User Manual

development of VVIS application:

Tobias Eberle <tobias.eberle@gmx.de >
Johannes Lampel <johannes@lampel.net>

development of Qt-GUI version:

Stefan Becker <stefan.becker@urz.uni-hd.de>
Ronald <rlautens@ix.urz.uni-heidelberg.de>

Practical Course Of:

Dr. Susanne Krömker
(IWR University of Heidelberg, Germany)

2006

Inhaltsverzeichnis

1 Introduction

vvis is designed to render 3D scalar fields, i.e. volume data collected by medical applications.
It can handle several file formats like the format of the University of Erlangen (used by openqvis (http://openqvis.sf.net)) and the format of the University of Heidelberg (used by the non-opensource
program Vrend). It can produce such files from a series of TIFF images. To display the volume different algorithms are available: texture method, texture method using pre-integration, raycasting, raycasting using pre-integration and shear warp.

2 Requirements

To compile vvis the following requirements have to met for the different platforms:

WIN

• qt 4.1.1 (at least)
• OpenGL
• libtiff (http://www.libtiff.org)

MAC

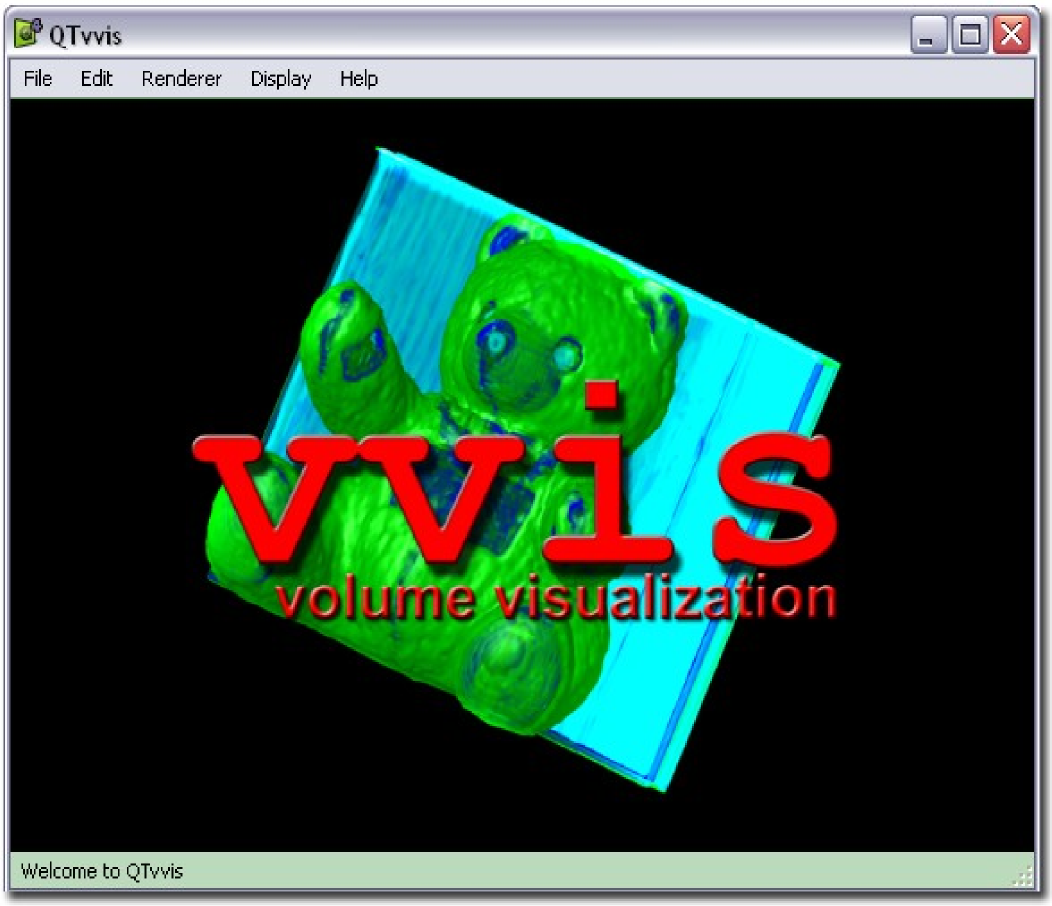
• qt 4.1.1 (at least)
• libtiff (http://www.libtiff.org)

LINUX

• qt 4.1.1 (at least)
• libtiff (http://www.libtiff.org)

3 Main window

3.1 The window



Welcome to QTvvis

Figure 3.1: This is the window you see after vvis has started. At the logo position in the middle the volume is displayed.

3.2 Description of the main menu

File	Open	Opens a file containing volume data. Currently OpenQVis and Vrend file format is supported. If you want to use several tiff files you have to convert them first using File—Convert. After choosing a file some calculations are done.
	Convert	Converting volume data from one file format to another. Please follow the instructions of the wizard.
	Information	Displays resolution and slice thickness of loaded voxel data.
	Save screenshot	Saves the volume as currently displayed as tiff image.
	Quit	close vvis
Edit	Transfer function	Opens or closes the transfer function editor. This editor is descripted in detail in chapter 4.
	Preferences	Opens a preferences dialog where you can configure vvis. See chapter 5 for details.
Renderer	(*)	Chooses the renderer used for displaying the volume. The differences between the renderes are described in chapter 6.
Display	Set Brightness	Opens a dialog to input a number between 0 and 255 to use a new brightness.
	Increment Brightness	Increments the brightness by 10 steps.
	Decrement Brightness	Decrements the brightness by 10 steps.
Help	About	Dialog showing some information about vvis.

3.3 The rendered volume display

Rotating

the volume Press the left mouse button and move the mouse around to rotate
the volume. In the preferences (see chapter 5) you can specify a lower texture size for the raycasting renderers used while rotating to get higher frame rates.

Zooming

Press the right mouse button and move the mouse fore and back.

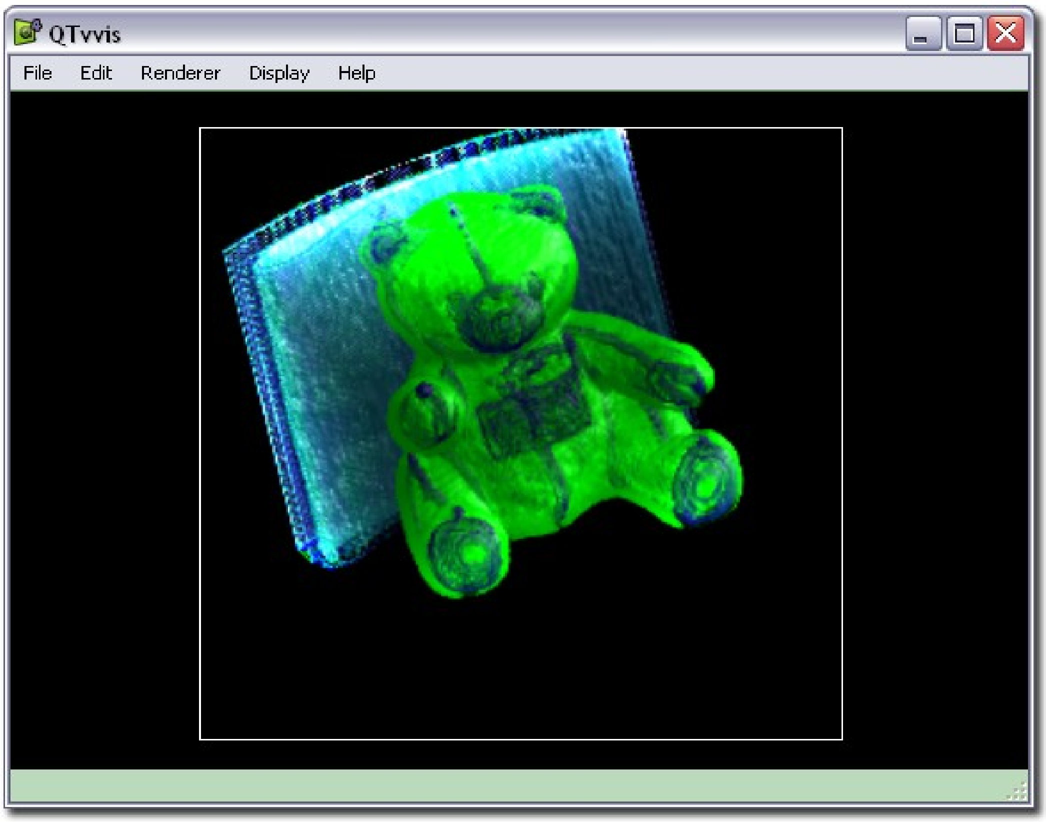


Figure 3.2: Main window showing volume data using the 'raycasting 2 renderer'.

4 Transfer Function Editor

With “Edit”—“Transfer function“ in the main menu (see chapter 3.2) you can open and close the transfer function editor.

4.1 What is a Transfer Function?

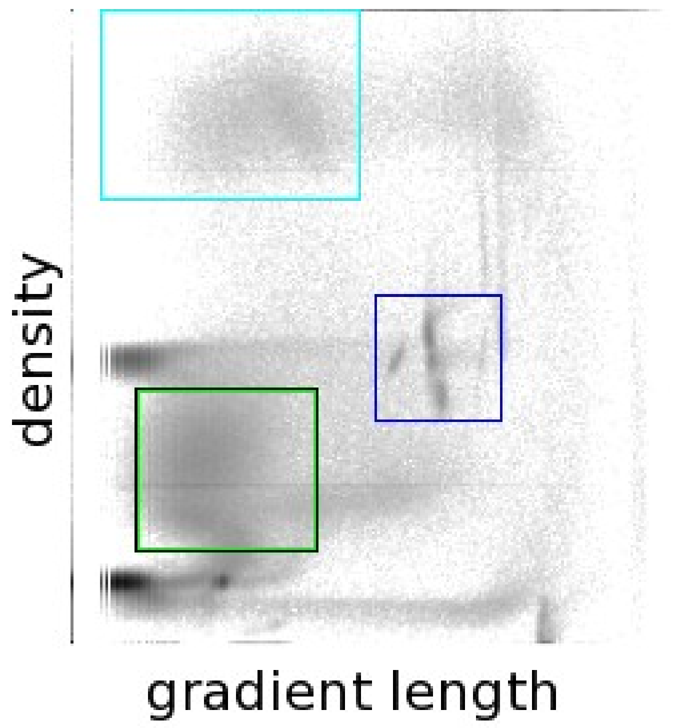
The data to be visualized by this program are density values on a rectangular grid. Drawing each of those voxels directly as some different shaded black and white transparent cubes might be possible, but this approach already fails if the data source produces density values around the object we want to look at which are not transparent.
Therefore this step has to be configurable. This is done by a so-called transfer function which assigns values needed for the display algorithms (see 6. Volume Renderers) to each of the voxels. The transfer functions used in vvis consist of one or more transfer function elements. Those elements specify the emissive, diffuse and specular color, the glossiness and absorbtivity for voxels inside a range of density values and gradient lengths.

4.2 Histograms

A histogram shows how often a certain voxel configuration exists in the currently loaded volume. There are two different histograms available: Density against gradient length and density only. The values at the current mouse position are displayed on the right below the histogram.

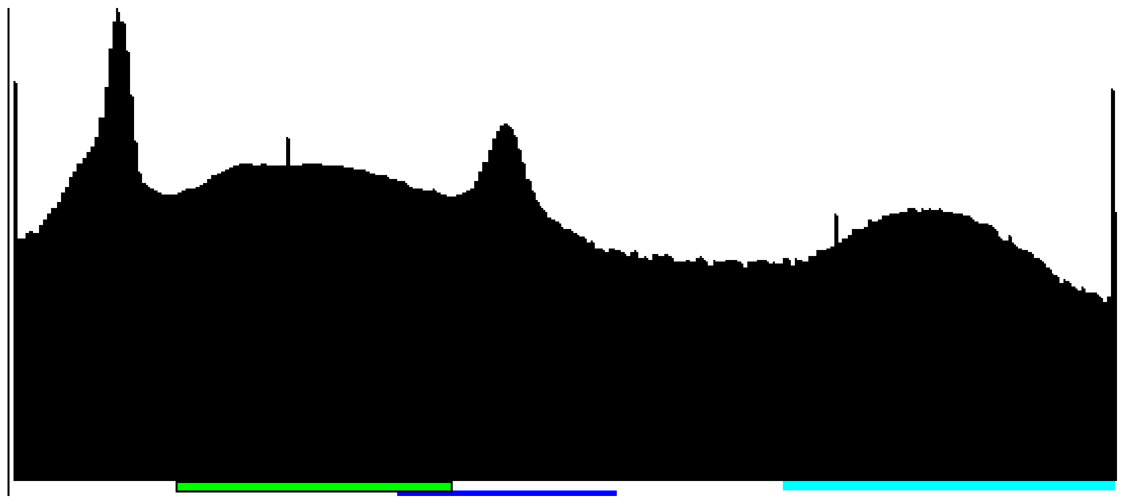
4.2.1 Density against gradient length

The gradient length is displayed at the x-axis, the density at the y-axis. Each point of the histogram is displayed in 256 gray values. The darker a value is the more often it occures. The origin is at the lower left. Existing transfer function elements are displayed as colored boxes using the “emissivity” color. If an element is selected a black border is displayed around the box. The Density is mapped linearly and the gradient length is mapped via the square root onto the square, since there are usually more interesting areas at the lower gradient lengths in the histogram. The brightness of each point on the histogram is proportional to the logarithm of the probability.
White represents a zero probability, black maximum probability.

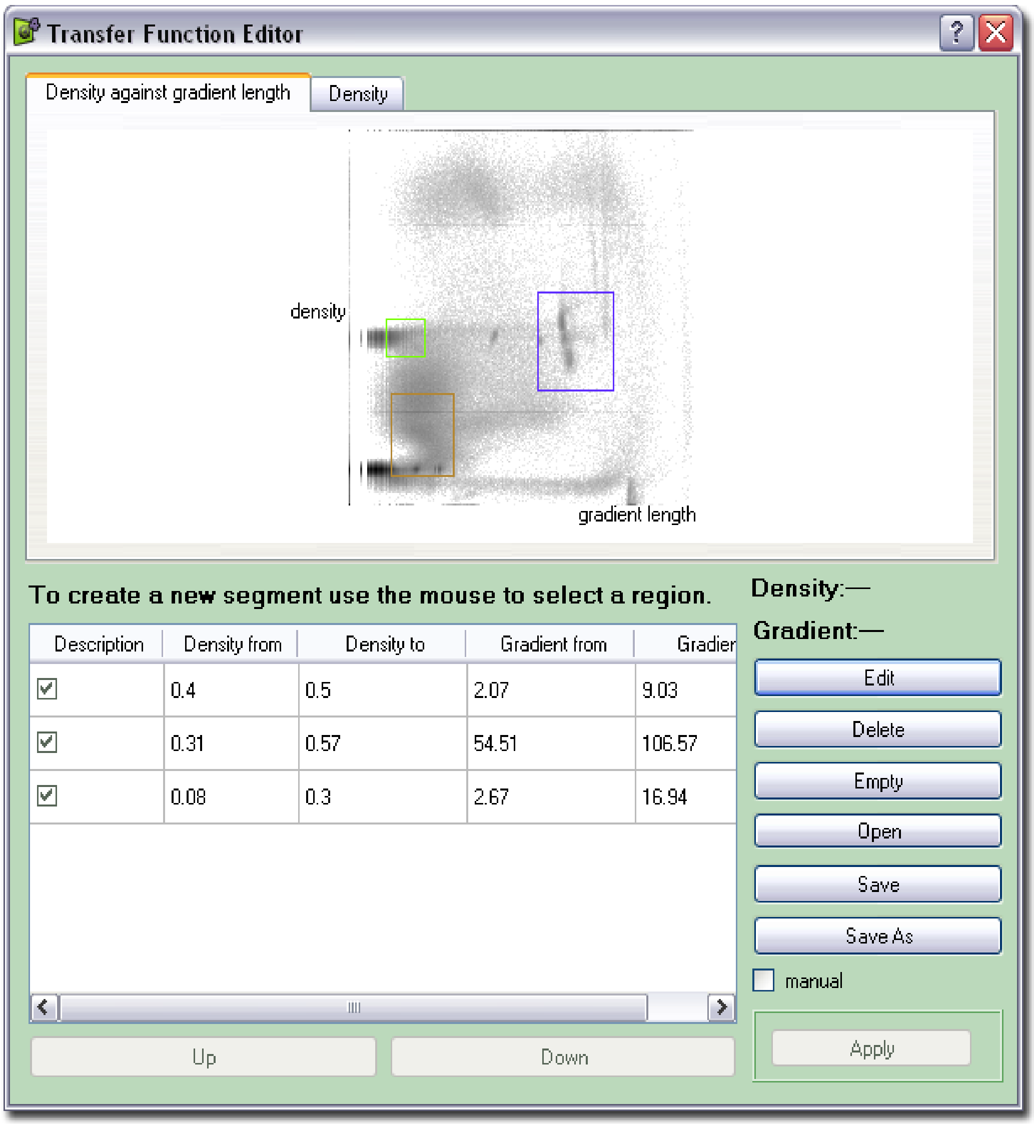


4.2.2 Density

The histogram shows how often a density value occures in the volume data. Below the histogram the existing transfer function elements are displayed as little boxes. The selected one is drawn with a black border.



4.3 Transfer function elements



To create a new segment use the mouse to select a region.

Description	Density from	Density to	Gradient from	Gradier
✓	0.4	0.5	2.07	9.03
✓	0.31	0.57	54.51	106.57
✓	0.08	0.3	2.67	16.94

4.3.1 Display

All transfer function elements are displayed in the list below the histogram. If you choose one element there it is highlighted in the histogram. For instance in the picture above the green transfer function element is selected. The list shows a description, whether it is enabled and the ranges of the element.

4.3.2 Manual apply

By default every change of the transfer function results in a recomputation of the computation that has to be done before displaying the first frame. If you want to make a set of changes this is annoying and wastes your time. To get rid of this feature enable “manual” checkbox. Doing so you have to click on “Apply” each time you want your changes be applied.

4.3.3 Create

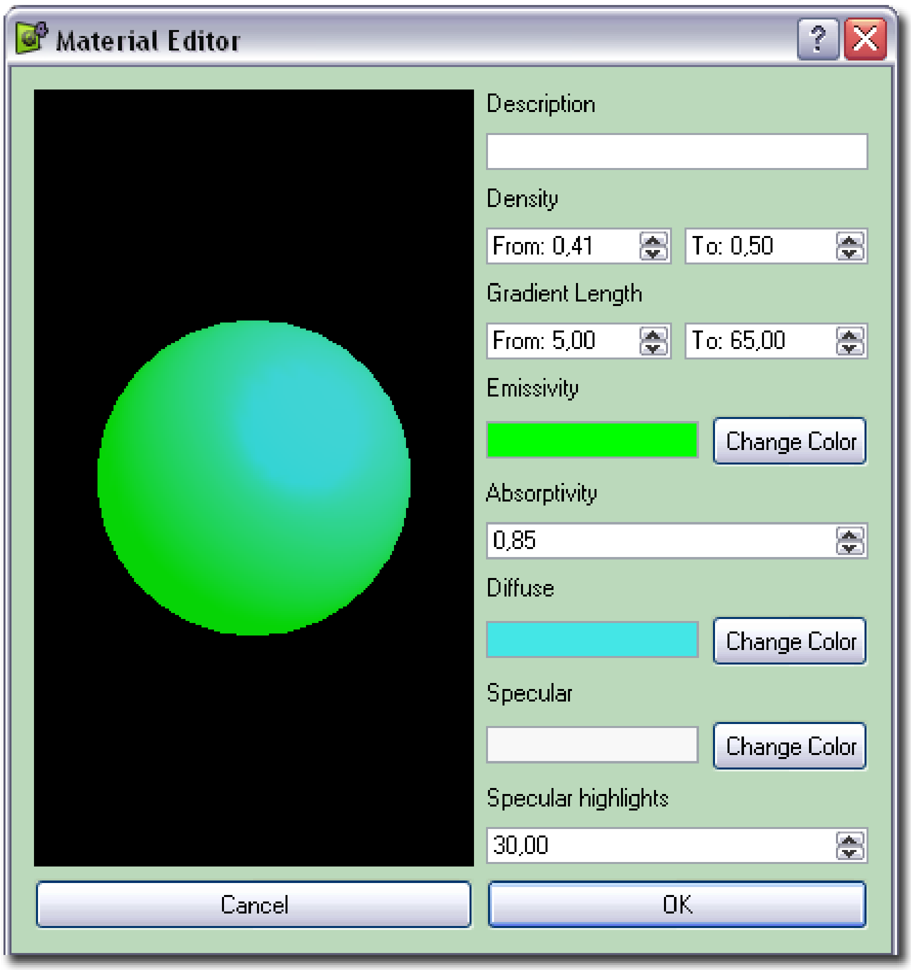
To create new transfer function elements use the mouse to select a region of the histgram: Move the mouse to the position to start at, press the left mouse button and leaved it pressed, move the mouse to stop position and release the mouse button. The “Material Editor” opens showing you your selected area. You can make some settings for the new element. Click on “OK” for creating the element or on “Cancel” for not creating it.

4.3.4 Order

The order of the transfer function elements is important if they overlap. The renderer uses the first one that fits. The transfer function element list shows the elements in the current order. Use the buttons ”Up“ and ”Down“ to change it.

4.3.5 Edit

If you want to edit an element select it and click on ”Edit“. The material editor opens:



You can specify a description, density and gradient length range, emissivity, diffuse and specular light and absorbtivity. The sphere on the left side shows a preview of the light colors and absorbtivity.

4.3.6 Remove

Transfer function elements can be deleted by selecting one and clicking on ”Delete“. With ”Clear“ you can delete all elements.

4.3.7 Save

You can save your created transfer function elements by clicking on ”Save“ or ”Save As“. The elements are saved to a XML file which can also be edited by hand. With ”Open“ you can load an existing transfer function file.

4.3.8 Enabling/disabling elements

Transfer function elements can be en-/disabled individually by clicking on the checkbox in the list. To enable or disable all elements right click on the list and choose the appropriate menuitem from the displayed contextmenu.

4.4 Hints for Classification of a new Volume Data Set

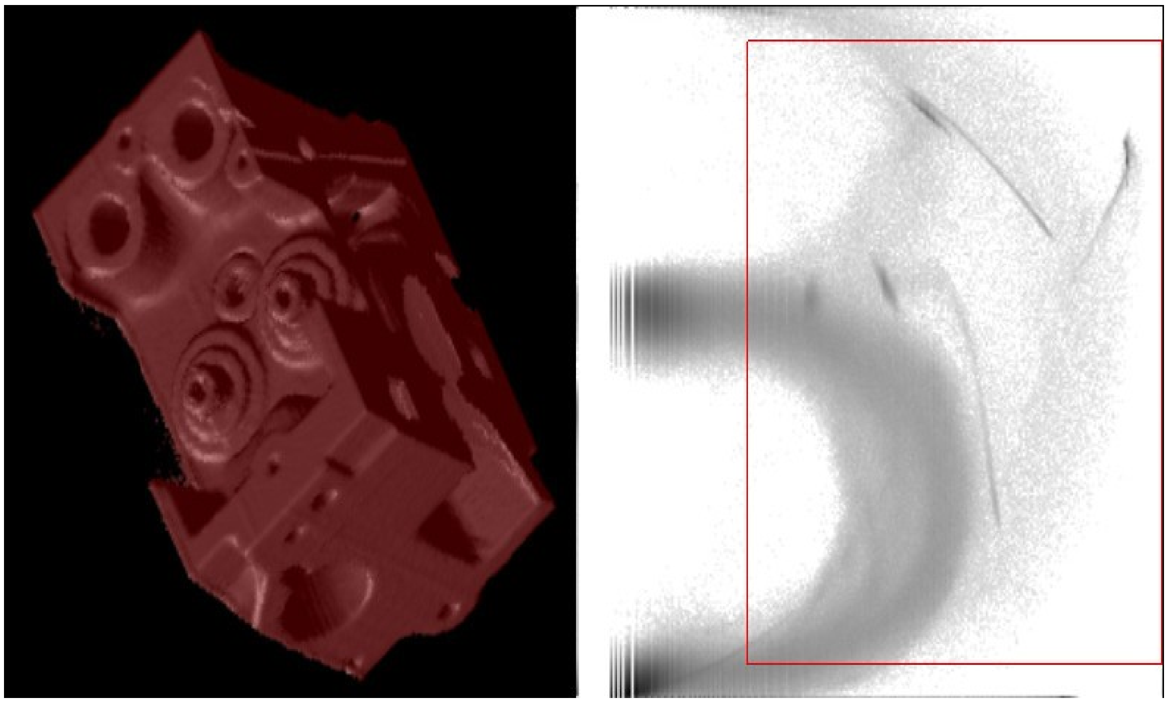


Figure 4.1: Getting a rough idea of the volume data.

First of all it is very useful to know what the data is all about, what we are aiming at to display. We need to orientate ourself, we need to know where is up, where is down, where we can find the main features we want to show. Here it is often useful to create a simple transfer function, then take a look what is already displayed and then continue to the details.

To do so, select a renderer which is running fast enough on your system, and which creates images showing enough details. In most cases the Shear-Warp-Renderer (see 6.3) is the best choice here. After changing the transfer function it needs some time for precalculating, but then you can examine the result at suitable refresh rates.
To get a rough idea of the volume data, just create a transfer function element which covers all or almost all density values, but which only covers higher gradient lengths. And already specify the lighting properties like diffuse and specular color to have a better impression of the object. (Figure 4.1)
Then take a look at the histogramm, and mark a part which looks promising, for example a irregular part, or a part of a geometric figure which looks interesting. (Figure 4.2)
Note that empty space around the actual object also created entries on the history table. This part of the data is often located on the left-bottom part of the histogram. (Figure 4.3)
Sometimes even different areas on the histogramm might produce similar looking results. In this case mainly the gradient lengths are different, therefore the red transfer function element is rather describing an outer isosurface of the objects, whereas the blue one describes rather describes an isosurface just a little bit more inside. When rendering this with a raycaster, you might get problems when the step size is that big that there are pixels where the red transfer function element has been hit, and others where the blue one has been hit. Therefore it might be better to remove one of them or merge them. (Figure 4.4)

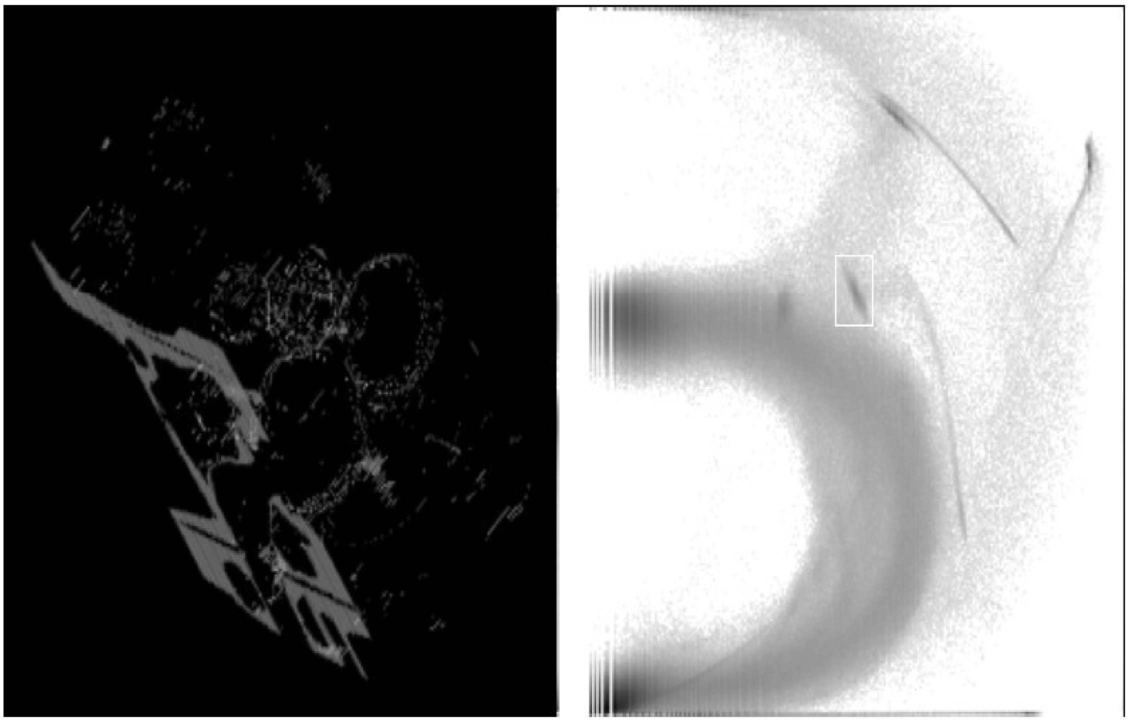


Figure 4.2: A maximum in the histogramm ... anything useful?

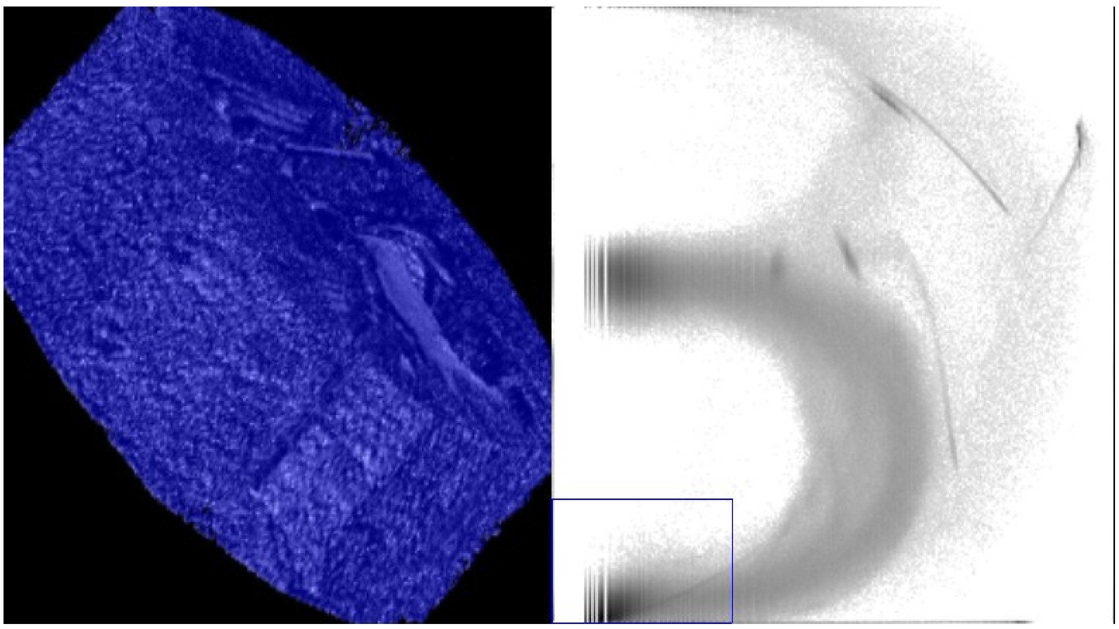


Figure 4.3: Having assigned an opaque material to the surrounding voxels.

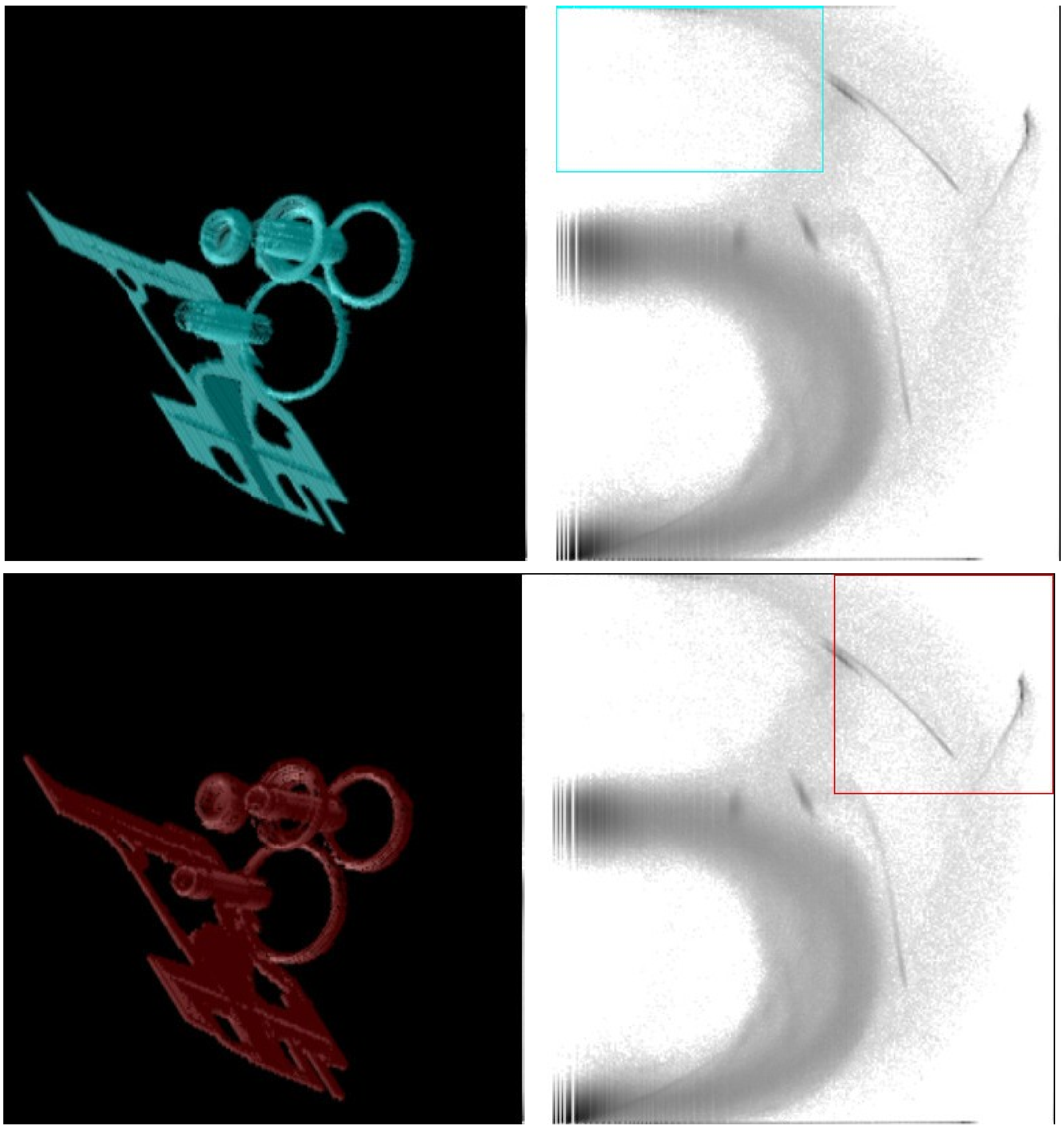
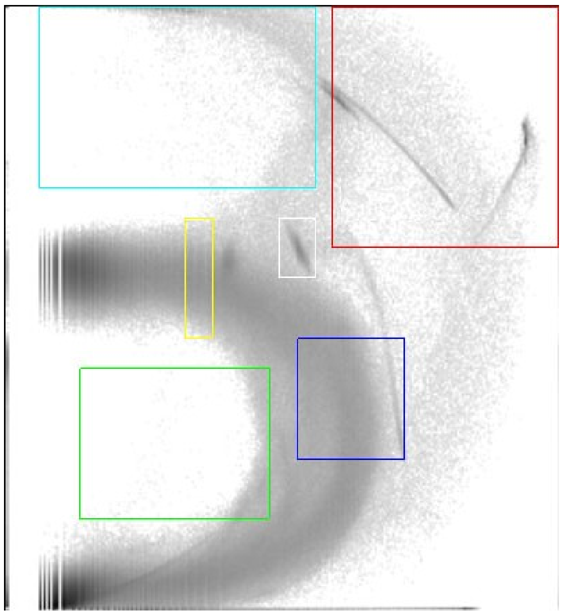


Figure 4.4: Different areas in the histogramm sometimes produce similar results.



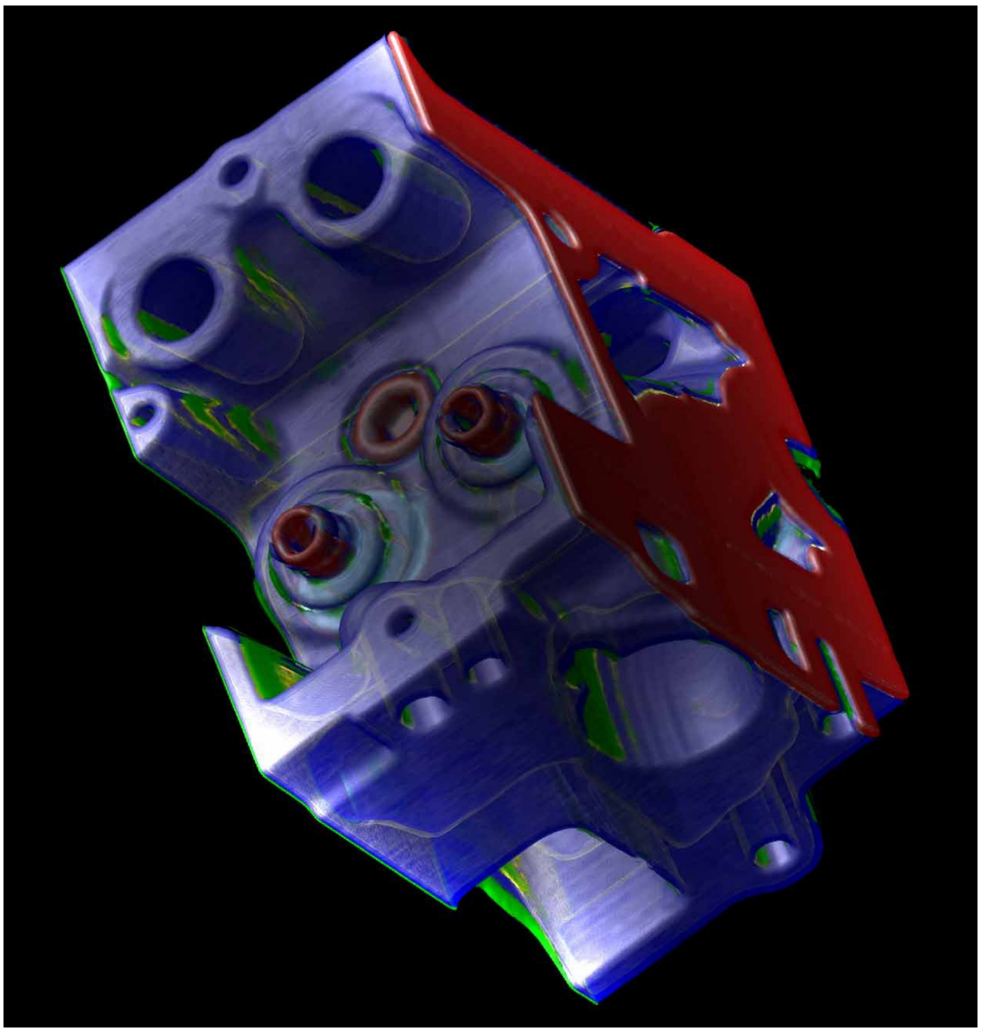


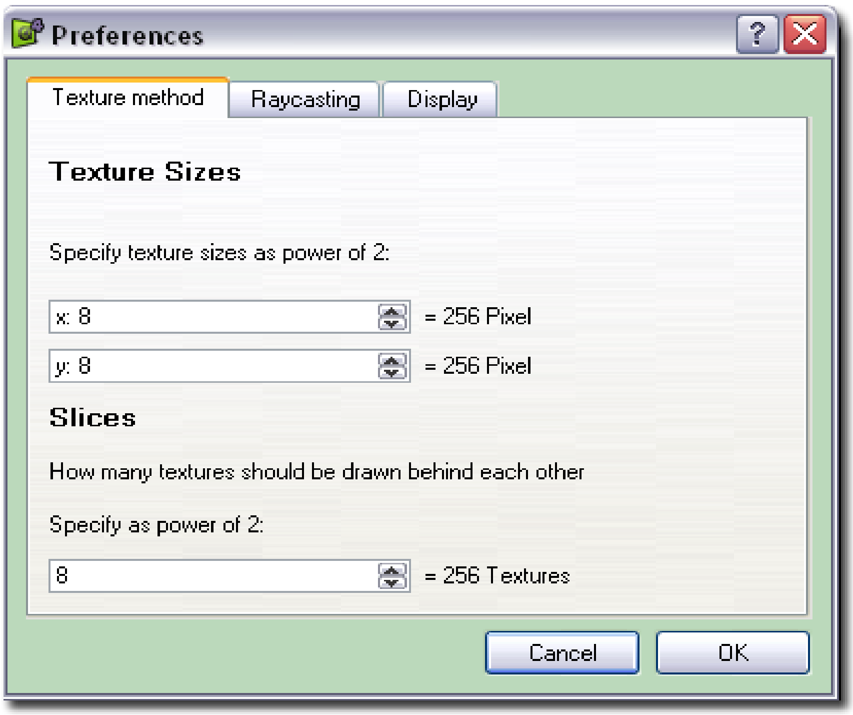
Figure 4.5: Result with some more transfer function elements. The blue surface is a bit transparent, you can look throught it. This image was rendered using the Raycasting2 renderer with a step size of 1/4.

5 Preferences

With "Edit" — "Preferences" in the main menu (see chapter 3.2) you can open the preferences dialog to configure vvis. Your settings are saved to a file called .vvis.conf. It is saved into your home directory.

5.1 Texture method

At "texture method" tab the texture method renderer can be configured.



Texture Sizes

Specify texture sizes as power of 2:

x: 8 = 256 Pixel

y: 8 = 256 Pixel

Slices

How many textures should be drawn behind each other

Specify as power of 2:

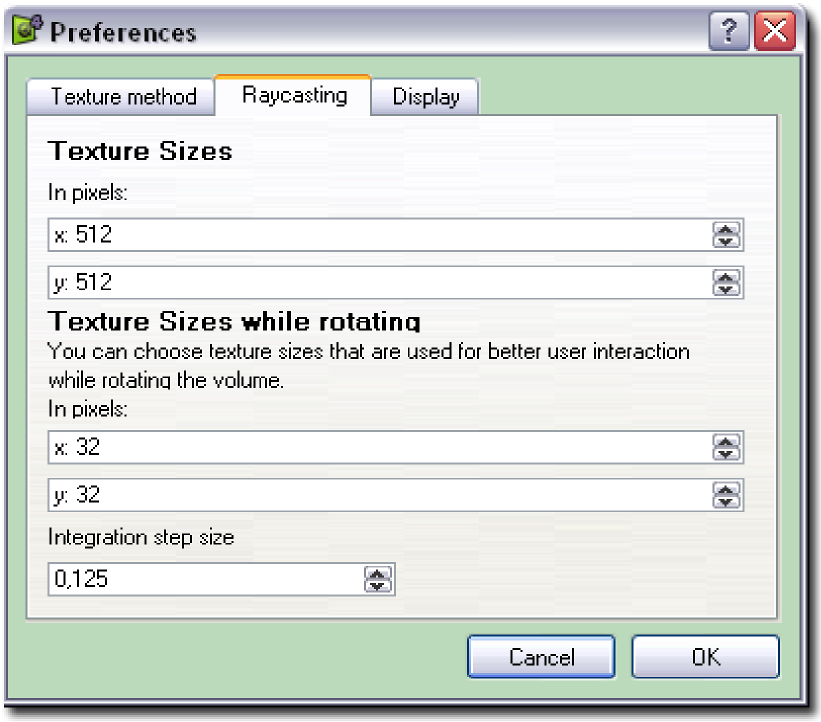
8 = 256 Textures

You can specify the texture size of the textures and the number of textures to be drawn behind each other. The texture method renderer requires both numbers to be a power of 2. Therefore you can specify the exponent to a base of 2. The resulting texture size respectivly slices is displayed right of the entry.

With greater texture sizes or more slices the quality gets better but the computation lasts longer. All the computation is done before displaying the first frame that is that you have to wait longer to see the volume but rotating and zooming is not effected.
Exceeding your graphic card's onboard memory will result in a drastic performance loss, so keep in mind how much memory one texture needs. All textures used here are RGBA textures, i.e. they need 4 byte for each texel, thus 256 256x256 textures need 64MB. The texture renderer often draws 2 stacks of textures, therefore this might be the upper limit for using the texture renderer on a 128MB graphics card.

5.2 Raycasting

At this tab the raycasting renderer (Raycasting 1, 2 and with pre-integration) can be configured.



Texture Sizes

In pixels:

x: 512

y: 512

Texture Sizes while rotating

You can choose texture sizes that are used for better user interaction while rotating the volume.

In pixels:

x: 32

y: 32

Integration step size

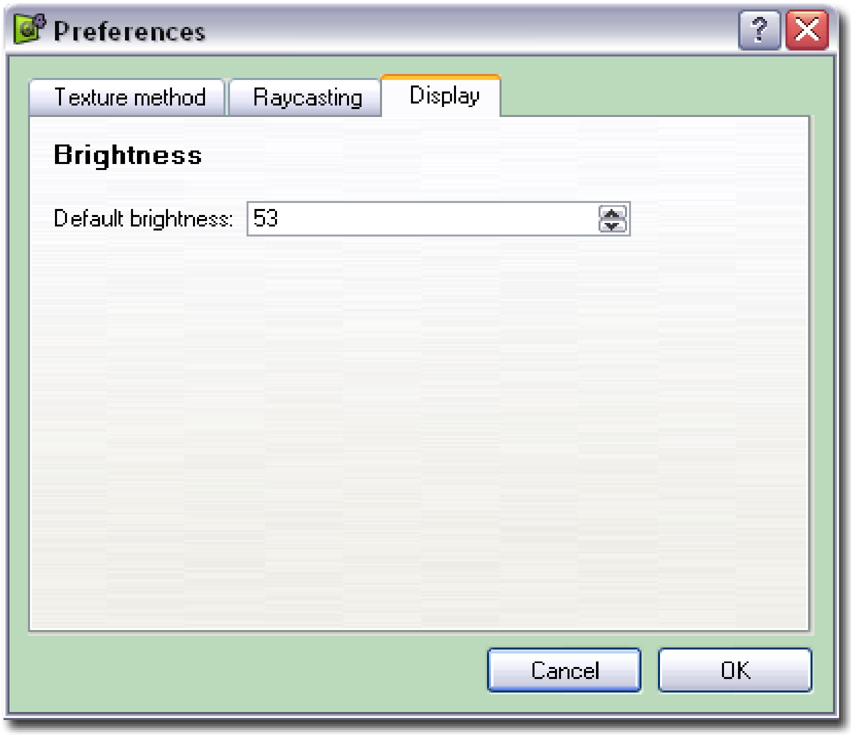
0,125

You can specify the texture sizes in pixels. The computation of the textures is done each frame. Therefore you can specify a lower texture size used while rotating the volume to get higher frame rates. During rotation the renderer is also in a preview mode, which disables any interpolation during the calculations to speed up the image generation.

The integration step size can be lowered for better image quality, for usual voxel data sets and transfer functions a step size of 1 should be ok. Bigger step sizes still work, but complete surfaces seen with a lower step size might have jagged edges or be broken down to several parts with the new configuration. Using a integration step size half as big as before will result in a twice as long calculation time.
An integration step size of 1 means that the step size is the same as the size of one voxel in the data set along the X-axis.

5.3 Display

At this tab display options can be configured.



Brightness

Default brightness: 53

Default brightness: You can specify the brightness that is used by default. It is used at vvis startup.

6 Volume Renderers

vvis provides different types of volume rendering methods. Each of them has advantages and disadvantages, therefore it is useful to know how they work. Summarizing one can say that the texture method and especially shear warp are the way to go when creating a new transfer function and you want to test how a new element added there affects the rendered images. The texture method provides a smoothly rotating view of the volume and the related emissivity values without caring much about absorbitivity and completely ignoring light properties. Shear Warp is a bit slower, but is capable of calculating light effects and absorbtivity, but the resolution is depending on the voxel data size.
The *raycasting renderers* can only provide interactive refresh rates at low resolutions like 64x64 on current hardware. But they are capable of creating images with high resolution and random sizes.
The data provided throught the transfer function is fully used with exception of the pre-integrated volume renderer.
In all renderers, we are often in between the positions of different density values. In such cases the density values are trilinear interpolated, except in the preview mode of the raycasting renderers which gain their main speedup by not interpolating the voxel data. The preview mode is enabled when you hold down the mouse button and rotate the volume.

6.1 Texture Renderer

The Texture Renderer produces images of the volume data by creating a stack of RGBA textures. Those textures are then drawn on several planes. The colors used in the texture are defined by the emissivity of the transfer function, as well as the transparency. Since you won’t see much of this texture stack from positions which are near to the planes the textures are drawn onto, a total of 3 stacks, for each viewing direction one, are created. Two of them are displayed; their selection depends on the viewing direction. The stack whose planes are the most perpendicular to the viewing direction is not displayed. The order in which the textures are drawn is also depending on the direction of view.

6.1.1 Pre-Integrated Texture Renderer

Between two planes which are mapped onto the textures, there might be imporant information, especially if we have a transfer function with steep slopes which would require high resolution sampling. For each viewing direction along the coordinate axis one could integrate between the texture planes using trinlinear interpolation and store the information on the textures. But since a table with the pre-integrated transfer function already existed, this was a test if this might also improve the image quality of the textur renderer. For the lookup table one value on a plane and a value at the same position on the plane on the plane’s neighbour are taken and used for the table lookup (see pre-int raycasting 6.2.1).
The used assumption here is that we are looking almost perpendicular onto the planes. Using this lookup tables has the same disadvantages as described in the part about pre-integrated raycasting: you have to neglect the information about the gradient length when using the transfer function. To avoid this, the pre-int lookup table class provides an integration function which also interpolates between given gradient lengths, not only density values. This cannot be stored in a lookup table, this calculations have to be done when the textures are generated, thus there is no speedup as with the lookup table compared to the normal texture renderer during texture generation.

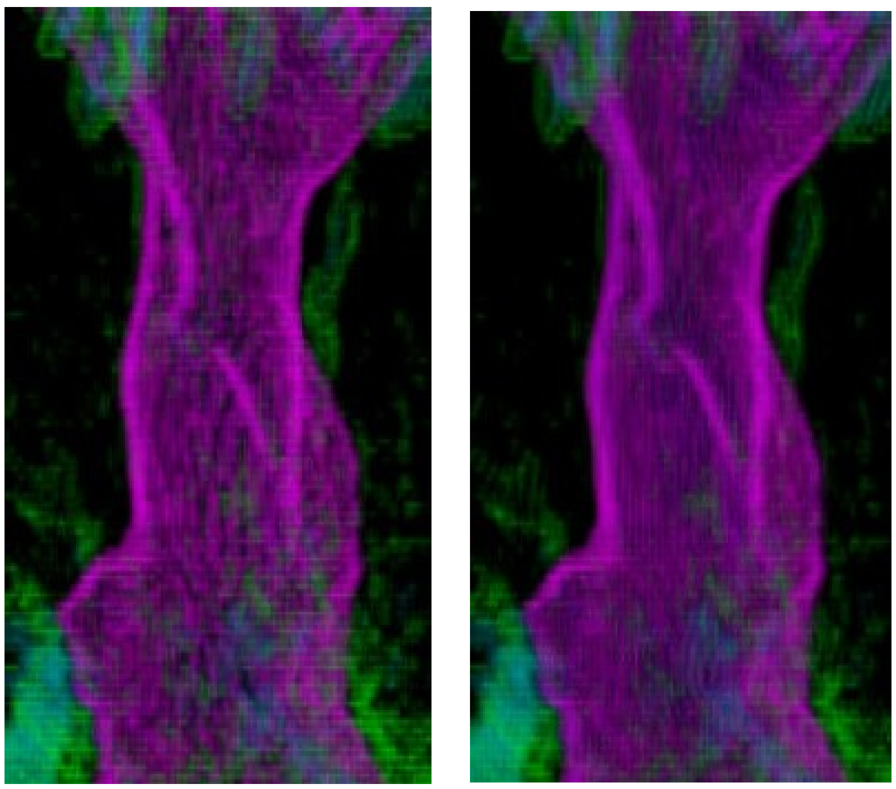


Figure 6.1: Example showing the difference between the normal texture renderer on the left and the integrating texture renderer also interpolating the gradient lengths on the right. One can see that the surface of the bonsai tree trunk is renderer more smoothly using the integration. This image was taken from a position 30 degree off a primary axis.

The function used in the pre-int class provides a method of calculating which is mainly a computation

of a mean value for the properties described by the transfer function. In this case this simplification even has an advantage since this computation does not depend on permutations of the start and end values. Therefore the same textures can be used for looking at the volume from opposite directions. The code of the renderer provides both methods, the slow one obeying gradient length information is currently used in the GUI.

6.2 Raycasting

Raycasting works by integrating a number of lines throught the volume. The intensity seen in the final result is dependent on the absorbtivity of the material assigned to the voxels, its emissivity and its behaviour when light is shed on it. All these properties are again defined by the transfer function. The integral looks like this for directional light specified by LightDirection and LightColor, a parametrization of our line x(§), a function to get the normal of the related isosurface from the voxel data normal(~x) and the results of the transfer function : Emissivity(~x), DiffuseColor(~x), SpecularColor(~x) and Glossiness(~x). By we mean component-wise multiplication.

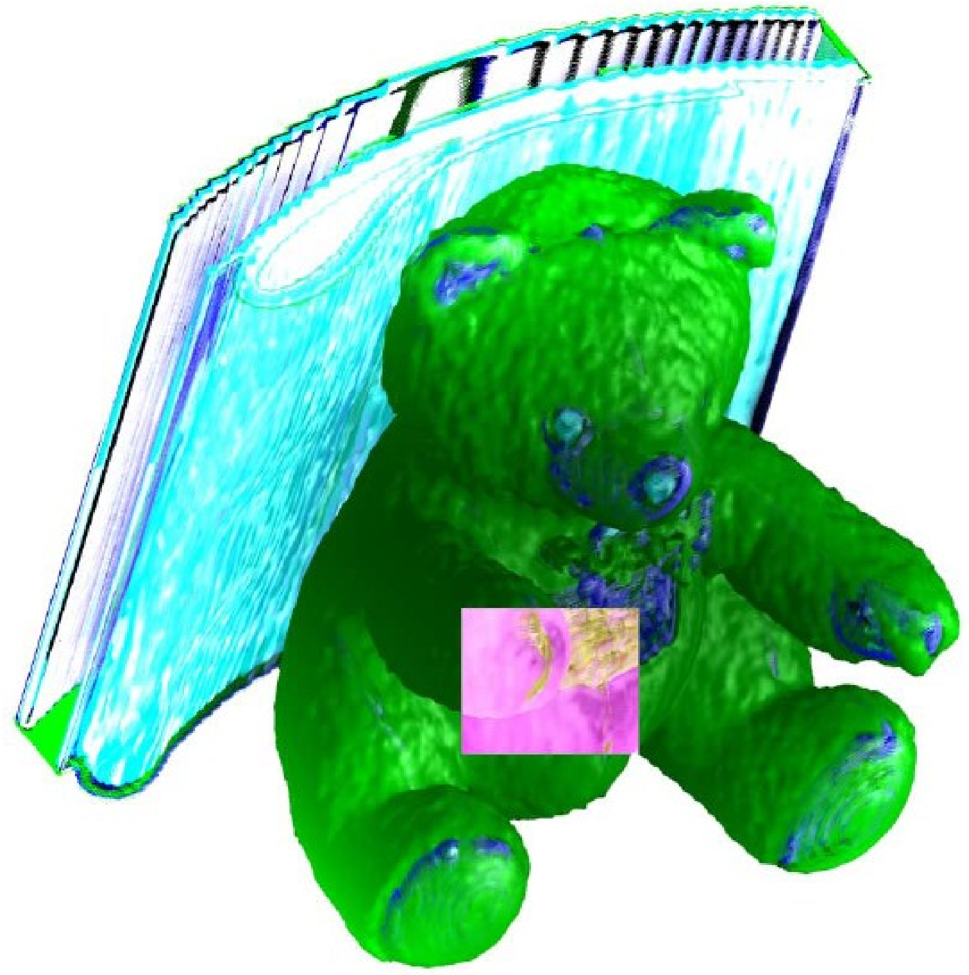


Figure 6.2: The following pictures about the Raycasting 1/2 Renderer are showing a part of the 'Teddybear' data set, which is marked inverse in this figure. This data set has an overall resolution of 128x128x62 Voxel. All time information correspond to a calculation on a Pentium4 2.66GHz, the output resolution at which the following images were rendered and used in this document is 640x480.

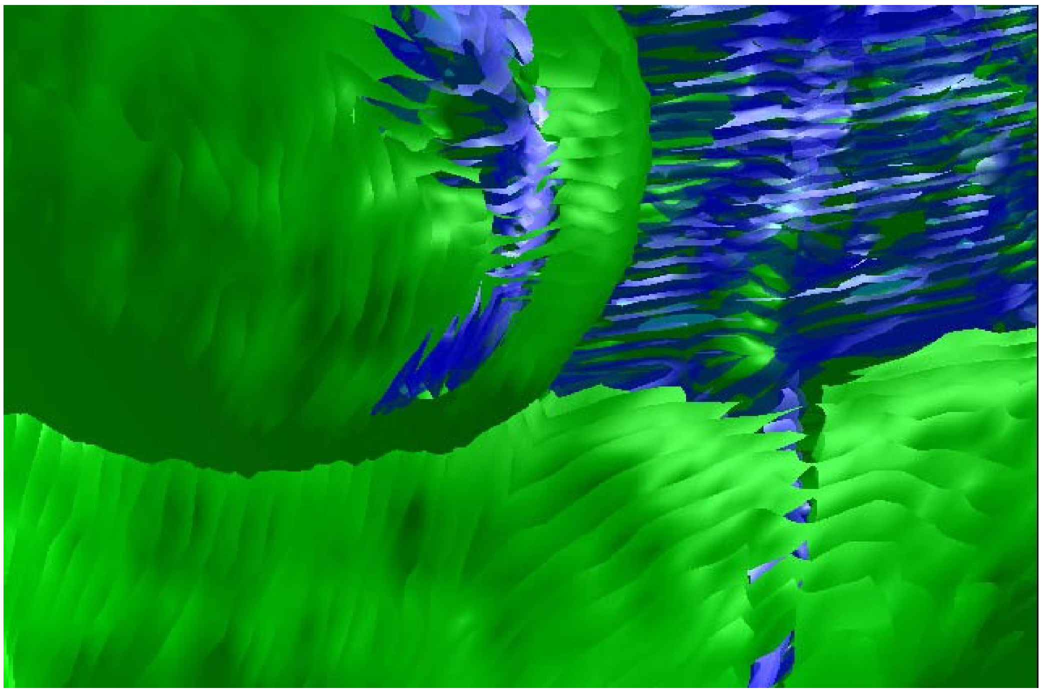


Figure 6.3: Image (see Figure 6.2) rendered using Raycasting 1 with an integration step size of 1. The calculation took 35s.

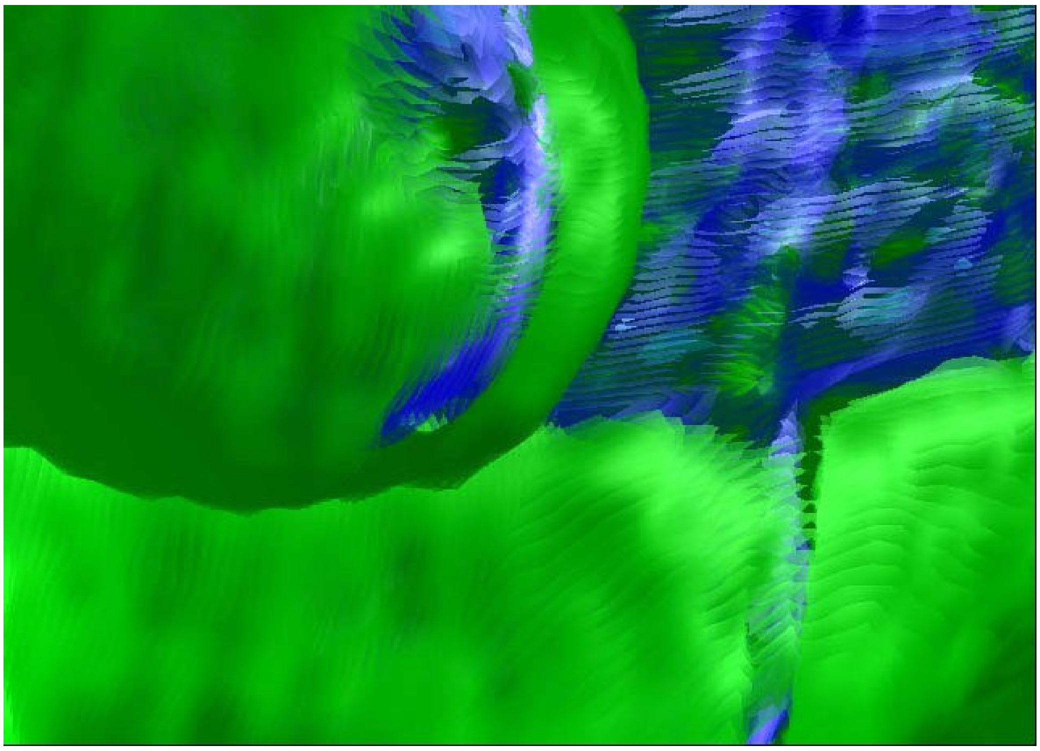


Figure 6.4 Image (see Figure 6.2) rendered using Raycasting 1 with an integration step size of 1/2. The calculation took 70s.

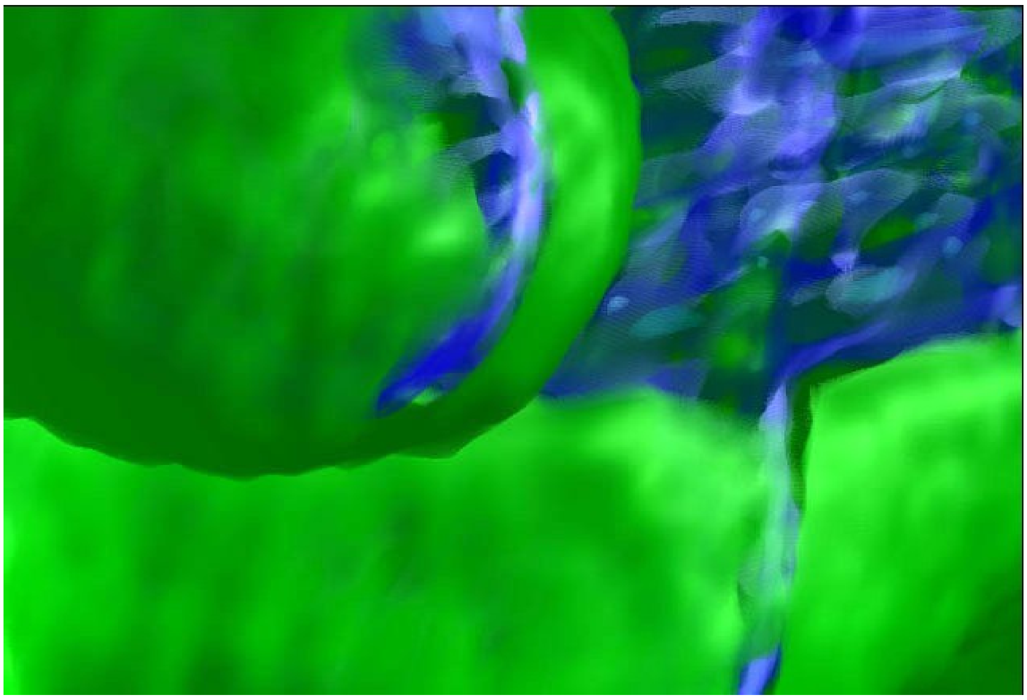


Figure 6.5: Image (see Figure 6.2) rendered using Raycasting 1 with an integration step size of 1/8. The calculation took 270s.

$$
\begin{aligned}
I &= \int_0^D \mathrm{color}(\vec{x}(\lambda)) \exp\Big(-\int_0^\lambda \mathrm{extinction}(\vec{x}(\mu))d\mu\Big)d\lambda && (6.1)\\
Diffuse(\vec{x}) &= (\mathrm{normal}(\vec{x}), \mathrm{LightDirection})\mathrm{LightColor} * \mathrm{DiffuseColor}(\vec{x}) && (6.2)\\
Specular(\vec{x}) &= (\mathrm{normal}(\vec{x}), \mathrm{LightDirection})^{\mathrm{Glossiness}(\vec{x})}\mathrm{LightColor} * \mathrm{SpecularColor}(\vec{x}) && (6.3)\\
color(\vec{x}) &= \mathrm{Emissivity}(\vec{x}) + \mathrm{Diffuse} + \mathrm{Specular} && (6.4)
\end{aligned}
$$

For actual calculation, we integrate from the back to the front with a specified step size. The exponential function is approximated by a first order taylor series. One step of the integration then looks like this, when starting with $I_{00} = 0$:

$$I'_i = color_i + (1 - extiction_i)I'_{i-1} \qquad (6.5)$$

6.2.1 Improvements

Pre-Integrated Integration The idea to use pre-integration came from a paper about using hardwareaccelerated pixel shading [3]. Their aim was to run the integration on a graphics chip and to avoid effects due to transfer functions which would need a very high sampling rate. The assumption here was that the density values between 2 subsequent points of our integration changes linearly. Thus all combinations for a given step size can be calculated before displaying the volume, during the rendering itself just a simple lookup has to be done, which can even be done with the limited capabilities of programmable graphics hardware.

The calculations currently used are the following, when i and j are the density values while s and t are the corresponding gradient lengths of the start and end point.

Lookup ignoring the gradient lengths (used in Pre-Int Raycaster) :

$$lookupEmissivity_{ij} = \int_0^1 \text{Emissivity}(\lambda i + (1-\lambda)j)d\lambda \tag{6.6}$$

Calculation using the gradient lengths (used Pre-Int Texture Renderer and the Raycaster 2) : ... and accordingly for the other data provided by the transfer function.

$$intEmissivity(i,j,s,t) = \int_0^1 \text{Emissivity}(\lambda i + (1-\lambda)j, \lambda s + (1-\lambda)t)d\lambda \tag{6.7}$$

This is just calculating the mean value. Also obeying absorption when calculating diffuse, specular and emissive material properties resulted in artefacts while rendering. The used code is still implemented as mode 5 and 6 in the PreIntegrator class.

In the paper mentioned above, they obviously used a one dimensional transfer function, which was only depending on density values[1]. Therefore they only needed to store a 2 dimensional field or a 3 dimensional field if different step sizes have to be available. Since our implementation also uses the gradient lengths for selecting a transfer function element, at least a 4 dimensional lookup table would have been necessary. Together with a sufficient quantization this table would have been really big, therefore we decided to see what happens if we create a table only relying on the density values[1]. This precalculated table is used in the Pre-Int Raycaster and can be used in the Pre-Int Texture Renderer.

A variant of this is the second raycaster. Here no table is precomputed, but similar calculations are done during rendering. For each density value inside the region defined by end and start value, the related transferfunction elements are used to calculate a mean value, also linearly interpolating the gradient lengths. This calculations are relatively fast and the result for similar rendering times look smoother for this method (compare figure 6.4 and 6.8).

1 We ran a test using a 2-dimensional hash table for the density values and then adding integrated values together with the gradient lengths of start and end point to lists. Two gradient lengths were said to be equal if their absolute difference was smaller than one, for testing purposes. During rendering there were a lot of hits in our 'cache', but this did not result in a performance boost, it slowed the rendering process even down by 10%. The main reason for this was probably the size of the stored data which easily reached 100MB when rendering a 1024x768 image and therewith the bad CPU-cache usage for those operations. Only applying this caching for regions on the histogram with high densities and calculating other values directly anyway or resorting the used STL vectors so that more used data would be faster to access has not been tested since it did not looked promising.

6.3 Shear-Warp

The idea of the shear-warp algorithm is to factorize the view matrix into a shear and a warp transformation. A shear transformation of the original volume data is performed, then this data is projected perpendicular to the sheared planes onto the so called intermediate image. This intermediate image is then warped and we get the final output. (see figure 6.6) In contrast to the raycasting renderer, this is an object-based algorithm, since we loop through the elements of our volume. This is the reason that the output size of the shear warp renderer is directly coupled to the volume size. On the other side, in the image-based renderers like the raycaster, one pixel of the final image is calculated after another and within this process the related data from the volume is gathered. When performing the shear operation, the voxel data is not interpolated. We implemented this once, but we encoutered a lot of flickering when rotating, maybe one would have to use 'intermediate slices' and then use trilinear interpolation to be able to render thin parts of the volume correctly. This would then be rather a shear-warp variant of the raycasting algorithms, but the main intention for

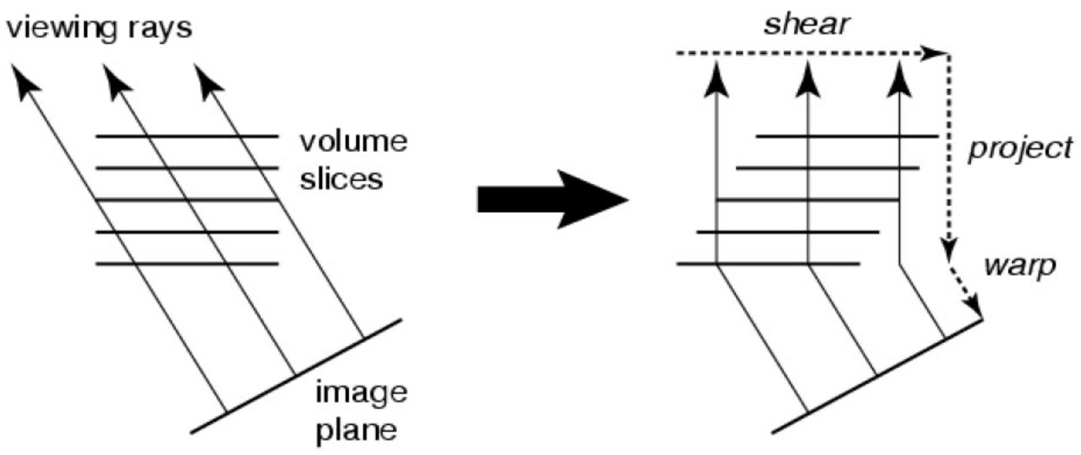


Figure 6.6: Idea of the Shear-Warp-Algorithm. (image taken from [1])

implementing this algorithm was to have a fast and sufficient-quality renderer for previewing the transfer functions.
The speedup of this algorithm is a result of the fact that complete axis-aligned voxel planes can be processed one after another. This is positive in terms of cache usage, but primarily it allows us creating run-length encoded tables, based on a volume which is already classified by a transfer function. The areas in which we only have transparent and non-emissive transfer function elements can be skipped during rendering. For the teddy image, for example, only 18% of the total voxels had to be processed, the rest was saved due to run-length encoding. Often the percentage of voxels not being transparent is even lower, you can look up the actual value in the log which is created when the renderer prepares the data.

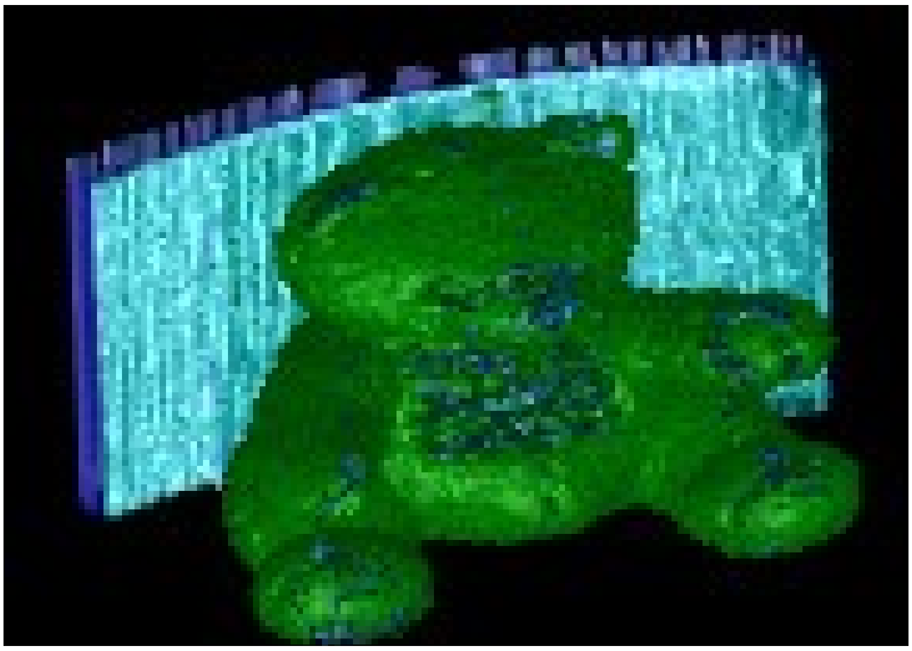


Figure 6.7: Output of Shear Warp Renderer. Calculation took 140ms on a Pentium4 2.66Ghz, the original image size was 256x256 pixel, this is a 147x106 pixel big part of it. Preparing the RLE tables took 420ms on the same system.

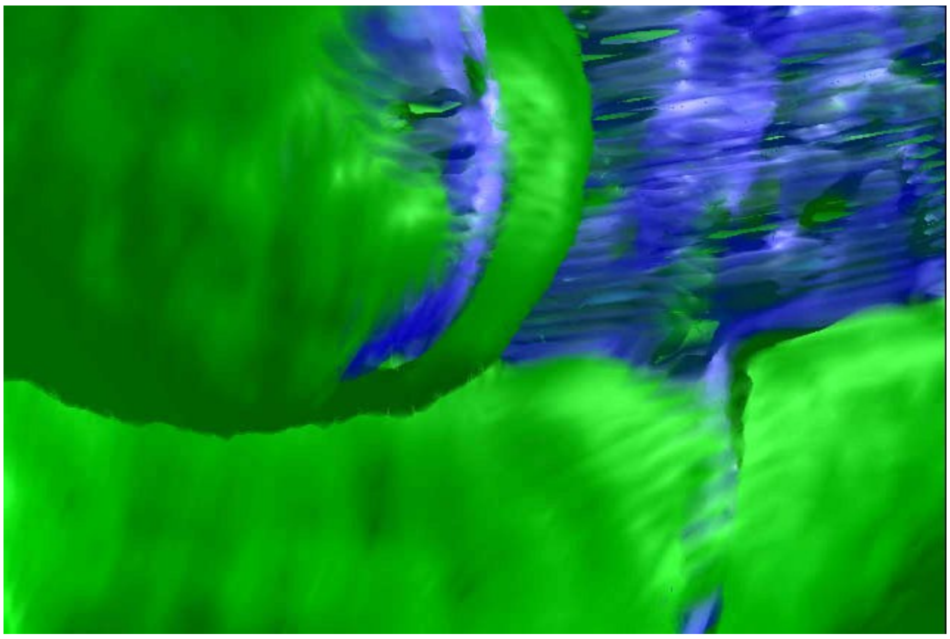


Figure 6.8: Image (see Figure 6.2) rendered using Raycasting 2 with an integration step size of 1.The calculation took 80s.

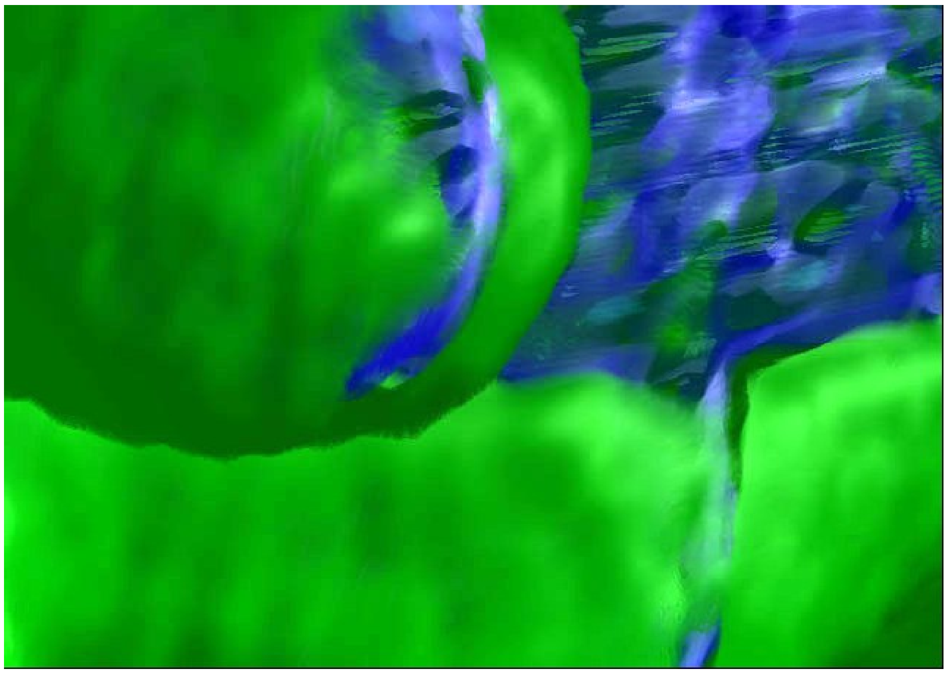


Figure 6.9: Image (see Figure 6.2) rendered using Raycasting 2 with an integration step size of 1/2. The calculation took 160s.

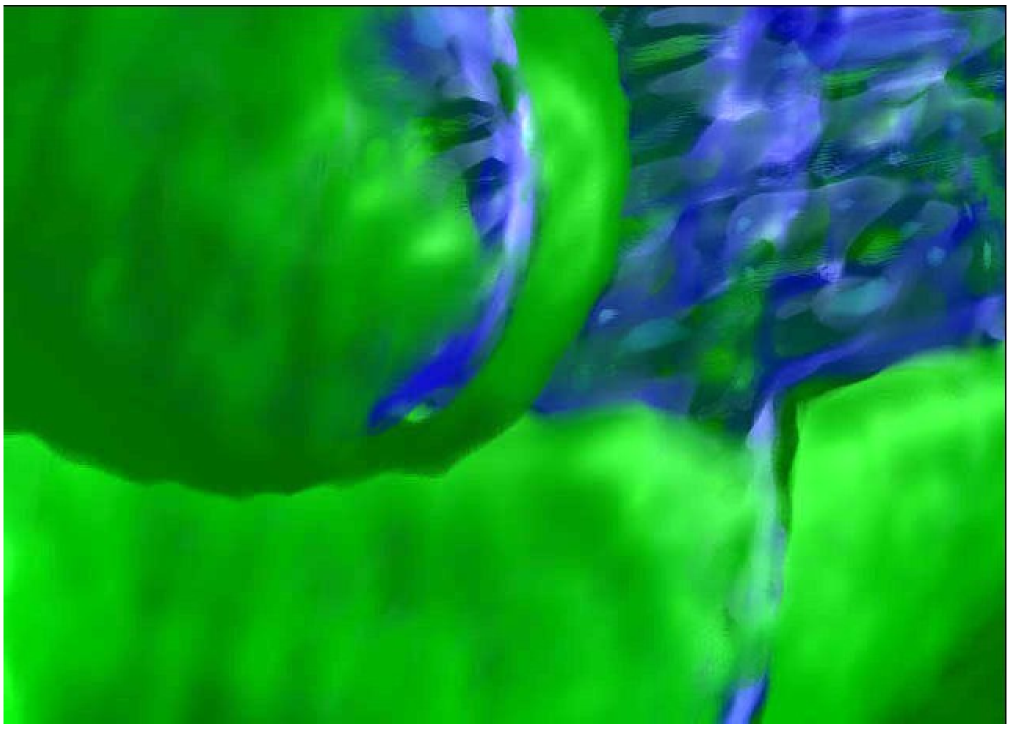


Figure 6.10: Image (see Figure 6.2) rendered using Raycasting 2 with an integration step size of 1/4. The calculation took 315s.

Bibliography

[1] Philippe Lacroute, Marc Levoy: Fast Volume Rendering Using a Shear-Warp Factorization of the Viewing Transformation
http://www-graphics.stanford.edu/papers/lacroute thesis/

[2] Jon Sweeney, Klaus Mueller: Shear-Warp Deluxe: The Shear-Warp Algorithm Revisited
http://www.cs.sunysb.edu/ mueller/papers/vissymFinal.pdf

[3] Klaus Engel, Martin Kraus, Thomas Ertl: High-Quality Pre-Integrated Volume rendering Using Hardware-Accelerated Pixel Shading
http://wwwvis.informatik.uni-stuttgart.de/ engel/pre-integrated/

[4] Christof Rezk-Salama, Klaus Engel, Fernando Vega Higuera: The OpenQVis Project
http://openqvis.sf.net/

[5] Various Authors : Slides for talks on Siggraph
http://www.cs.utah.edu/ jmk/sigg crs 02/courses 0067.html