Title
Bridging the resolution gap: superimposition of multiple multi-channel volumes

Permalink
https://escholarship.org/uc/item/0491p491

Author
Liang, Chih K.

Publication Date
2008

Peer reviewed|Thesis/dissertation
Bridging the Resolution Gap:  
Superimposition of Multiple Multi-Channel Volumes

A thesis submitted in partial satisfaction of the requirements for the degree
Master of Science

in

Computer Science

by

Chih K. Liang

Committee in charge:

Matthias Zwicker, Chair
Samuel R. Buss
Henrik Wann Jensen
Jürgen P. Schulze
Ruth West

2008
The thesis of Chih K. Liang is approved:

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

________________________________________________________________________

Chair

University of California, San Diego

2008
# TABLE OF CONTENTS

Table of Contents ................................................................. iv
List of Figures ................................................................. v
List of Tables ................................................................. vi
Acknowledgements ............................................................ vii
Abstract of the Thesis ........................................................ viii
Chapter 1  Introduction .......................................................... 1
Chapter 2  Related Work ........................................................ 4
Chapter 3  Volume Rendering ..................................................... 7
Chapter 4  Implementation ....................................................... 11
   4.1  User Input Variables .................................................... 11
   4.2  Transfer functions ...................................................... 12
       4.2.1  Gamma Function ................................................ 14
       4.2.2  High-Pass Filter ............................................... 15
       4.2.3  Histogram CDF ............................................... 16
   4.3  Data Variables .......................................................... 17
   4.4  Pixel Shader ............................................................ 18
       4.4.1  Shader Program ............................................... 20
   4.5  Sorting Algorithm for Textured Planes ............................. 22
   4.6  Blending Slices ....................................................... 23
       4.6.1  Alpha Blending ............................................... 23
       4.6.2  MIP (Maximum Intensity Projection) ....................... 24
Chapter 5  Results .............................................................. 25
Chapter 6  Conclusions & Future Work ....................................... 33
References ................................................................. 35
LIST OF FIGURES

Figure 3.1: Simplified volume rendering model. (a) At each point $x$ inside the volume, while $\epsilon(x)$ determines the emission of light, $\phi(x)$ is the attenuation factor along the direction. (b) Points inside the volume are discretely sampled at regular spacing along a viewing ray. .................................................. 7

Figure 4.1: Transfer functions. Volume using (a) Gamma function with $\gamma^c = 0.5$ for all channels, $\gamma^o = 2$ for volume opacity, (b) High-Pass filter with $\sigma^c = 0.2, \eta^c = 2, \sigma^o = 0.6, \eta^o = 1$, (c) Histogram CDF with $\gamma^o = 4$. .................... 13

Figure 4.2: Gamma Function ............................................................................. 14
Figure 4.3: High-Pass Filter ............................................................................. 15
Figure 4.4: Each image size is 512x512 with 8-bit intensity values. Each histogram is performed on 262144 points with 256 bins. (a) Histogram and Histogram CDF of the dark image on the left, (b) Histogram of output image on the left after mapping original image with Histogram CDF. ................................. 16

Figure 4.5: Resulting images with the alpha equations and different channel weights. (a) and (b) are obtained by using Equations 4.2 and 4.3, respectively, and by specifying the weight of the green channel twice as much as other channel weights. (c) and (d) are obtained by using Equations 4.2 and 4.3, respectively, and by specifying equal channel weights. ................................................................. 19

Figure 4.6: Interleaving 6 slices ($d_1$ apart) from Volume 1 and 4 slices ($d_2$ apart) from Volume 2. ................................................................. 22

Figure 5.1: Overlapping 2-channel volumes to different degrees: (a) Full, (b) Half, (c) None. ........................................................................................................ 26

Figure 5.2: Multiple multi-channel volumes collected from NCMIR in 1600 x 900 window: (a) Rendering of volume slices according to the sequential order of the volumes. Running at $\sim$14fps; (b) Rendering after interleaving depth-sorted volume slices. Running at $\sim$14fps. .................................................. 27

Figure 5.3: (a) LM: $512 \times 246 \times 128$ voxels, (b) EM: $256 \times 512 \times 128$ voxels, (c) LM and EM superimposed with sequential rendering of LM first and EM later, (d) LM and EM superimposed with slice interleaving. ............................................. 28

Figure 5.4: 3-channel volume data sets from structures within a kidney. (a) Low-resolution superset: $512 \times 256 \times 128$ voxels, (b) High-resolution subset: $256 \times 256 \times 128$ voxels, (c) Superimposition of the 2 volumes with sequential rendering of (a) first and (b) later, (d) Superimposition of the 2 volumes with slice interleaving. 29

Figure 5.5: (a) Volumes with Cyan, Yellow, Magenta as channel colors; (b) Volumes with Red, Green, Blue as channel colors. ............................................. 31

Figure 5.6: (a) Maximum Intensity Projection of volumes at 21.7 fps; (b) Alpha Blending of volumes at 22.3 fps. ................................................................. 32
LIST OF TABLES

Table 5.1: Performance results from Figure 5.1, rendered in a window of 1600 × 900 pixels. The volume sizes in number of voxels are 512 × 512 × 52 and 256 × 256 × 104. Numbers of volume slices are 64 and 128 with MIP blending method. Render time includes processing time for OpenGL commands, including glFinish and glDraw. ................................................................. 26

Table 5.2: Performance results for single-channel volumes (dendrites), rendered in a window of 1600 × 900 pixels. The major difference in rendering time between EM and LM is caused by the GPU fill rate. The pixel size of the volume rendered on the screen affects the rendering speed. The EM display is smaller on the screen, hence the higher frame rate. .................................................... 29

Table 5.3: Performance results for 3-channel volumes (kidney cells), rendered in a window of 1600 × 900 pixels. ................................................................. 30

Table 5.4: Performance results for a multi-channel volume in Figure 5.4(b) with different numbers of existing channels. .................................................... 31
ACKNOWLEDGEMENTS

Many people have contributed and provided support to my work. First, I would like to thank my two primary advisors, Dr. Jürgen Schulze and Prof. Matthias Zwicker, for their time and technical advice. Without their exceedingly high expectations, I would not have embarked on, nor completed, my thesis. The demands they made of me pushed me to my mental limits and strengthened my resolve and focus. I want to attribute any success I’ve had to them; it has been an excellent learning experience and opportunity.

Furthermore, I would like to thank Ruth West, Maryann Martone and the rest of the NCMIR group for providing the volume data sets. In collaboration with NCMIR, we have been able to use our application tool to visualize their useful and beautiful volume data sets for my thesis. Ruth and Maryann also helped critique my submission to Siggraph and reimburse my trip to the Siggraph 2007 conference.

I want to thank my thesis committee including Ruth West, Prof. Sam Buss and Prof. Henrick Wann Jensen for taking time to review my thesis. And, I thank everyone else that I’ve asked for help at some point.

Finally, I am deeply indebted to Gary Scillian from High Moon Studios, who took so much of his valuable time to read and correct my thesis paper more than once.
ABSTRACT OF THE THESIS

Bridging the Resolution Gap:
Superimposition of Multiple Multi-Channel Volumes

by

Chih K. Liang
Master of Science in Computer Science
University of California, San Diego, 2008

Matthias Zwicker, Chair

I present a new algorithm to render multiple overlapping multi-channel volumes in real-time. The approach uses textured slices for rendering volumes and hardware-accelerated shaders for the application of transfer functions and the blending of data channels.

Volume rendering is an important technique for visualizing scientific and biological data. The National Center for Microscopy and Imaging Research (NCMIR) images biological systems at multiple scales and resolutions using a variety of techniques. Visualization across scales is useful to scientists because 1) they often must combine multiple microscopies with different resolutions to understand a specimen; and 2) it is more efficient to take lower resolution views of broad expanses of tissue and higher resolution views of features of interest. Combining them together allows detail to be appreciated in context. Generating data sets at different levels of spatial resolution does not provide the precise coordinates required to co-locate, or superimpose the resulting
volumes automatically. The algorithm enables users to interactively co-locate multiple nested volumes in real-time.
Chapter 1

Introduction

Volume rendering allows the visualization of volume data sets. Our volume data sets consist of 3D arrays of voxels. Voxels are equivalent to volumetric pixels in 3D space. Similar to a pixel, a voxel can be associated with a number of scalar values. The volume data set is a single-channel when there is only one value in every voxel. It becomes a multi-channel volume when every voxel has more than one channel. The number of channels is uniform across all voxels.

Volume data sets are collected using various imaging methods. One of the most common methods, X-rays, generates single-channel volumes with voxel values corresponding to the densities of the imaged structure at the equivalent locations in space. The method we are focusing on is laser scanning confocal microscopy. Confocal microscopy images containing multiple channels are produced as a laser beam is focused within a biological specimen containing multiple fluorescent dyes, each emitting fluorescent light of a different wavelength that is observed as a distinct color. Specimens are placed on computer controlled microscope stages and the laser is focused within them point by point at multiple coordinates in x, y and z, thus generating a volume that can be optically sectioned. The image is reconstructed by specialized software into an ultra wide-field tiled mosaic, or volume. The values within a channel correlate to the presence of a particular fluorescent dye.
Multi-volume rendering serves as an important technique for the interpretation of many different volumes simultaneously in a scene. Biologists often work with 3D volume images of multiple resolutions obtained from different microscopic imaging modalities, such as the combination of laser scanning confocal microscopy (LM) and electron tomography (EM). Researchers from the National Center for Microscopy and Imaging Research (NCMIR) obtain enormous volume data sets (greater than 10GB) from a variety of imaging techniques. Many of them are multi-channel LM volumes. Visualization of multiple overlapping multi-channel volumes can result in exposing interesting or important features and facts. Examples:

- After superimposition, comparison of healthy cells and those affected by various disease processes can indicate important features that reveal the differences between them.

- Multiple single-color confocal images of an identical structure can be interactively overlaid onto one another and form essentially a multi-channel volume to represent the structure with multiple colors at each point in space.

- After superimposition, data acquired using both LM and EM can be used to investigate how biological structures at smaller scales (EM) relate to or influence structures and their associated functions at larger scales (LM).

We developed a method, as described in our Siggraph 2007 poster (Liang, Schulze, West, Martone, and Zwicker, 2007), to render multiple overlapping multi-channel volumes interactively. Our method includes:

- Slice interleaving for texture-based multi-volume rendering – Recently, slice interleaving (Rößler, Tejada, Fangmeier, Ertl, and Knauff, 2006) was introduced to intermix the data from multiple volumes. Though other techniques exist, this texture-based technique can exploit the graphics hardware for speed.

- Pixel shaders for applying transfer functions and combining multi-channel data – Multi-volume rendering has been used only on single-channel volumes. For
the first time, we are extending it to multi-channel volumes using a pixel shader for fast rendering.

- Transfer function types and parameters – The specification of a transfer function has always been a tedious manual process for users. In our algorithm, users can generate the transfer function by specifying only the function type and the parameters.

- Color and intensity for individual channels – Traditionally, if a volume contains less than three channels, the channels are assigned red, green and blue for their colors (Schulze and Rice, 2004). To remove this limitation, the algorithm lets users assign any color to any channel. Also, users can tune out any channel by clearing the channel intensity.

- An interface for controlling multiple volumes individually – Every volume can be scaled, rotated and translated independently, so multiple volumes can be oriented and positioned for superimposition of volumes and alignment of feature points. Also, the image quality of a volume can be changed by adjusting its number of slices in the interface.
Chapter 2

Related Work

A lot of work and research have been published in the visualization community; only a select few concentrate on multi-channel volumes. The approaches presented in these papers rarely incorporate multi-channel volumes into multi-volume rendering. Moreover, in these papers, the number of channels is restricted to no more than three. The research areas related to our work are:

- Multi-volume rendering – In one of the early papers, Drebin, Carpenter, and Hanrahan (1988) present a technique for rendering single-channel volumes containing mixtures of materials. The resulting density is computed by summing up the products of the percentage of each material in the voxel times the assigned density of the material. In one of the most recent and relevant papers, Rößler, Tejada, Fangmeier, Ertl, and Knauff (2006) provided the framework for rendering multiple overlapping single-channel volumes by sorting the textured slices. Individual volumes can be manipulated and superimposed upon one another. Another important multi-volume work is TROVE (Two-level Representation of Volumetric Environments) proposed by Leu and Chen (1999). The two-level hierarchy reduces the storage consumption for rendering multiple volumes, but the algorithm breaks down for intersecting volumes. Plate, Holtkamper, and Frohlich (2007) created a framework that supports multiple arbitrarily intersect-
ing large volumetric data sets using multi-resolution octree-based structures. The interactive shader composer developed for the framework handles only single-channel volumes.

- Multi-channel volumes – Sakas, Vicker, and Plath (1996) proposed a volume rendering system that allows visualization of Laser Confocal Microscopy data in a convenient, fast and interactive way. However, a simple transfer function was implemented for only a single volume data set. Also, Razdan, Patel, Farin, and Capco (2001) present volume visualization of multicolor laser confocal data with ray casting techniques and pre-processing routines for noise removal, smoothing and edge enhancement. The three lasers of different colors simultaneously merge at the voxel level. The different colors are used for the different channels of a single volume. Recently, Schulze and Rice (2004) implemented a method with a pixel shader to visualize four-channel data sets. Their method can superimpose specifically a one-channel volume onto a three-channel volume from the same structure.

- Transfer functions – The direct manipulation widgets developed by Kniss, Kindlmann, and Hansen (2002, 2001) have been shown to reveal important structures (such as material boundaries and subtle surface properties) without concealing them in trivial regions. In addition to scalar data, the transfer functions can be applied to the multivariate data, which contains multiple scalar values at each sample point, using a matrix of first partial derivatives. However, the papers omit how the multi-dimensional transfer functions can be used for multiple overlapping volumes. Later, Kniss, Schulze, Wössner, Winkler, Lang, and Hansen (2004) present a new immersive visualization workspace layout and the transfer function design approaches for non-overlapping volumes. On the other hand, O’Conor, Voorheis, and O’Sullivan (2004) describe how confocal microscopy data can be displayed in a meaningful and useful manner with hardware-accelerated volume visualization. Dependent texturing with texture
shaders is used for implementing multi-dimensional transfer functions for a single volume.

- Rendering approaches – Many different rendering techniques exist for 3D visualization. Zuiderveld and Viergever (1994) integrate the different techniques using object-oriented methodologies. The software architecture offers flexibility and extensibility for future implementations. However, it is not sufficient for the data acquired with Laser Confocal Microscopy, because they can be huge in size and have low signal-to-noise ratio and other artifacts. Based on ray casting techniques, Cai and Sakas (1999) proposed several approaches to intermix data from different volumes. Similarly, the V-Objects developed by Grimm, Bruckner, Kanitsar, and Gröller (2004) depend on ray casting for multi-volume rendering. Their methods efficiently distinguish between regions of intersection and non-intersection using a brick-wise volume traversal scheme. The ray casting techniques yield more accurate results at the cost of rendering speed.
Chapter 3

Volume Rendering

To visualize 3D volume data, we must model light propagation through a partially opaque material. A simplified model has been proposed, as shown in Figure 3.1(a). At any point in space, there exist two values – emission and extinction coefficients. Hence, volume data can be represented by their continuous functions in 3D space.

![Figure 3.1: Simplified volume rendering model. (a) At each point $x$ inside the volume, while $\epsilon(x)$ determines the emission of light, $\phi(x)$ is the attenuation factor along the direction. (b) Points inside the volume are discretely sampled at regular spacing along a viewing ray.](image)

The emission coefficient determines the amount of light emitted from a point in the volume to the viewer, very similar to the property of color. As light travels along a viewing ray (from the viewer to a point inside the volume), it attenuates according to the extinction coefficient. Therefore, the attenuation determines the opacity at the point.
Other factors such as scattering and reflection are ignored for simplicity. The model also assumes isotropic absorption of light. With these assumptions, the resulting intensity of a viewing ray can be greatly simplified from the differential form of the equation of transfer (Schulze, Kraus, Lang, and Ertl, 2003). The resulting volume rendering equation is given by:

\[ I = \int_{x_{\text{start}}}^{x_{\text{end}}} e^{-\int_{x_{\text{start}}}^{x} \phi(x')dx'} \epsilon(x)dx \]  

(3.1)

The integral is evaluated from the entry \(x_{\text{start}}\) to the exit \(x_{\text{end}}\) of the viewing ray through the volume, with extinction and emission coefficients as \(\phi\) and \(\epsilon\), respectively.

Equation 3.1 can be approximated by sampling discretely at equal spacing along the viewing ray, as shown in Figure 3.1(b).

\[ I = \sum_{i=0}^{n-1} e^{-\sum_{j=0}^{i-1} \phi(x_{\text{start}} + j\Delta x) \Delta x} \epsilon(x_{\text{start}} + i\Delta x) \Delta x \]  

(3.2)

The discrete approximation divides the ray inside the volume into \(n\) segments of length \(\Delta x = (x_{\text{end}} - x_{\text{start}})/n\). \(n\) points are sampled at the spacing of \(\Delta\). It assumes \(\phi\) and \(\epsilon\) to be uniform within each segment.

By substituting

\[ \alpha_i = 1 - e^{-\phi(x_{\text{start}} + i\Delta x) \Delta x} \]

\[ C_i = \epsilon(x_{\text{start}} + i\Delta x) \Delta x / \alpha_i \]
we can rewrite the discrete equation as

\[
I = \sum_{i=0}^{n-1} \epsilon(x_{\text{start}} + i\Delta x)\Delta x \times \prod_{j=0}^{i-1} e^{-\phi(x_{\text{start}}+i\Delta x)\Delta x} \\
= \sum_{i=0}^{n-1} C_i \alpha_i \times \prod_{j=0}^{i-1} (1 - \alpha_j) \\
= C_0 \alpha_0 + C_1 \alpha_1 (1 - \alpha_0) + C_2 \alpha_2 (1 - \alpha_0) (1 - \alpha_1) + \\
\cdots + C_{n-1} \alpha_{n-1} (1 - \alpha_0) \cdots (1 - \alpha_{n-2})
\]

(3.3)

From the definitions, \( \alpha \) and \( C \) are the opacity and the color at the sampled point respectively. The expanded series (Equation 3.3) is the solution to the following equation with \( I = I_0 \) and \( I_n = 0 \):

\[
I_i = I_{i+1} (1 - \alpha_i) + C_i \alpha_i
\]

(3.4)

Essentially, this equation implements Equation 3.2. As a point is sampled from the nearest to the farthest distance to the viewer, its opacity and color are blended into the intensity.

The opacity and the color are not often defined for the volumes obtained for our work. The volumes are usually represented by a 3D array of voxels. A voxel is a volumetric pixel that contains values from data channels. Hence, channel data at an arbitrary point can be interpolated from those of adjacent voxels. Then, transfer functions are used to map the interpolated data values to opacity and/or color. Transfer functions are not trivial to specify, because the channel data might not correspond to color and opacity in a direct way. Optimal transfer function specification can prevent occlusion of important regions and highlight them with the desired color. Hence, users usually specify their own transfer functions based on their visualization preferences. Through trial and error, users can specify transfer functions which provide deeper insight into their volume data.

In texture-based volume rendering, 3D texture maps are used to store voxel data. When a volume is sliced into uniform slabs parallel to the viewing plane, the graphics
hardware interpolates slice samples from the 3D texture maps. A pixel shader calculates color and opacity per pixel from a slice sample. Then, a hardware implementation of Equation 3.4 is used to blend slice by slice. This blending procedure is called alpha blending. As each slice is blended in, the result is stored in the frame buffer that gets displayed to the screen.

Besides this texture-based method, we can implement the volume rendering equation with ray casting. While it produces a more realistic image using a volume rendering model with fewer assumptions, the rendering speed can be slow due to computations not supported by the graphics hardware, such as intersection tests. Hence, our approach is completely based on texture-based volume rendering for the sake of speed.

Similarly, multiple volumes can be rendered by merging them into a single volume by resampling the volumes (Nadeau, 2000) as a preprocessing step. Whenever individual volumes change or move relative to one another, the merged volume has to be computed. Therefore, it is an inefficient way to render multiple volumes interactively.

Texture-based volume rendering can be extended to multiple volumes, even though the simplified volume rendering model is specific to a single volume. However, the intermixing of data from multiple volumes (Cai and Sakas, 1999) can be achieved by treating slices from all volumes as slices from the single combined volume. Though the slices may not be equidistant in the overlapping region, the rendered result gives a fast approximation to the region.

For a single-channel volume, a transfer function is needed to map a sample value to a color or an opacity value. The procedure is called classification. This step can be more tedious for multi-channel volumes. The transfer functions can be multi-dimensional for multi-channel volumes, since the functions take values from multiple channels as input parameters. Hence, the multi-dimensional transfer functions can be difficult to visualize, and the design of the classification interface can be very challenging. To relieve users of the tedious task of manipulating individual complex transfer functions, our simple interface only requires users to assign different colors to different channels and to select transfer function types for classification.
Chapter 4

Implementation

Our interactive application tool renders multiple volumes with different numbers of channels and lets the users specify various options and parameters. In Section 4.1, we introduce the user inputs, including volume intensity, channel color and weight. In Section 4.2, we discuss three types of transfer functions and the parameters that can be specified by users. In Section 4.3, we describe the channel data values that users do not control directly.

For this chapter, we will use the following notations:

\[ i \] = The volume number
\[ j \] = The channel number
\[ v \] = The total number of volumes
\[ c_i \] = The total number of channels in the \( i^{th} \) volume

4.1 User Input Variables

The main input variables are volume intensity, channel color and channel weight. These controls can be manually adjusted to change the overall appearance of the volumes.
### Volume Intensity

<table>
<thead>
<tr>
<th>Notation</th>
<th>$I_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Floating-point number</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>$0.0 \leq I_i \leq 1.0, \forall i \in {1, 2, 3, \ldots, v}$</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>Controls final optical properties of specific volume. Higher value means higher RGBA values.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>When $I_1 = 0.0$, the $1^{st}$ volume becomes invisible because voxels are black and transparent.</td>
</tr>
</tbody>
</table>

### Channel Color

<table>
<thead>
<tr>
<th>Notation</th>
<th>$C_{i,j} = (R^{C_{i,j}}, G^{C_{i,j}}, B^{C_{i,j}})$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Vector of 3 floating-point numbers for red, green and blue components</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>$0.0 \leq R^{C_{i,j}} \leq 1.0, 0.0 \leq G^{C_{i,j}} \leq 1.0$ and $0.0 \leq B^{C_{i,j}} \leq 1.0, \forall i \in {1, 2, 3, \ldots, v}$ and $\forall j \in {1, 2, 3, \ldots, c_i}$</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>Specifies the color of a channel for its data. Higher channel data value results in brighter color.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>When $C_{1,2} = (1.0, 1.0, 0.0)$, the $2^{nd}$ channel of the $1^{st}$ volume displays shades of yellow.</td>
</tr>
</tbody>
</table>

### Channel Weight

<table>
<thead>
<tr>
<th>Notation</th>
<th>$W_{i,j}$</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>Floating-point number</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>$0.0 \leq W_{i,j} \leq 1.0, \forall i \in {1, 2, 3, \ldots, v}$ and $\forall j \in {1, 2, 3, \ldots, c_i}$</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>Controls the color contribution of a data channel to the color of its volume.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>When $W_{1,2} = 0.0$, the $2^{nd}$ channel of the $1^{st}$ volume is essentially ignored.</td>
</tr>
</tbody>
</table>

### 4.2 Transfer functions

Design of transfer functions has been difficult, especially for multi-channel volumes. Since visualization of a multi-dimensional function can be difficult, we designed the interface to be intuitive (and applicable) to our volume data. By changing a few parameters or selecting a type, users can specify a transfer function quickly and easily. While the users may not have the complete freedom to specify all possible transfer functions, our transfer functions are easy to adjust and less tedious to manage.
Our transfer functions map one floating-point number to another. Because a higher channel data value is assumed to have higher luminance as in biological images, the transfer functions are created to be non-decreasing. The implementation uses 1D textures as look-up tables for the transfer functions.

The transfer function for $j^{th}$ data channel in $i^{th}$ volume is denoted by

$$y = T_{i,j}^o(x)$$

and the transfer function for the opacity of $i^{th}$ volume, which allows the user to control the overall opacity, is

$$y = T_i^o(x)$$

with $0.0 \leq x \leq 1.0$ and $0.0 \leq y \leq 1.0$.

A transfer function for a data channel can be one of three types: Gamma function, High-Pass Filter, and Histogram CDF (Cumulative Distribution Function), whereas an opacity transfer function is either Gamma function or High-Pass Filter. For simplicity, all data channels in a given volume can specify only one type of transfer function, with different parameters for each channel. Moreover, the opacity transfer function for the volume must be the same type. The exception is when the transfer function type is Histogram CDF; the transfer function for opacity becomes a Gamma function.

![Figure 4.1: Transfer functions. Volume using (a) Gamma function with $\gamma^c = 0.5$ for all channels, $\gamma^o = 2$ for volume opacity, (b) High-Pass filter with $\sigma^c = 0.2, \eta^c = 2, \sigma^o = 0.6, \eta^o = 1$, (c) Histogram CDF with $\gamma^o = 4$.](image)
Data channels may contain an undesirable distribution of values. To alleviate this problem, Gamma, High-Pass, or Histogram CDF transfer functions can be applied. The volume opacity can also be adjusted with a transfer function. Figure 4.1 shows a volume with different types of transfer functions with specified parameters. Because the transfer functions are pre-computed and implemented using 1D textures, the performance of all transfer function types is the same.

### 4.2.1 Gamma Function

The function is $f(x) = x^\gamma$, where $\gamma$ is specified by the user for each Gamma function. See Figure 4.2.

<table>
<thead>
<tr>
<th>Notation</th>
<th>$\gamma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Floating-point number</td>
</tr>
<tr>
<td>Value</td>
<td>$0.0 \leq \gamma \leq \infty$</td>
</tr>
<tr>
<td>Usage</td>
<td>Changes gamma function curve to distribute output values evenly. High value is used when the histogram is in the high range; low value, low range.</td>
</tr>
<tr>
<td>Example</td>
<td>If $\gamma = 0.0$, all input values to map to the maximum value of 1.0; If $\gamma = 1.0$, all output values are unchanged from the input values, and this gamma value is best used when the histogram is even distributed.</td>
</tr>
</tbody>
</table>
4.2.2 High-Pass Filter

This type of function is the Butterworth equation (Gonzalez and Woods, 2002): 

\[ f(x) = \frac{1}{1 + (\sigma/x)^{2\eta}} \]

where \(\eta\) and \(\sigma\) are specified by the user for each function of this type (see Figure 4.3).

<table>
<thead>
<tr>
<th>Cut-off</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notation</strong></td>
<td>(\sigma)</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Floating-point number</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>(0.0 \leq \sigma \leq \infty)</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>Moves the middle point ((x = \sigma, y = 0.5)) of the function curve. Higher value is used for higher input value range to better distribute output values across its histogram.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>0.0 causes all values to have high output values, usually for a histogram with data in low range.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Notation</strong></td>
<td>(\eta)</td>
</tr>
<tr>
<td><strong>Type</strong></td>
<td>Floating-point number</td>
</tr>
<tr>
<td><strong>Value</strong></td>
<td>(0.0 \leq \eta \leq \infty)</td>
</tr>
<tr>
<td><strong>Usage</strong></td>
<td>Determines the slope up thru the middle point ((x = \sigma, y = 0.5)) of the function curve. Higher value for sharper cut-off.</td>
</tr>
<tr>
<td><strong>Example</strong></td>
<td>(\infty) makes a step function; 0.0 makes a smooth curve</td>
</tr>
</tbody>
</table>
Figure 4.4: Each image size is 512x512 with 8-bit intensity values. Each histogram is performed on 262144 points with 256 bins. (a) Histogram and Histogram CDF of the dark image on the left, (b) Histogram of output image on the left after mapping original image with Histogram CDF.

4.2.3 Histogram CDF

The distribution of data values from a channel is often unknown and can be very uneven. One result is that the channel data occupy the low range of values and occlude the bright voxels behind. Histogram CDF, also known as Histogram Equalization (Gonzalez and Woods, 2002), maps the data values to output values such that the output
values are spread more evenly across the possible data range. For example, a dark image with mostly low intensity values becomes brighter with a higher contrast ratio after applying this transfer function. Applied to a bright image, Histogram CDF maps it to a darker image with higher contrast.

The Histogram CDF transfer function requires no user input to automatically map a channel with an unknown distribution of intensity values. It takes only the data from a specific channel as input and performs the CDF calculation on the histogram of the channel data.

$$\text{CDF}(\text{hist}(x)) = \sum_{k=0}^{x} \text{hist}(k)$$

Since the opacity pertains to no particular data channel, Histogram CDF is excluded as a type of transfer function for $T_i^\phi$.

Figure 4.4 shows an image with an uneven distribution. The histogram of the image after applying the Histogram CDF is spread out to correct the original image. The distribution may not be perfectly even, because in this case there are 256 discrete values (8-bit data) for the intensity.

### 4.3 Data Variables

The data value for a channel at location $(x, y, z)$ is obtained by interpolation from the channel data of nearby voxels (Razdan, Patel, Farin, and Capco, 2001). $(x, y, z$ are floating-point numbers.)

<table>
<thead>
<tr>
<th>Notation</th>
<th>$D_{i,j}(x, y, z)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Floating-point number</td>
</tr>
<tr>
<td>Value</td>
<td>$0.0 \leq D_{i,j}(x, y, z) \leq 1.0, \forall i \in {1, 2, 3, \ldots, v}$ and $\forall j \in {1, 2, 3, \ldots, c_i}$</td>
</tr>
<tr>
<td>Usage</td>
<td>Sampled from a scanning technique for scientific visualization.</td>
</tr>
<tr>
<td>Example</td>
<td>When $D_{1,2}(0, 0, 0) = 1.0$, data value of the $2^{nd}$ channel in the $1^{st}$ volume at origin is maximum.</td>
</tr>
</tbody>
</table>
4.4 Pixel Shader

As a volume is sliced up as described in Section 4.6, the pixel shader computes color and opacity values of each point on the slices, using the user and data variables mentioned above.

The pixel shader accelerates volume rendering by using textures as look-up tables and performing parallel computations on GPU hardware (Rost, 2006). In addition, the 3D volume data is stored in the 3D textures that allow fast interpolations of values between voxels on the GPU.

The pixel shader is coded in OpenGL Shading Language (GLSL). The GLSL program calculates the color and opacity values at a location \((x, y, z)\) within the \(i^{th}\) volume. The shader output color \(\vec{C}^S\) is a vector (R, G, B):

\[
\vec{C}_i^S(x, y, z) = \sum_{j=1}^{c_i} I_i \times W_{i,j} \times T_i^c(D_{i,j}(x, y, z)) \times \vec{C}_{i,j}
\]

(4.1)

The shader output opacity \(\alpha^S\) can be computed according to one of the two equations:

\[
\alpha_i^S(x, y, z) = \frac{\max\{I_i \times W_{i,j} \times T_i^c(D_{i,j}(x, y, z)) \mid 1 \leq j \leq c_i\}}{\max\{I_i \times W_{i,j} \mid 1 \leq j \leq c_i\}}
\]

(4.2)

or,

\[
\alpha_i^S(x, y, z) = \frac{\sum_{j=1}^{c_i} I_i \times W_{i,j} \times T_i^c(D_{i,j}(x, y, z))}{\sum_{j=1}^{c_i} I_i \times W_{i,j}}
\]

(4.3)

The denominators normalize the alpha values to be between 0 and 1. After the calculations, a pixel with the output color and opacity values gets blended into the frame buffer.

In Equation 4.2, the maximum value is selected based on the product of channel weight and other values. Hence, the channel with the maximum channel weight can
Figure 4.5: Resulting images with the alpha equations and different channel weights. (a) and (b) are obtained by using Equations 4.2 and 4.3, respectively, and by specifying the weight of the green channel twice as much as other channel weights. (c) and (d) are obtained by using Equations 4.2 and 4.3, respectively, and by specifying equal channel weights.

Influence the opacity value more directly. When an important channel is assigned a maximum weight, the channel can be relatively visible in the image as the opacity and the color intensity increase with the channel weight. Figure 4.5(a) shows an image where a particular channel is emphasized, and by using the maximum-weight alpha equation, the channel color becomes dominant with higher intensity and opacity values. In Figure
4.5(c), all channel weights are the same, so all channels are equally prominent.

In Equation 4.3, all channels will be weighted in the calculation of the opacity value according to their channel weights. Because the opacity value is normally smaller from the result of the weighted average than the result of the maximum value, Figures 4.5(b) and (d) yield more transparent images and reveal more detailed internal structures than Figures 4.5(a) and (c). Hence, the weighted-channel alpha equation is our favorite choice for displaying the images with equal channel weights.

### 4.4.1 Shader Program

In the following shader code, the data channels are separated into different 3D texture maps to facilitate access of data in each channel. Besides, the volume has one transfer function for every channel and only one opacity transfer function. All uniform variables are passed from the main application, including the texture maps (sampler1D, sampler3D). For example, the main application computes and passes $I_i \times W_{i,j}$ to the shader as weight[MAXCHAN]. Also, GLSL assigns a texture the same data value for its red, green and blue components, if its format is specified as GL_LUMINANCE in the OpenGL application.

The following concise code provides the implementation details but fails to work properly, because arrays of 3D textures are not supported in GLSL. The simple workaround is to unroll and hard-code the texture names in the array and use them in the declaration as well as in the loop.
// Maximum number of channels allowed
const int MAXCHAN = 7;
// Number of channels in current volume
uniform int numChannel;
// 3D data of all channels in current volume
uniform sampler3D gl3dTex[MAXCHAN];
// Channel & Opacity Transfer functions
uniform sampler1D tfTexData[MAXCHAN];
uniform sampler1D tfTexOpacity;
// Volume Intensity*Channel Weights & Channel colors
uniform float weight[MAXCHAN];
uniform vec3 color[MAXCHAN];
// Denominator for normalization
uniform float normAlpha;
// Equation select for calculation of alpha
uniform int alphaMode;

void main() {
    gl_FragColor = vec4(0.0); float alpha = 0.0;
    for (int c = 0; c < MAXCHAN; c++) {
        if (c < numChannel) {
            // Get channel value from 3D texture map
            float data = texture3D(gl3dTex[c],gl_TexCoord[0].stp).r;
            // Transfer function mapping & weighted output
            float product = texture1D(tfTexData[c],data).r * weight[c];
            // Compute for shader color output
            gl_FragColor.rgb += product*color[c];
            // Compute for shader opacity output
            alpha = (alphaMode==0) ? max(alpha,product) : alpha+product;
        }
    }
    // Map opacity value with transfer function
    gl_FragColor.a = texture1D(tfTexOpacity, alpha/normAlpha).r;
}
4.5 Sorting Algorithm for Textured Planes

For every volume, 2D textured slices parallel to the screen (Cabral, Cam, and Foran, 1995) are calculated from the volume in graphics hardware, utilizing techniques for 3D texture-based volume rendering (Cullip and Neumann, 1993). The algorithm interleaves the textured slices of all volumes and renders them from farthest to closest with respect to the viewer (Rößler, Tejada, Fangmeier, Ertl, and Knauff, 2006), as shown in Figure 4.6. As the number of slices and the distance change, the opacity must be adjusted to ensure consistent image result. The corrected opacity $\alpha_2$ is calculated to be $1 - (1 - \alpha_1)^{d_2/d_1}$, where $d_1$ and $d_2$ are the slice distances before and after the change (Schulze, Kraus, Lang, and Ertl, 2003).

![Figure 4.6: Interleaving 6 slices (d_1 apart) from Volume 1 and 4 slices (d_2 apart) from Volume 2.](image)

Whenever slices of different volumes are rendered, it switches to the respective transfer functions of the current volume. The algorithm is:

```
1 for all slices s[k] do
```
for each volume $v[i]$ do
    Compute farthest slice $v[i].fs$
    if $v[i].fs.depth > s[k].depth$ then
        $s[k] := v[i].fs$
    endif
endfor
if $s[k].volume != s[k-1].volume$ then
    Switch transfer functions and etc
endif
Render $s[k]$
endfor

The complexity is $O(l \times v)$, where $l =$ total number of slices and $v =$ total number of volumes.

4.6 Blending Slices

The interpolated pixel values from each slice of 2D texture are blended into the pixels in the frame buffer using graphics hardware acceleration (Westermann and Ertl, 1998). The frame buffer will be output to the screen after all slices have been rendered.

There are two popular blending options for displaying biological volume data – Alpha Blending and Maximum Intensity Projection (MIP). The options can be selected in the user application for visualization of 3D volume data.

4.6.1 Alpha Blending

Alpha Blending provides a means to display transparency in 3D objects. If the depth-sorted textured slices (Kruger and Westermann, 2003) are numbered from 1 to $l$, the output color $\mathbf{C}_k$ at the screen location $(s, t)$ in the frame buffer after blending in
textured slice $k$ is

$$\vec{C}_k(s,t) = (1 - \alpha^S_i(x, y, z)) \cdot \vec{C}_{k-1}(s,t) + \alpha^S_i(x, y, z) \cdot \vec{C}_i^S(x, y, z) \quad (4.4)$$

where location $(x, y, z)$ in world space, on slice $k$ of $i^{th}$ volume, is projected to pixel $(s, t)$ in screen space. The equation equivalent to Equation 3.4 is built in OpenGL to accelerate GPU-based texture blending. In OpenGL, $\vec{C}_0$ is cleared to 0, as the frame buffer is initialized to black.

The OpenGL blending equation (4.4) is recursively iterated each time a new textured slice is to be rendered. $\vec{C}_k(s,t)$ becomes the final color of the screen pixel at $(s,t)$ when $k$ is total number of slices, at which point all slices have been rendered on screen. Consequently, all multi-channel volumes are visualized.

### 4.6.2 MIP (Maximum Intensity Projection)

When rendering a textured slice, OpenGL simply replaces Equation (4.4) with

$$\vec{C}_k(s,t) = \max\left(\vec{C}_{k-1}(s,t), \vec{C}_i^S(x, y, z)\right) \quad (4.5)$$

Every frame buffer pixel and its incoming pixel are compared, and the one with the larger color intensity is updated to the frame buffer. Hence, MIP is suited to images with a black background. However, MIP offers less perception of depth and occlusion in volume rendering than Alpha Blending does. For instance, the rendering order of slices makes no difference in the output image for MIP. Hence, Alpha Blending is mostly used throughout the sections.
Chapter 5

Results

Our visualization tool is MultiVOX (Multiple multi-channel VOlumes eXplorer) and is based on the framework of DeskVOX (Schulze, 2006). Its graphics API is OpenGL GLUT (OpenGL.org, 2000), and its simple graphical user interface is implemented with GLUI (Rademacher, 2006). Currently, the same source code runs on both Windows and Linux. All experiments are performed on a Windows XP machine (3.15GHz Intel Core 2 Duo, 2GB RAM, Nvidia 320MB GeForce 8800GTS).

Every volume used in all experiments in this section has one byte of value for each channel in a voxel. Hence, the file size of a volume can be approximated by taking the product of the number of voxels in all dimensions and the number of channels and dividing by $2^{20}$ to get the size in megabytes. However, the number of voxels in each dimension must be increased to the next multiple of 2 to be compatible with the older video cards. The volume file size that can be rendered is currently limited by the video memory size. In all tests, the number of slices is selected for consistency to be the minimum of the numbers of padded voxels in all three dimensions regardless of the viewing direction.

According to Table 5.1 measured from Figure 5.1, the current slice-sorting algorithm shows insignificant performance changes despite changes in the overlap of the volumes. It performs sorting without the intersection testing or hierarchy traversing...
Figure 5.1: Overlapping 2-channel volumes to different degrees: (a) Full, (b) Half, (c) None.

Table 5.1: Performance results from Figure 5.1, rendered in a window of $1600 \times 900$ pixels. The volume sizes in number of voxels are $512 \times 512 \times 52$ and $256 \times 256 \times 104$. Numbers of volume slices are 64 and 128 with MIP blending method. Render time includes processing time for OpenGL commands, including \texttt{glFinish} and \texttt{glDraw}.

<table>
<thead>
<tr>
<th></th>
<th>Full</th>
<th>Half</th>
<th>None</th>
</tr>
</thead>
<tbody>
<tr>
<td>Render time</td>
<td>36.369 ms</td>
<td>22.588 ms</td>
<td>17.646 ms</td>
</tr>
<tr>
<td>— GL draw</td>
<td>0.461 ms</td>
<td>0.436 ms</td>
<td>0.447 ms</td>
</tr>
<tr>
<td>Misc time</td>
<td>0.772 ms</td>
<td>0.877 ms</td>
<td>0.892 ms</td>
</tr>
<tr>
<td>— Slice sorting</td>
<td>0.041 ms</td>
<td>0.040 ms</td>
<td>0.040 ms</td>
</tr>
<tr>
<td>Frame rate</td>
<td>$\sim$27 fps</td>
<td>$\sim$42 fps</td>
<td>$\sim$57 fps</td>
</tr>
</tbody>
</table>

common in ray casting. The miscellaneous time, which includes the sorting time and other computations, also gives minor differences. However, the render time varies significantly because it takes into account the GPU processing of image data, the rendering and frame buffer blending. When the volume is at the origin with perspective projection, it is the closest to the screen. A closer volume results in a larger image on the screen.
and a slower rendering time due to the GPU fill rate.

Figure 5.2: Multiple multi-channel volumes collected from NCMIR in 1600 × 900 window: (a) Rendering of volume slices according to the sequential order of the volumes. Running at ∼14fps; (b) Rendering after interleaving depth-sorted volume slices. Running at ∼14fps.

According to Table 5.1, the sorting and interleaving algorithm has negligible impact on the frame rate. Figure 5.2 demonstrates that sorting and interleaving do not noticeably decrease the frame rate. Also, the interleaving gives the correct output image. As shown in Figure 5.2(a), a previously rendered volume seems to be occluded by the
more recently rendered volume. If a volume is in front of another, the other volume will appear as if it is in front. After the depth-sorted slices are interleaved and rendered, the tool shows accurate occlusion cues on a per slice basis.

![Images showing different volume renderings](image1.png)

Figure 5.3: (a) LM: 512 × 246 × 128 voxels, (b) EM: 256 × 512 × 128 voxels, (c) LM and EM superimposed with sequential rendering of LM first and EM later, (d) LM and EM superimposed with slice interleaving.

NCMIR provided two single-channel volume data sets – light microscopy (LM) and high resolution electron microscopy (EM), each representing the same section within the dendrite of a neuron. The tool allows multiple volumes to be manipulated individually. After using the tool to superimpose the EM volume on the LM volume (Figure 5.3), I measured the frame rates for each volume and their combination and list the results in Table 5.2. The multi-volume rendering seems to be slower by roughly 14% than the single-volume rendering. Also, the table shows that the image size on the screen determines the frame rate much more than the sorting algorithm or the number of volumes or channels. Hence, the rendered image sizes on the screen can be limited to achieve real-time speed for multiple volumes.

In addition, I took similar measurements on two 3-channel volume data sets
Table 5.2: Performance results for single-channel volumes (dendrites), rendered in a window of $1600 \times 900$ pixels. The major difference in rendering time between EM and LM is caused by the GPU fill rate. The pixel size of the volume rendered on the screen affects the rendering speed. The EM display is smaller on the screen, hence the higher frame rate.

<table>
<thead>
<tr>
<th></th>
<th>LM</th>
<th>EM</th>
<th>LM+EM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential</td>
<td>Interleaving</td>
<td></td>
</tr>
<tr>
<td># Voxels</td>
<td>512×246×128</td>
<td>256×512×128</td>
<td>N/A</td>
</tr>
<tr>
<td>Distance</td>
<td>0.165×0.12×0.15</td>
<td>0.03×0.04×0.02</td>
<td>N/A</td>
</tr>
<tr>
<td># Slices</td>
<td>128</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td>Render time</td>
<td>23.634 ms</td>
<td>4.911 ms</td>
<td>26.936 ms</td>
</tr>
<tr>
<td>– GL draw</td>
<td>0.195 ms</td>
<td>0.049 ms</td>
<td>0.355 ms</td>
</tr>
<tr>
<td>Misc time</td>
<td>0.347 ms</td>
<td>0.441 ms</td>
<td>0.706 ms</td>
</tr>
<tr>
<td>– Slice sorting</td>
<td>No sorting</td>
<td>0.355 ms</td>
<td>0.399 ms</td>
</tr>
<tr>
<td>Frame rate</td>
<td>∼42 fps</td>
<td>∼188 fps</td>
<td>∼36 fps</td>
</tr>
</tbody>
</table>

Figure 5.4: 3-channel volume data sets from structures within a kidney. (a) Low-resolution superset: $512 \times 256 \times 128$ voxels, (b) High-resolution subset: $256 \times 256 \times 128$ voxels, (c) Superimposition of the 2 volumes with sequential rendering of (a) first and (b) later, (d) Superimposition of the 2 volumes with slice interleaving.
Table 5.3: Performance results for 3-channel volumes (kidney cells), rendered in a window of 1600 × 900 pixels.

<table>
<thead>
<tr>
<th></th>
<th>Superset</th>
<th>Subset</th>
<th>Superimposition</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>512×256×128</td>
<td>256×256×128</td>
<td>N/A</td>
</tr>
<tr>
<td># Voxels</td>
<td>2.9×4.5×1.34</td>
<td>2.0×1.875×1.39</td>
<td>N/A</td>
</tr>
<tr>
<td>Distance</td>
<td>128</td>
<td>128</td>
<td>256</td>
</tr>
<tr>
<td># Slices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Render time</td>
<td>86.057 ms</td>
<td>10.221 ms</td>
<td>96.947 ms</td>
</tr>
<tr>
<td>– GL draw</td>
<td>0.047 ms</td>
<td>0.047 ms</td>
<td>0.096 ms</td>
</tr>
<tr>
<td>Misc time</td>
<td>0.275 ms</td>
<td>0.338 ms</td>
<td>1.013 ms</td>
</tr>
<tr>
<td>– Slice sorting</td>
<td>No sorting</td>
<td></td>
<td>0.052 ms</td>
</tr>
<tr>
<td>Frame rate</td>
<td>~12 fps</td>
<td>~95 fps</td>
<td>~10 fps</td>
</tr>
</tbody>
</table>

table from structures within the kidney. The high-resolution image is nested within the low-resolution image, as shown in Figure 5.4. The performance results are listed in Table 5.3. The rendering speed of multiple volumes is only 2 fps lower than that of the slowest single volume. By increasing the rendering speed of the single volume, the multiple-volume rendering can be increased.

If individual volumes can be rendered separately in real time, the users can manipulate multiple volumes interactively. They will be able to position the volumes, match the corresponding features within the volumes, and superimpose the volumes.

Table 5.4 shows the frame rates measured for the volume in Figure 5.4(b) with different numbers of data channels removed. As expected, the frame rate increases with decreasing numbers of channels, because the volume size, along with 3D texture size, decreases proportionally, and more arithmetic calculations need to be executed.

A multi-channel volume can have different colors assigned to the data channels. Figure 5.5 shows the 3-channel volumes that are applied a different color scheme. The color scheme has no impact on the performance, because the shader merely uses different values for the channel colors. The color scheme offers flexibility for the users to customize their output image. Individual data channels can be turned off by setting the channel weights to zero.
Table 5.4: Performance results for a multi-channel volume in Figure 5.4(b) with different numbers of existing channels.

<table>
<thead>
<tr>
<th># channels</th>
<th>Frame rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>~12 fps</td>
</tr>
<tr>
<td>2</td>
<td>~16 fps</td>
</tr>
<tr>
<td>1</td>
<td>~31 fps</td>
</tr>
</tbody>
</table>

Figure 5.5: (a) Volumes with Cyan, Yellow, Magenta as channel colors; (b) Volumes with Red, Green, Blue as channel colors.

The application can display volume data sets by blending textured slices with Maximum Intensity Projection or Alpha Blending, as in Figure 5.6. MIP and Alpha Blending equations (4.5 and 4.4) are implemented in the GPU hardware and invoked through the use of the OpenGL command `glBlendEquation`. Since the blending runs on fixed function hardware, the performance can vary depending on the GPU. In this case, MIP performs slightly better than Alpha Blending.
Figure 5.6: (a) Maximum Intensity Projection of volumes at 21.7 fps; (b) Alpha Blending of volumes at 22.3 fps.
Chapter 6

Conclusions & Future Work

If individual volumes are relatively small on the screen, our program can achieve real-time rendering speed displaying multiple overlapping multi-channel volumes. The speed is crucial for the users to visualize their volume data interactively. Essentially, the superimposition of multi-channel volumes circumvents the static merging of channels or volumes. The low-cost slice sorting makes the visualization of multiple volumes possible. Even though it may be a coarse approximation to the volume rendering model, the visualization result provides occlusion and positional cues in overlapping regions, so that the users can distinguish the relative positions of different volumes.

The user interface for the transfer functions was the most time-consuming part of our implementation, but it gives users simple yet effective options in visualizing their volume data. The users are not required to specify multi-dimensional transfer functions. Though we worked mainly with confocal images, our design can also be applied to volumes from other imaging technologies. Also, the interface for other options, such as MIP and channel colors, lets users satisfy their own preferences for the visualization.

Aligning multiple 3D volumes with conventional 2D displays is quite difficult due to occlusion and a lack of positional cues. These limitations can be overcome by implementing the software in an immersive, tracked virtual reality environment.

Furthermore, the size of every volume (number of voxels mainly) is currently
limited by the texture memory capacity of the graphics hardware. Large volumes can only be loaded into the application tool and rendered correctly after the bricking algorithm is used on this framework. The bricking algorithm divides a volume into smaller volumes that fit into the texture memory and get rendered.

The current derivation of the opacity values using the shader is intuitive and simple for multi-channel volumes. Many possibilities are available and may suit specific needs. The opacity can be the mean, median, or minimum of data values from all channels. Alternatively, a multi-dimensional gaussian function can be incorporated into the calculation of the opacity. However, the current two options can be sufficient in many cases.
References


