Chapter 1

Volume rendering

In this assignment, you will develop a volume renderer based on a raycasting approach. The skeleton code 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.

1.1 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.

1.2 Skeleton code

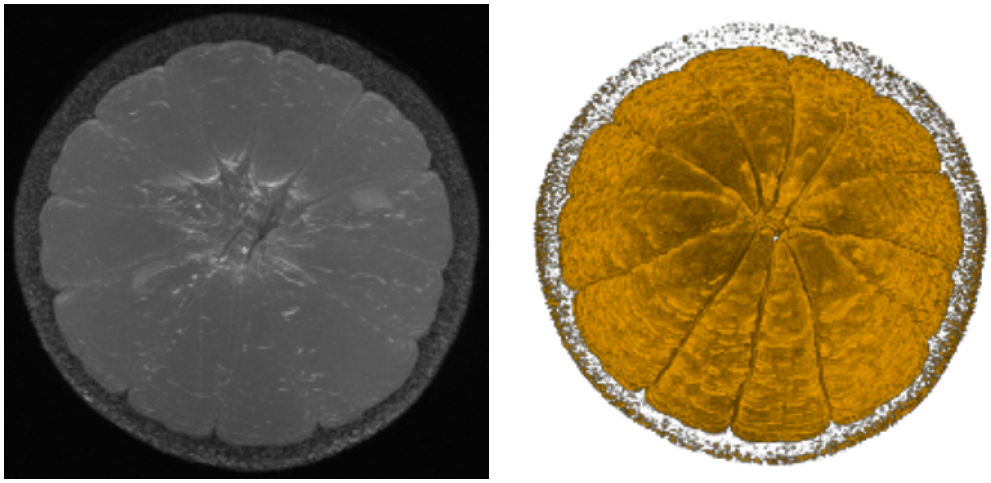
The skeleton has the familiar set-up of a main application (`VolVisApplication`), which holds the visualization and user interface. The helper classes `Volume` and `VolumeIO` represent volumetric data and a data reader (AVS field files having extension `.fld`), respectively. The class `GradientVolume` is partially implemented, and provides methods to compute (not implemented!) and retrieve gradient vectors. To make working with 1-D and 2-D transfer functions easier, the classes `TransferFunction`, `TransferFunctionEditor`, and `TransferFunction2DEditor` provide methods to store and graphically edit transfer functions. The class `VectorMath` offers a number of static methods for vector computations. Finally, `TrackballInteractor` provides a virtual trackball to perform 3-D rotations.

The class `RaycastRenderer` provides a simple renderer that visualizes a view-plane aligned slice through the center of the volume data. The renderer also draws a bounding box to give visual feedback of the orientation of the data set.

1.3 Assignments

1.3.1 Ray casting

Study the method `slicer()` in the class `RaycastRenderer`, and use this as a basis to develop a raycaster that supports both *Maximum Intensity Projection* and *compositing* ray functions. The coordinate system of the view matrix is explained in the code. Furthermore, the code already provides a transfer function editor so you can more easily specify colors and opacities to be



(a) (b)

Figure 1.1: *(a) MIP of the orange dataset. (b) Result of the compositing ray function on the orange dataset with the default settings in the transfer function editor.*

used by the compositing ray function. Figure 1.1 shows reference result images for the orange dataset based on the MIP ray function (Fig. 1.1(a)) and the compositing ray function using the default settings of the transfer function editor (Fig. 1.1(b)).

Implement the following functionalities:

1. Tri-linear interpolation of samples along the viewing ray.
2. MIP and compositing ray functions.

As you may notice, a software raycaster is quite slow. You can easily make the application a bit more responsive by lowering the resolution during rendering (both in number of steps along the view ray as in the number of pixels you sample). The `RaycastRenderer` already has a variable `interactiveMode` which is set to `true` when you interact with the mouse, and is set to `false` when interaction stops.

1.3.2 2-D transfer functions

As discussed in the lectures, 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.

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.
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. 1.2. The Phong shading parameters in this example are $k_{\text{ambient}} = 0.1$, $k_{\text{diff}} = 0.7$, $k_{\text{spec}} = 0.2$, and $\alpha = 10$.

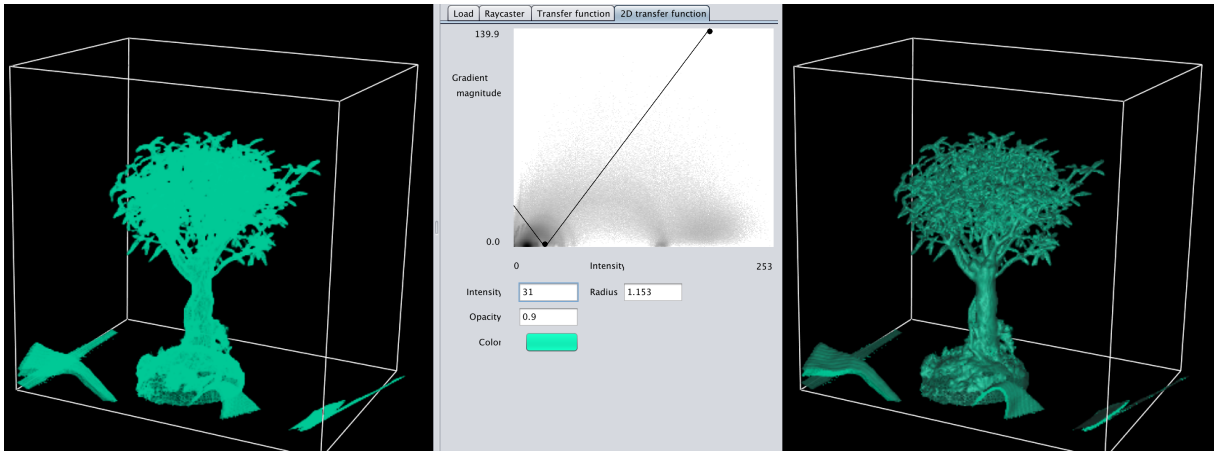


Figure 1.2: *Gradient-based opacity weighting result on the bonsai data. The image on the left is without illumination.*

1.3.3 Evaluation

Experiment with various data sets and produce a number of visualizations that show interesting features of these data sets. Demonstrate that your MIP, 1-D transfer functions, and 2-D transfer functions work as intended.

1.3.4 Apply your volume renderer to a real use case

The Scientific Visualization community runs a yearly visualization challenge, see `http://sciviscontest.ieeevis.org` for an overview.

The 2018 edition features simulation data of deep water asteroid impacts. The contest page can be found at `https://sciviscontest2018.org`, and it has an extensive description of the data and its purposes. As the full dataset is much too large to handle, we have prepared a subset of the data, containing only a few time points and only the scalar fields `v02` (volume fraction water), `v03` (volume fraction of asteroid), `tev` (temperature in electronvolt), and `prs` (pressure in microbars).

Your assignment is to produce an insightful visualization of (some of) this data. The document that describes the dataset provides quite some inspiration.

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.