided by Dr. Leo Chalupa at the Division of Biological Sciences of the University of California at Davis. The turbulent flow data set was provided by Dr. GuoWei He at ICASE/NASA LaRC. The CT-dataset XMasTree was generated from a real

world Christmas Tree by the Department of Radiology, University of Vienna and the Institute of Computer Graphics and Algorithms, Vienna University of Technology. The authors would especially like to thank the editors, Gunter Knittel and Bengt-Olaf Schneider, for the suggestions that they provided to improve the manuscript.

References

[1] Cabral B, Cam N, Foran J. Accelerated volume rendering and tomographic reconstruction using texture mapping hardware. In: Proceedings of the 1994 Symposium on Volume Visualization; 1994. p. 91–8, Washington DC.

[2] Pfister H, Hardenbergh J, Knittel J, Lauer H, Seiler L. The VolumePro real-time ray-casting system. In: Proceedings of SIGGRAPH '99 Conference; 1999. p. 251–60, Los Angeles, CA.

[3] Meißner M, Kanus U, Wetekam G, Hirche J, Ehlert A, Straßer W, Doggett M, Forthmann P, Proksa R. VIZARD II: a reconfigurable interactive volume rendering system. In: Proceedings of the Eurographics Workshop on Graphics Hardware; 2002, Saarbucken, Germany.

[4] Hadwiger M, Theußl T, Hauser H, Gröller E. Hardware-accelerated high-quality filtering on PC hardware. In: Proceedings of the Vision, Modeling, and Visualization; 2001. p. 105–12, Stuttgart, Germany.

[5] Pfister H, Lorensen B, Bajaj C, Kindlmann G, Schroeder W, Sobierajski Avila L, Martin K, Machiraju R, Lee J. The transfer function bake-off. IEEE Computer Graphics and Applications 2001;21(3):16–22.

[6] Engel K, Kraus M, Ertl T. High-quality pre-integrated volume rendering using hardware-accelerated pixel shading. In: Proceedings of the Graphics Hardware 2001; 2001, Los Angeles, CA.

[7] Kniss J, Kindlmann G, Hansen C. Multidimensional transfer functions for interactive volume rendering. Transactions on Visualization and Computer Graphics 2002;8(3):270–85.

[8] Tzegn F-Y, Lum E, Ma K-L. A novel interface for higher-dimensional classification of volume data. In: Proceedings of IEEE Visualization 2003 Conference; 2003. (To appear) Seattle, WA, October, 19–24.

[9] Van Gelder A, Hoffman U. Direct volume rendering with shading via three-dimension textures. In: ACM Symposium on Volume Visualization '96 Conference Proceedings; 1996, San Francisco, CA.

[10] Treavett S, Chen M. Pen-and-ink rendering in volume visualisation. In: Proceedings of IEEE Visualization 2000 Conference; 2000. p. 203–9, Salt Lake City, UT.

[11] Ebert D, Rheingans P. Volume illustration: non-photorealistic rendering of volume models. In: Proceedings of IEEE Visualization 2000 Conference; 2000. p. 195–202, Salt Lake City, UT.

[12] Lum EB, Ma K-L. Hardware-accelerated parallel non-photorealistic volume rendering. In: Proceedings of the International Symposium on Nonphotorealistic Animation and Rendering; 2002. p. 67–74, Annecy, France.

[13] Stompel A, Lum E, Ma K-L. Visualization of multidimensional, multivariate volume data using hardware-accelerated non-photorealistic rendering techniques. In: Proceedings of Pacific Graphics 2002 Conference; 2002. p. 394–402, Beijing, China.

[14] Lum E, Ma K-L, Rendering isosurfaces directly from 3-d textures. Technical Report CSE-2003-10, Department of Computer Science, University of California at Davis, April 2003.

[15] Guthe S, Wand M, Gonser J, W. Straßer, Interactive rendering of large volume data sets. In: Proceedings of Visualization 2002 Conference; 2002. p. 53–60, Boston, MA.

[16] Lum EB, Ma K-L. Non-photorealistic rendering using watercolor inspired textures and illumination. In: Pacific Graphics '01 Conference Proceedings; 2001, Tokyo, Japan.

[17] Stompel A, Ma K-L, Lum E, Ahrens J, Patchett J. SLIC: scheduled linear image compositing for parallel volume rendering. Technical Report CSE-2003-8, Department of Computer Science, University of California at Davis, March 2003.

[18] Moll L, Heirich A, Shand M. Sepia: Scalable 3-d compositing using PCI pamette. In: Proceedings of IEEE Symposium on FPGAs for Custom Computing Machines; 1999. p. 146–55, Napa, CA.

[19] Gordon S, Eldridge M, Patterson D, Webb A, Berman S, Levy C, Caỹwood R, Taverira M, Hunt S, Hanrahan P. A high performance display subsystem for PC clusters. In: Proceedings of SIGGRAPH 2001 Conference; 2001. p. 141–8, Los Angeles, CA.

[20] Blanke WJ, Fussell DS, Bajaj C, Zhang X. The metabuffer: a scalable multiresolution multidisplay 3-d graphics system using commodity rendering engines. Technical Report No. 2000-16, University of Texas at Austin, 2000.

[21] Lombeyda S, Moll L, Shand M, Breen D, Heirich A. Scalable interactive volume rendering using off-the-shelf components. In: Proceedings of the 2001 Symposium on Parallel and Large-Data Visualization and Graphics; 2001. p. 115–21, San Diego, CA.