Intuitive and Interactive Manipulation of 3D Datasets by Integrating Texture Mapping Based Volume Rendering into the OpenInventor Class Hierarchy

Peter Hastreiter, Hüseyin Kemâl Çakmak, Thomas Ertl

Lehrstuhl für Graphische Datenverarbeitung (IMMD 9), Universität Erlangen
Am Weichselgarten 9, 91058 Erlangen, Germany
hastreiter@informatik.uni-erlangen.de

Abstract: *For the visualization of 3D datasets from medical imaging modalities (MR, CT, PET) a volume rendering technique based on the texture mapping hardware of high-end workstations, as suggested by Cabral et al. [1], was implemented and integrated into the OpenInventor class hierarchy which allows for interactive and intuitive rendering of 3D medical datasets. By integrating a color editor for the direct manipulation of separate transfer functions for the RGB-Alpha channels and by integrating a method for the rendering of tagged volumes color and transparency of subvolumes with different tags can be manipulated separately and at interactive rates. In order to appropriately view the volume from inside several clipping planes, a method for changing the opacity of entire planes parallel to the viewport and a method for clipping a cube can be employed.*

Keywords: Volume Rendering, Texture Mapping, OpenInventor

1 Introduction

An important prerequisite for the visualization of medical image data from 3D modalities (MR, CT, PET) in practice is the ease of interpretation. Traditional inspection of 2D slice images can be effectively assisted by automatic segmentation procedures and projection images of the whole dataset gained by volume rendering techniques. In practice, however, application of volume rendering was hindered by time-intensive computations restricting the free manipulation of visualization parameters and viewing directions. Although image generation was accelerated up to few seconds by optimizing existing algorithms and by introducing new algorithmic approaches [2, 3, 4] interactive visualization was still not possible even if there was extensive preprocessing. Therefore, realtime volume visualization is currently only feasible on parallel systems [5, 6] or on special purpose hardware [1, 7].

In 1994 Cabral et al. [1] introduced a volume rendering technique which takes advantage of the hardware interpolation and compositing capabilities of the texture mapping subsystem of high-end graphics workstations, which allows for realtime volume rendering of reasonably sized data sets. With the recent introduction of a new generation of graphics hardware accelerated texture mapping became available on the desktop making this volume rendering technique accessible for a wider class of users. Demonstration implementations based on OpenGL and OSF/Motif exist, but show a rather limited functionality and non-standard ways of user interaction.

OpenInventor, an object oriented toolkit built onto OpenGL has become a defacto standard for the interactive modeling, rendering and manipulation of 3D surface representations. The goal of the work presented here is to integrate texture-mapping based volume rendering into the OpenInventor framework. By introducing a new class for OpenInventor the volume renderer is represented as a separate object within the hierarchical structure of the scenegraph. This allows easy application of built-in manipulators, sensors, editors and other predefined classes, methods and features (light sources, antialiasing, stereo mode, perspective/parallel rendering, fly, walk, trackball) (see Fig.2 (left side)).

After a short survey of volume rendering theory in general and the technique of volume rendering based on 3D texture mapping hardware as presented in chapter 2 we describe OpenInventor related details of our implemention in chapter 3. In addition to high speed volume rendering with various OpenInventor manipulators being applicable to moving the rendered volume in 3D space two methods were implemented for interactively viewing the volume dataset from inside. As presented in chapter 3.1 the user may employ a number of clipping planes which may also be positioned with one of the OpenInventor specific manipulators or a special box clipping functionality. Furtheron, in chapter 3.2 a color editor is described which allows for the interactive manipulation of separate transfer functions for the RGB-Alpha channels and for the direct manipulation of the opacity for whole planes parallel to the viewport. Finally, the capability of visualizing tagged volume datasets and the technique of bricking in case of volume datasets bigger than the available 3D texture memory will be explained in chapter 3.3 and chapter 3.4 respectively.

2 Volume rendering based on texture mapping

The basic equation of volume visualization is the general form of the equation of light transfer which is derived from the conservation law of energy [8]. It describes the change of specific intensity due to absorption, emission and scattering. Neglecting scattering this leads to the *emission-absoption model* with a simplified equation of transfer which in integral form is

$$I(s) = I(s_0)e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(s')e^{-\tau(s',s)}ds' \tag{1}$$

with

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)ds \tag{2}$$

describing the optical depth and $\kappa(s)$ being the absorption coefficient. If absorption and emission are constant within volume elements which is the case for medical volume data the *compositing* equation

$$C_{out} = C_{in}(1-\alpha) + C\alpha, \tag{3}$$

is obtained with α being the opacity of a volume element. Illustrated in Fig.1 (right side) the emitted intensity C_{out} of a volume element then is the sum of its own intensity weighted with its opacity and the intensity C_{in} entering the volume element weighted with the element's transparency $1-\alpha$. If a fast 3D texture mapping subsystem is available a volume dataset can be loaded to the 3D texture memory. Due to the system's realtime interpolation capabilities equidistant planes can then be 'cut' out of the 3D texture at high speed with the obtained 2D textures being mapped onto polygons parallel to the

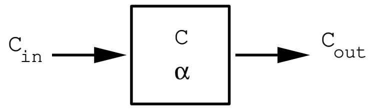

Figure 1: Illustration of the compositing equation

viewport. After mapping the data values onto color and opacity values through a lookup table compositing of output images is achieved in a back-to-front order by successive blending according to Eqn.3.

3 Implementation

The implementation was performed on a Onyx RealityEngine2 (RE2) of Silicon Graphics with two raster managers which allows for using volume datasets as 3D textures and which provides fast interpolation, mip-mapping, antialiasing and blending capabilities which is the basis for the above mentioned volume rendering approach. All programming was done with OpenInventor an object oriented graphics toolkit built onto OpenGL which provides the user with interactive 3D graphics applications. The rendering is performed with OpenGL and its extensions with the definition and management of all objects being completely done by OpenInventor.

The objects containing various information for the generation of a 3D scene are represented by nodes within the scenegraph of OpenInventor which has a tree like structure. There are *shape nodes* for geometric objects, *property nodes* defining the characteristics of an object (color, texture, ...), *group nodes* like SoSeparator combining several nodes, *sensor nodes* identifying any change within the 3D scene, etc.. Furtheron, for the manipulation of objects within the 3D scene there are several *manipulators* which allow for rotating, relocating and scaling parts of the scene. Additionally, there are various *material editors* which allow for changing the material properties of the objects (color, transparency). Finally, the visualization of the objects may be changed interactively with the user having the choice to render with different modes (e.g: antialiasing, stereo, ...). For a complete reference see [9]. With a new class *Volume* being defined including methods 'cutting' planes out of a 3D texture and methods generating the polygons for rendering this class must be integrated into the scenegraph (see Fig.2 (right side)). For this purpose the SoSeparator *volumeSeparator* is defined which contains all nodes necessary for the rendering process: *SoLightModel* containing all lighting paramters, *SoResetTransform* reseting the transformation paramters, *SoMaterial* containing global color and opacity values, *SoTransform* keeping all transformation paramters of various manipulators which finally affect the node *Volume*.

3.1 Clipping Planes and Box Clipping

In order to view structures of the volume from inside the user may employ a number of clipping planes or a special box clipping technique (see Fig.3 (right side)). Each clipping plane can be addressed separately and can be manipulated interactively with the posi-

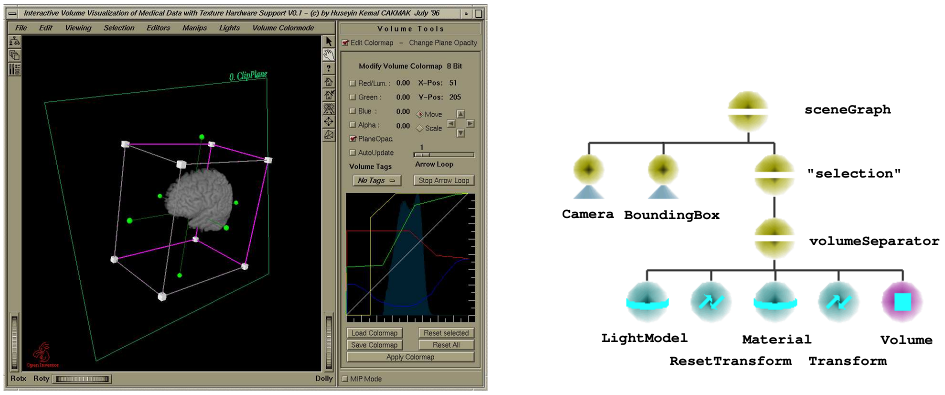

Figure 2: Volume rendering with OpenInventor (left side); scenegraph after adding the separator *volumeSeparator* (right side)

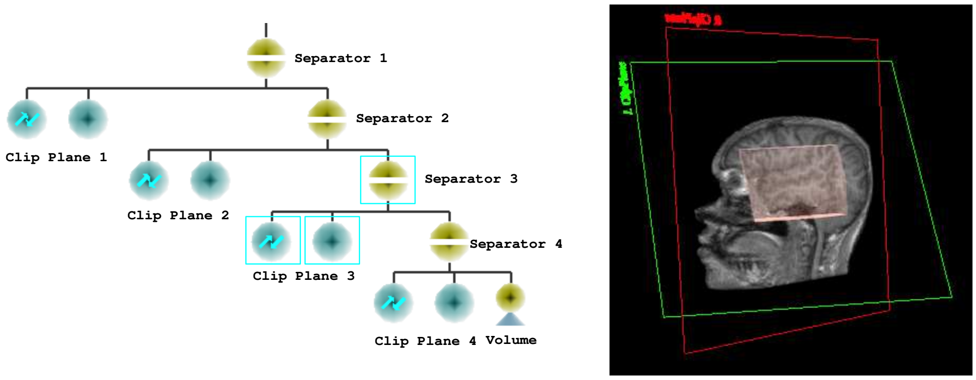

Figure 3: Hierarchy of clipping planes (left side); applying two clipping planes and box clipping (right side)

tioning performed with one of the OpenInventor specific manipulators. Since only those objects are affected which are positioned behind a clipping plane in the scenegraph there must be a hierarchy for all clipping planes terminating with the node *Volume* which contains the actual data (see Fig.3 (left side)). Using the box clipping technique the origin and the extent of the cube must be specified in order to allow manipulation of color or opacity within the interior or exterior part of the cube. Using the material editor the user may then change color or opacity of the respective part.

3.2 Color and Planeopacity Editor

As another feature of the volume renderer we added a separate color editor which allows for interactive manipulating of the currently applied lookup table. Separate transfer functions for the RGB-Alpha channels can be adjusted graphically by freehand drawing or by indicating start and end points of segments for linear mappings.

Furtheron, the plane opacity editor allows for manipulating the opacity of whole planes which are parallel to the viewport. These planes are identical to those which are generated during the rendering process of every output image. Since this method always adresses planes parallel to the viewport the clipping area changes if the rendered object is rotated.

3.3 Tagged Volumes

Having segmented different structures of a volume dataset in a preprocessing step and having assigned a tag to every voxel according to the segmentation the thus created tagged volume can be visualized with our implementation offering the additional ability to manipulate color and transparency of subvolumes with different tags separately. Since one lookup table can be used every original data value $origValue$ must be modified according to

$$newValue[i] = origValue[i] + (tag[i] \cdot 4 \cdot 256) \tag{4}$$

so that different color and opacity values are assigned to subvolumes of different tags (RGB-Alpha mode is used!). Since the *glColorTableSGI* routine which is used for interactive manipulation of color and opacity values only accepts lookup tables of 256 different colors the feature of interactive manipulation is so far only applicable to non-tagged volume data of 8 bit.

3.4 Further Features of the Volume Renderer

If the amount of input data exceeds the available texture memory the volume has to be divided into several appropriately sized subvolumes, called bricks. The bricks treated as independent textures are then successively loaded into the texture memory resulting in a final output image of the whole volume data.

Since volume datasets require to render a certain amount of planes with a specific in plane resolution in order not to suppress any information there is a limit of interactivity due to a restricted number of trilinear interpolations per second (trips) depending on the hardware. With a typical medical dataset consisting of 30 to 200 slices ($256 * 256$ or $512 * 512$ pixels per slice) there are 2 to 50 Mvoxels requiring 32 to 200 Mtrips, according to [7], in order to suppress aliasing problems. In comparison to that there are 40 Mtrips provided by every raster manager of a RE2 with 160 Mtrips at maximum. In order to overcome this problem we decided only to take a reduced number of planes for rendering while the volume is moved. After remaining at a position for a user defined amount of time the rendered image is automatically switched to full resolution.

Finally, the application allows interactive switching between the blending modes RGB-Alpha and MIP (maximum intensity projection) the later used for the visualization of vascular information. Furtheron, since implemented within the OpenInventor framework the user may simultaneously render volume and polygonal data.

4 Conclusion and Future Work

Since adequate speed is essential for the application of volume rendering the interactive capabilities of 3D texture mapping seem to be superior to other approaches. Almost naturally, this involves the ease of interpretation which is claimed for clinical usage. Additionally, the intuitive capabilities are enhanced by integrating the volume renderer into the OpenInventor framework.

In the near future we will improve the rendering tool by adding more features of manipulating the volume and by allowing multiple volumes to be rendered. Furtheron, we will investigate other clinically relevant extensions including hardware assisted segmentation and automatic navigation procedures.

References

[1] B. Cabral, N. Cam, and J. Foran. Accelerated Volume Rendering and Tomographic Reconstruction Using Texture Mapping Hardware. *Symposium on Volume Visualization, Washington DC*, pages 91–98, 1994.

[2] L. Westover. Footprint Evaluation for Volume Rendering. *Computer Graphics*, 24(4):367–376, August 1990.

[3] D. Laur and P. Hanrahan. Hierarchical Splatting: A Progressive Refinement Algorithm for Volume Rendering. *Computer Graphics*, 25(4):285–288, July 1991.

[4] T. Malzbender. Fourier-Volume-Rendering. *ACM Transactions on Graphics*, 12(3):233–250, July 1993.

[5] P. Lacroute. Real-Time Volume Rendering on Shared Memory Multiprocessors Using the Shear-Warp Factorization. In *Parallel Rendering Symposium*, pages 15–22, Atlanta GA USA, 1995. ACM.

[6] R. Grosso, Th. Ertl, and R. Klier. A Load Balancing Scheme for Parallelizing Hierarchical Splatting on a MPP System with Non-uniform Memory Access Architecture. In M. Chen, P. Townsend, and J. A. Vince, editors, *Proc. of the Int. Sym. on High Perf. Comp. for Comp. Graphics and Vis., Swansea*, pages 125–134. Springer, 3-5 July 1995.

[7] K.J. Zuiderveld, P.M. van Ooijen, J.W. Chin-A-Woeng, P.C. Buijs, M. Olree, and F.H. Post. Clinical Evaluation of Interactive Volume Visualization. In *Proceedings of Visualization'96*, 1996.

[8] H.C. Hege, T. Höllerer, and D. Stalling. Volume Rendering, Mathematical Foundations and Algorithmic Aspects. Technical Report TR93-7, Konrad-Zuse-Zentrum für Informationstechnik, Berlin, 1993.

[9] J. Werneke. *The Inventor Mentor, Programming Object-Oriented 3D Graphics with OpenInventor*. Addison-Wesley, release 2 edition, 1994.