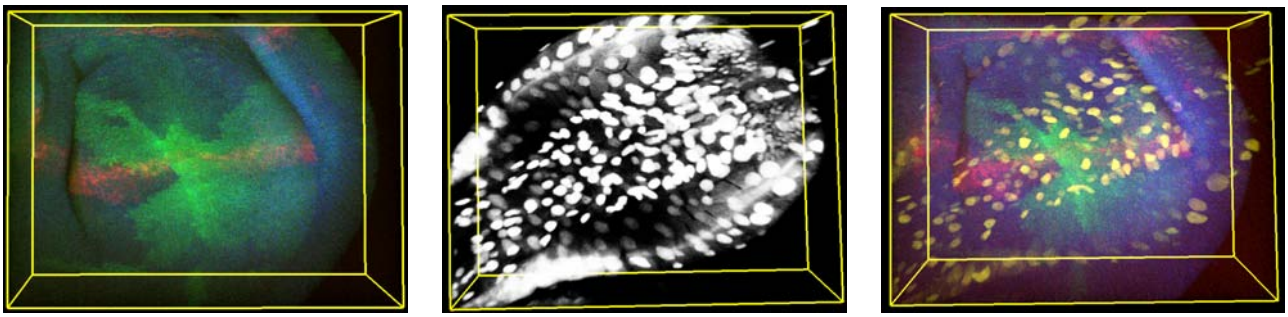
Real-Time Volume Rendering of Four Channel Data Sets

Jürgen P. Schulze Alexander Rice

{schulze|acrice}@cs.brown.edu
Computer Science Department
Brown University
Providence, RI



(a) (b) (c)

Figure 1: Sample of the larva of a fruit fly (Drosophila): (a) Three channels; (b) single channel; (c) combined data set with four channels.

ABSTRACT

We present a novel method to encode four data channels in a volumetric data set, and render it at interactive frame rates with maximum intensity projection (MIP) using textured polygons. The first three channels are stored in the volume texture's red, green, and blue components. The fourth channel is stored in the alpha channel. To achieve real-time rendering speed we are using a pixel shader.

1 INTRODUCTION

Our research has come out of a collaboration with Brown University biologists who work with images from confocal laser scanning microscopes (CLSM). These microscopes generate stacks of images (z-stacks) from fluorescent specimen. Sample sizes are typically in the range of tens of micrometers. Scans done with different laser wavelengths result in separate intensity images, with the voxels (volumetric pixels) perfectly co-registered between images. The voxels are located on a cartesian grid. In the following, the word *channel* describes the data scanned at a particular wavelength. The biologists are used to viewing the reconstructed samples rendered using maximum intensity projection (MIP), using the software that the microscope is delivered with.

The microscope currently used by our collaborators is a Leica TCS SP2 [2]. It scans up to four different data channels and outputs them as one TIFF image per channel and per z-depth. The slice images are typically 512^2 pixels, and they are taken at a few dozen z-depths.

With standard image processing software, the biologists reconstruct the sample in the following way. They map three of the channels to red, green, and blue, respectively (RGB, see Figure 1a). The fourth channel (Figure 1b) is assigned a hue that ideally does not (or not abundantly) exist in the RGB data set. The images of the fourth channel are then superimposed with the images of the other three channels (Figure 1c): For each voxel, the selected hue is multiplied by the fourth channel's intensity value.

Our method allows to render a four-channel data set in real-time, with interactive changes of the RGB channels' intensities, as well as the selection of the hue for the fourth channel. The advantage for the biologists is that they can much find a hue that does not occur in the RGB image much faster than before, which makes it easier for them to create an unambiguous fourth channel.

Rendering the data set in real-time while allowing the above interactive changes is possible with a pixel shader. Since we do not alpha-blend the slices together but use MIP, we do not need to deal with complex multi-dimensional transfer functions like those presented in Kniss et al. [1].

2 RENDERING

The first three channels of the volume data set are stored in the red, green, and blue components of a volumetric texture, which requires the OpenGL extension `GL_EXT_texture3D` to be supported. The volume can be rendered directly by creating a 3D texture and rendering with MIP, which requires the OpenGL extension `GL_EXT_blend_minmax`.

The fourth channel is stored in the alpha channel of the volume texture. It is rendered with an arbitrary, user selectable color. This color can potentially be selected from of the entire 24 bit color space. However, for optimum results a color with maximum intensity should be used, so that the renderer can use the full intensity range to map the fourth channel. So in the hue, saturation, and brightness color space, only hue and saturation should be selectable by the user. In our experiments, good choices have been gray, yellow, purple, and cyan, each of which consist of equal amounts of two or three of the basic colors red, green, and blue.

We wrote a pixel shader in Nvidia's Cg language [3]. It currently requires a GeForce FX graphics board at minimum. The Cg code of our pixel shader is shown in Table 1. The shader expects:

- a 3D texture consisting of the RGBA (4-channel) volume data set (`pix3dtex`);

Table 1: Pixel shader for 4-channel volume rendering.

```
struct PIN {
float3 coord3d :  TEXCOORD0;
};

float4 main(
const sampler3D in uniform pix3dtex :  TEXTURE0,
const sampler2D in uniform pixLUT,
const float3 in uniform pixModifier,
const PIN in pin) :  COLOR0{

uniform float4 origColor = tex3D(pix3dtex,pin.coord3d);
uniform float4 surfColor;
uniform float modColor = tex2D(pixLUT, float2(0,origColor.w)).w;

surfColor.x = tex2D(pixLUT, float2(0,origColor.x)).x+pixModifier.x*modColor;
surfColor.y = tex2D(pixLUT, float2(0,origColor.y)).y+pixModifier.y*modColor;
surfColor.z = tex2D(pixLUT, float2(0,origColor.z)).z+pixModifier.z*modColor;
surfColor.w = max(surfColor.x, max(surfColor.y, surfColor.z));

return surfColor; }
```

- the transfer function which maps intensity to data values for each channel (`pixLUT`);
- the color of the fourth channel at maximum intensity (`pixModifier`).

The shader first computes the color of the fragment as if no shader was active (`origColor`). Then it computes the intensity of the fourth channel (`modColor`). Finally, it computes the fragment color and opacity in `surfColor`. Note that we even set the alpha value of the fragment, so that it works with alpha blending, too, not only with MIP.

3 Results

We implemented our new shader in a software which allows to control the 4th channel's hue and the channel intensities directly from Brown University's virtual reality Cave.

In a performance test, we loaded three versions of a $512 \times 512 \times 28$ voxels data set, one at a time. One of the versions had three channels (RGB), another one was the separate fourth channel, and yet another one was the superimposed four channel data set. Table 2 lists the results: we get about a 30% performance penalty for the additional code required to render the fourth channel, as opposed to rendering only three channels. In the performance test, all data sets were rendered full screen at a resolution of 1024×768 pixels. The data sets were reconstructed using 155 textured polygons. Note that by using a different number of textures, the frame rate can be changed to arbitrary trade-offs between image quality and frame rate.

Figure 1 shows the three data sets we used in the performance test. They are CLSM scans of the larva of a fruit fly (Drosophila).

Table 2: Rendering performance.

# Channels	Frame Rate (fps)
1	4.3
3	1.8
4	1.4

4 Conclusion

Our new rendering method allows our collaborating scientists to work with their biological data sets in an interactive way, which helps them to explore new and complex data sets quickly on the desktop and in virtual environments. It saves them a great deal of effort which they used to spend on merging the fourth channel into the slice images by using standard image processing software. If they needed to change the fourth channel's color or intensity they had to start over again.

In the future we would like to improve the images we get using alpha blending with four channel data sets. We are trying to get around a true four-dimensional transfer function, or the Gaussians from Kniss et al.'s above cited approach. We think this should be possible because in our collaborators' data the channels are largely independent from each other.

References

[1] J. Kniss, S. Premoze, M. Ikits, A. Lefohn, C. Hansen, and E. Praun. *Gaussian Transfer Functions for Multi-Field Volume Visualization*. IEEE Visualization '03 Proceedings, 2003.

[2] Leica TCS SP2. *Leica Microsystems home page*. URL: http://www.leica-microsystems.com, 2004.

[3] Nvidia Cg Toolkit. *Nvidia developers' home page*. URL: http://developer.nvidia.com/page/cg_main.html, 2004.