Volume Rendering Assignment

(Anna Vilanova based on TU/e course by Michel Westenberg)

Please, read this document completely before starting. It is extensive but it will help you to progress better. Also have a look at the evaluation sheet.
In this assignment, you will develop a volume renderer based on the raycasting approach as described during the lectures. This will allow you to fully understand how direct volume rendering exactly works.
The skeleton code is in Java and provides a set-up of the whole graphics pipeline, a reader for volumetric data, and a slice-based renderer. The main task is to extend this to a full-fledged raycaster based on the provided code skeleton.

1 Getting the skeleton code running

We provide skeleton code to help you get started on the assignments. The code is in Java, and provided as Netbeans projects. It should be possible to run the project out-of-the-box on Mac OS X, Linux, and Windows systems. For OpenGL graphics, the code relies on the JOGL libraries, which are included in the archive. Netbeans also provides means to debug the code, breakpoints and watching variables, have a look at this possibilities.

You will need algorithmic and programming skills for this project, however, you do not need deep knowledge of Java given that we have provided the basic skeleton. If needed you can have a look at http://www.javatpoint.com/simple-program-of-java.
You need a basic knowledge of object oriented programing (what is a class, and what are the functions in a class). The syntax should be similar to most other programing languages you might have been using before, but you might need some time to get used to it.

2 Skeleton code

The skeleton has the familiar set-up of a main application (VolVisApplication), which holds the visualization and user interface. For this assignment you do not need to go through all the files and classes, but you should understand the ones that are mentioned in this document and that will contain the code you need to use and the code you need to extend.
The helper classes Volume and VolumeIO represent volumetric data and a data reader (AVS field files having extension .fld), respectively.
The class VolumeIO, you do not need to understand the code, you will just use it to read the files.
The class Volume stores the volume and contains functions to retrieve values of the volume. It is an important class where you will also need to implement some of the methods for retrieval and interpolation.

The class GradientVolume is partially implemented (i.e., you need to implement parts of it). It stores and provides methods to compute and retrieve gradient vectors. You will need to implement several methods in this class.
In the module GUI, you have the interface classes like TrackballInteractor provides a virtual trackball to perform 3-D rotations. To facilitate working with 1-D and 2-D transfer functions, the classes TransferFunction, TransferFunction2D, TransferFunctionEditor, and

TransferFunction2DEditor provide methods to store and graphically edit transfer functions. Unless you want to implement an extension, you do not need to get deep in the user interface classes in the GUI module. You will basically use them to get access to the user designed transfer functions.

From the util module, the class VectorMath offers a number of static methods for vector computations that you can use for your computations.

The class RaycastRenderer provides a simple renderer that currently generates a view-plane aligned slice through the center of the volume data (see Fig. 1). The renderer also draws a bounding box to give visual feedback of the orientation of the data set. This class is where most of the raycasting algorithms will take place and the class that you will need to update and implement

3 Assignment

3.1 Part I: Ray casting

To start with study the method slicer() in the class RaycastRenderer, we will use this class as a basis to develop a raycaster. Try to understand what is happening in the function.

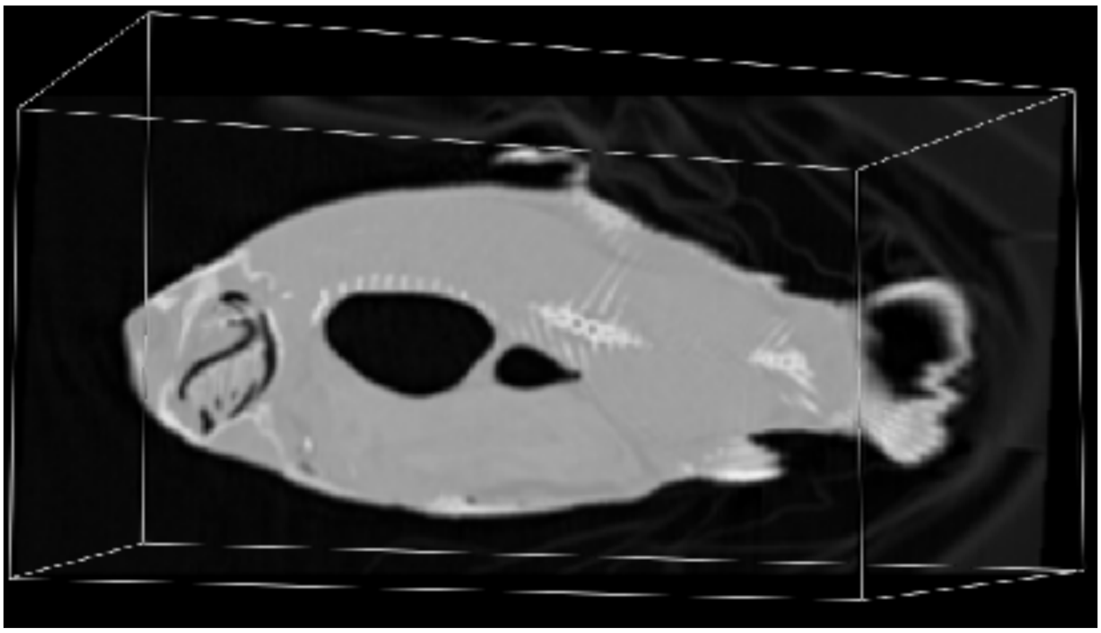
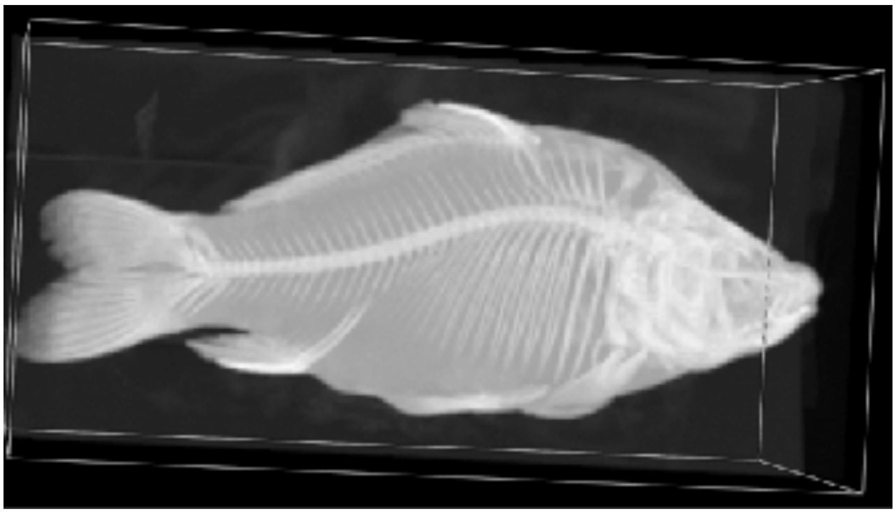


Figure 1 *Visualization of carp8 using the slicer. Current visualization of the framework.*

Study the function raycast to see how the different types of visualizations can be activated and the initial direction and position of the ray are calculated .

Furthermore, the code already provides a transfer function editor so you can more easily specify colors and opacities to be used by the compositing ray function. Look at the function slicer() and activate the Transfer Function (TF) in order to understand how it is used.

Figure 2 shows result images for the *Carp8* dataset based on the MIP ray function (Fig. 2 (top)) and the compositing ray function using the default settings of the transfer function editor (Fig. 2 (bottom)). We also provide some data sets that you can use. We recommend to use *Carp8* for testing given its small size.



Implement the following functionalities:

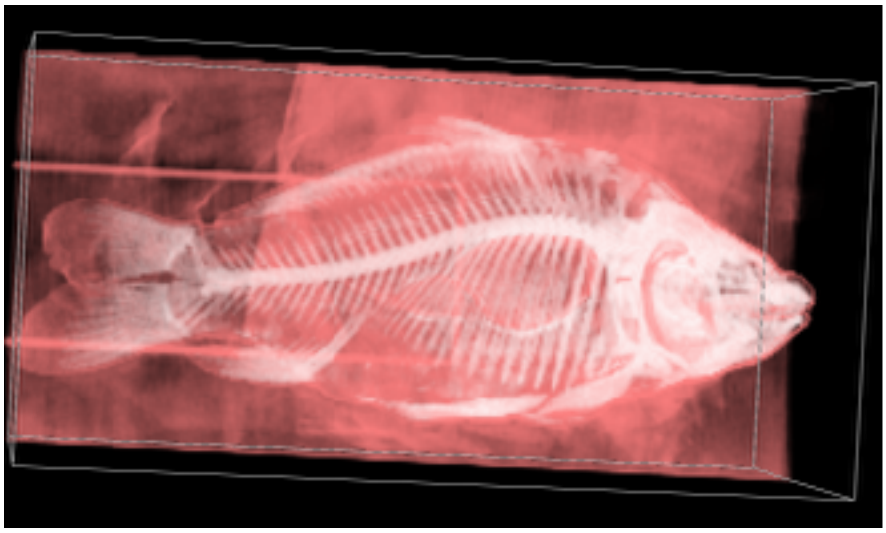


Figure 2 (top) *MIP of the* carp8 dataset. (bottom*) Result of the compositing ray function on the* carp8 *dataset with the default settings in the transfer function editor.*

1. Tri-linear interpolation in the class Volume (function getVoxelLinearInterpolate ()) is needed for the samples along the viewing ray at the moment Nearest Neighbor is implemented in function getVoxelNN(). This function is also used in slicer() you can see the effects already there. Implement getVoxelLinearInterpolate().

2. MIP (function traceRayMIP in the RaycastRenderer) – Result for the Carp should be as shown in Figure 2 (Top)

3. compositing ray functions (function traceRayComposite in the RaycasRenderer) and create the necessary functions within RaycastRenderer. Result for the Carp 8 using the default transfer function should be as shown in Figure 2 (bottom)

4. As you may notice, a software raycaster is quite slow. Make the application more responsive during user interaction. A straightforward way to do this is to compute the

image at a lower resolution. You can think about other strategies. (**hint** you can use the interactiveMode attribute of the renderer)

Experiment with various data sets, try to highlight interesting features in these datasets, and compare pros and cons of MIP and compositing.

3.2 Part II: 2-D transfer functions and Shading

As discussed in the lectures, implementation of illumination (e.g., Phong model) is needed to be able to distinguish shape clearly. Furthermore, simple intensity-based transfer functions do not allow to highlight all features in datasets. By incorporating gradient information in the transfer functions, you gain additional possibilities. For this section we will also use Levoy's paper[2].

Below are the requirements:

1. Implement gradient-based opacity weighting in your raycaster, as described in Levoy's paper [2] in the section entitled "Isovalue contour surfaces", eq. (3). The code already contains a 2-D transfer function editor which shows an intensity vs. gradient magnitude plot, and provides a simple triangle widget to specify the parameters. The class GradientVolume should be extended so that it actually computes the gradients. You need to implement function compute I the GradientVolume and the function related to linear interpolation.
   You also need to extent the computation of the transfer function based on the triangle information provided by Transferfunction2D *tFunc2D*.

2. Implement an illumination model, e.g., Phong shading as discussed in the lectures. An example result with and without illumination is shown in Fig. 3. The Phong shading parameters in this example are $k_{\text{ambient}} = k_{\text{a}} = 0.1$, $k_{\text{diffuse}} = k_{\text{d}} = 0.7$, $k_{\text{specular}} = k_{\text{s}} = 0.2$, and $\alpha = \text{n} = 10$.

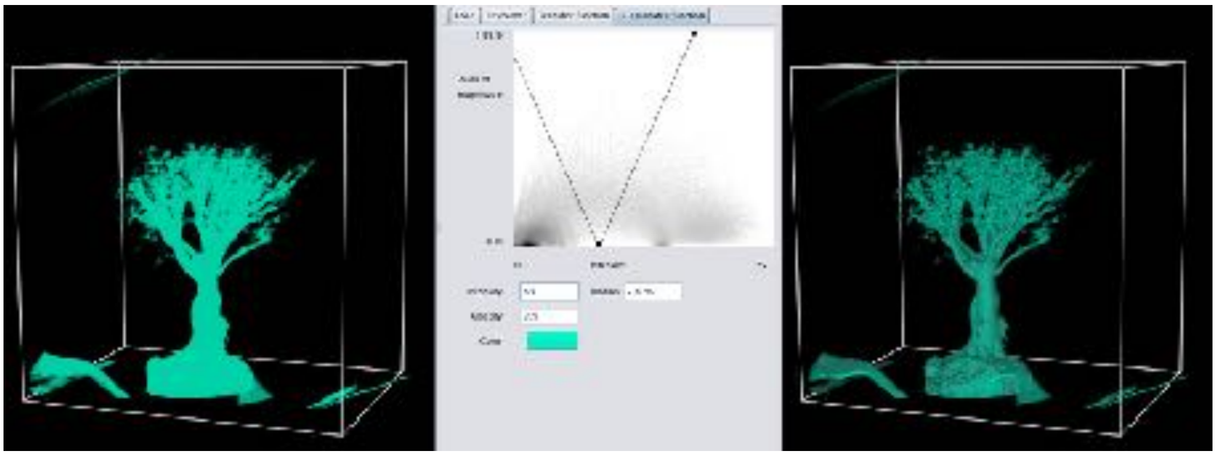


Figure 3: *Gradient-based opacity weighting result on the bonsai data. The image on the left is without illumination.*

Experiment with various data sets and compare the results with those obtained by the approach in the previous exercises.

3.3 Part III: Extend the basic volume render

Now you have created a basic volume renderer. You can implement an extension and here you can chose which extension you would like to use.

A possible extension is to extend the basic triangle widget by the approach described by Kniss et al. [1], which introduces extra parameters to control the range of gradient magnitudes that are taken into account in the computation of the transfer function. This refers specifically to section 4.5, you would need, however, to read the whole paper.

You are welcome to add other extensions. A possibility is to add illustrative methods like silhouette, or tone shading. You can also think about adding shadows or accelerations. You can also be creative and think about your own extension.

We provide some data sets that you can use, but feel free to search for more (notice that we read a specific format so this would mean you need to build a converter or reader).

- https://www.cg.tuwien.ac.at/xmas/#Download%20data Xmas tree.
- http://medvis.org/datasets medical data sets.
- http://www.sci.utah.edu/download/IV3DData.html

4 Deliverables

This is a group project but you all should be involved and understand all components of the project.

- **Source code** with comments. We should be able to understand your code to be able to evaluate it, so make sure that the code is well structured, variable names have meaning, and that you have comments indicating what you are doing for each function.
- **Report** indicating:
  - What has exactly been implemented. Describe in a theoretical level and in your own words, what you implemented and where this implementation can be found in the source code (i.e., what exact class and function where it is implemented). Do not repeat extensive theory, just say what you implemented exactly.
  - The report should include results and evaluation where you analyze the functionality of the parts that you implemented.
  - Include a results section and analyze different volumetric data sets of your choice with the tool you have developed. Show features that would be interesting to visualize from the data, show images and explain how you achieved the visualization, e.g., indicate the transfer function. Comment on

the usability of your tool and the different components(i.e., times, interaction, effectiveness, etc.) . Notice that often is useful to show the transfer function.

  - **The Report** should be a maximum of **3 pages A4 size** of **text**. The pages should have similar font size and margins as in this project description document. The text has to be to the point. So 2 well written clear pages are better than 3 pages with the extra redundant content. Images will not be counted as part of the 3 pages so include as many pictures as you consider adequate. Illustrating what you can achieve is important.

- **Individual reports:** there should be a max. 300 words document for each of the group members describing what you did individually in the project. The idea is that **ALL** of you are involved and responsible for all components of the project. You **ALL** should be able to explain any part of the code. It can be that one focuses in one part, and explains it to the others, but at the end each of you is responsible for the whole project. So ideally the individual report is that you all worked together in all parts in an equal manner.
  Partial projects will be evaluated equally independent on the amount of members that have developed it. Just in the evaluation of the extension and the results presentation the amount of group members will be considered.
  It is important that we know that your partners also agrees with your individual reports, therefore, submit it together with the rest of the project. If you work in all aspects of the project together, which is desirable, just indicate that shortly. You do not need to extend yourself unnecessarily.

Bibliography

1. Joe Kniss, G. L. Kindlmann, and C. D. Hansen. Multidimensional transfer functions for interactive volume rendering. *IEEE Trans. Visualization and Computer Graphics*, 8(3): 270–285, 2002.
2. M. Levoy. Display of surfaces from volume data. *IEEE Computer Graphics and Applications*, 8(3):29–37, 1988.