IN4089 - Data Visualization - Project 2: Volume Visualization

Summary

Introduction: In this project, you will implement a number of algorithms used in volume visualization as a group project; (1.) tri-linear interpolation, (2.) first hit direct volume rendering (DVR) (iso-surface rendering), (3.) Phong shading, (4.) DVR with compositing and 1D transfer function, and (5.) DVR with compositing and 2D transfer function. You should avoid distributing the algorithms to individual members and instead work on each algorithm together. All group members should understand each algorithm. Additionally, you will implement at least one extension of your choice on an individual basis.

Learning Objectives: The goal of this project is to assess to what degree you have mastered LO5: Implement visualization systems for a given practical data analysis problem.

Purpose: The project is designed to practice the practical aspects of volume visualization. You will implement volume visualization algorithms discussed in class. The project contributes 35% to your final grade.

Resources: On Brightspace, you can find the volume visualization framework providing all basic functionality from loading the data to setting up the main rendering views. Alongside the framework, we provide documentation to get you started installing the framework and describing the existing functionality. The framework is implemented in C++. You will need some algorithmic and programming skills for this project (e.g., knowledge of object oriented programming - what is a class, and what are the functions in a class). However, as large parts of the code is provided in the framework, you do not need deep knowledge of C++. The syntax should be similar to most other programming languages that you might have used before.

Submission

Submit your deliverables to Brightspace. For the group parts use the **VolVis Project [group]** assignment. For the individual part use **VolVis Extension [individual]**. The deadline for the final project and all corresponding deliverables is **Jan. 28, 2022**.

Deliverables

1. **Basic Implementations** [group], use 'VolVis Project [group]' assignment
    - **complete code** needed to run your solution in a zip file **(No binaries or provided data)**.
    - **a single pdf** containing up to two screenshots for each implemented feature (Interpolation, Isosurface, Shading, Compositing 1D TF,Compositing 2D TF), showing a result with a dataset of your choice that show good examples of the specific properties of the implemented feature. Briefly justify **why the images are a good example** for the method in no more than **100 words** per feature. Use the provided template/example.
2. **Extension** [individual], use 'VolVis Extension [individual]' assignment
    - **complete code** needed to run your extension(s) in a zip file (No binaries).
    - **a single pdf** containing for each implemented extension (minimum one) up to two screenshots of the result using a dataset of your choice that show good examples of the specific properties of the implemented extension. Briefly **explain the extension (max 200 words)** and justify **why the images are a good example** for the extension **(max 100 words)**. Use the provided template/example.

Note: **all limits are hard limits**, everything above the limits will not be graded.

Assesment Criteria

1. **Basic Implementations (50%)**. Each of the five above listed algorithms/parts will contribute 10%.
2. **Extension (50%)**

Both will be graded based on the **correctness and quality** of your code (i.e., the code works as expected, is efficient, readable, and well documented), as well as your **selected images and justification** for those.

Instructions

Implement your solutions within the provided framework. Document your code and add 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 a meaning, and that you have comments indicating what you are doing for each function you implemented. Before submission, **make sure that the integrity test completely succeeds.**

WARNING: While the project offers you freedom in your choices and you can **add** additional classes and functionality, please make sure that the **original code structure remains intact!** Your code will be tested partially via an automatized solution in order to detect the functions that have been implemented correctly and those that contain mistakes. Hence, you are not allowed to modify the existing function headers. Classes can only extend the original classes but need to implement the original functions as outlined in this document.

1. Tri-linear interpolation

In the Volume class (`volume/volume.cpp`) implement tri-linear interpolation. Different interpolation functions are called from `getSample(...)`, depending on what is selected in the GUI. In the base framework, only nearest neighbor interpolation is implemented in `getSampleNN(...)`.

- Start with the function `getSampleTriLinearInterpolate(...)` which has the continuous 3D coordinate as a parameter.
- Before we worry about the interpolation itself, make sure that the incoming coordinate is within the volume bounds. These are defined in the 3D vector `m_dim`.
- As discussed in the lecture, the 3D interpolation can be split up into a set of 2D and 1D interpolations. Fill in the functions `biLinearInterpolateXY(...)` and `linearInterpolate(...)` to implement these 2D and 1D interpolations, respectively and make use of these functions.
- Hint: for your screenshots you can use any of the existing render modes or those that you will implement. Consider comparing nearest neighbor and linear interpolation and pick the mode where you can show the differences best.

2. Iso-surface Raycasting

Implement isosurface raycasting in the Renderer class (`render/volume.cpp`). This method will show the surface defined by an isovalue (see lectures). The GUI already contains the necessary widgets to set the isovalue. You can retrieve it from the configuration object, connected to the GUI as `m_config.isoValue`.

- Implement the ray traversal for basic iso-surface raycasting the `traceRayISO(...)` function. Look at traceRayMIP for an example of how to step through the ray. As discussed in the lecture, this will only produce a single colored silhouette (example with default parameters and the carp8 dataset is shown in Figure 1).

Figure 1: Result of the isosurface raycasting on carp8 data set with given default values.

- Before continuing with making the iso-contour more precise in the next step, implement Phong shading (next section) to actually see the difference.
- Isosurface raycasting can be achieved more accurately, if an extra step is added, in which we search for a more precise position of the isovalue. Implement the bisection algorithm (in `bisectionAccuracy(...)`) for the isosurface raycasting.

3. Phong Shading

Shading/illumination (e.g., using the Phong model) is needed to be able to distinguish shapes clearly—as exemplified by the example in Figure 1 without shading.

- For shading we need normals, or in volume rendering gradients. The GradientVolume class `volume/gradient_volu` computes the gradients in the function `computeGradientVolume(...)` for each voxel position. The `getGradient(...)` function allows multiple interpolation modes, similar to `getSample(...)`. Implement linear interpolation for the gradient vectors.
- Implement the Phong shading model (in `computePhongShading(...)` in the `Renderer` class) as discussed in the lectures and include it in the isosurface raycasting method.
- go back to the previous section and implement `bisectionAccuracy(...)`.

For comparison, example results with shading are shown in Figure 2. The Phong shading parameters in this example are $k_a = 0.1$, $k_d = 0.7$, $k_s = 0.2$, and $\alpha = 100$.

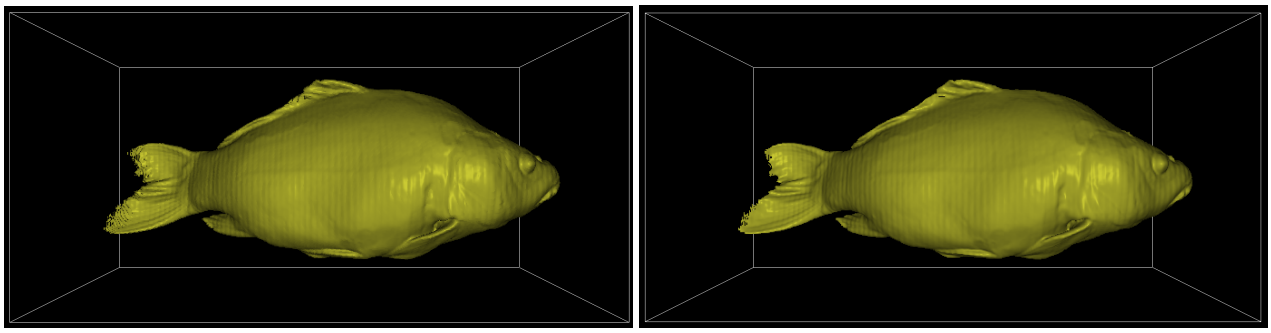

Figure 2: Result of the isosurface raycasting using Phong shading. Left: standard, right: with increased accuracy.

4. Compositing with 1D Transfer function

Implement compositing raycasting functions to be able to look inside the data and composit multiple materials.

- Implement the ray traversal for composite raycasting using a 1D transfer function in `traceRayComposite(...)`. The result for the Carp8 using the default transfer function should be as shown in Fig. 3.
- Add Phong shading to the compositing function (using your already implemented `computePhongShading(...)` method).
- Hint: play with the transfer function and different data sets to create example images that really highlight the strengths of compositing.

Figure 3: Result of the composite raycasting on carp8 data set with given default values.

5. Compositing with 2D Transfer function

Simple intensity-based transfer functions do not allow you to highlight all features in datasets. Here, you will implement compositing using a 2D transfer function based on intensity and gradient magnitude. Look at the approach defined by Kniss et al. [1] on how to use the gradient information.
The code already contains a 2D transfer function editor in the last tab in the GUI, which shows an intensity vs. gradient magnitude histogram in the background, and provides a simple triangle widget to specify the parameters. We use triangular widgets, which are particularly effective for visualizing surfaces in scalar data.

- Implement the ray traversal for composite raycasting using the 2D transfer function in `traceRayTF2D(...)`.
- For the provided 2D TF widget you have a constant rgb-color that you can get from `m_config.TF2DColor`.
- You need to compute the opacity based on the intensity and gradient magnitude, though. Implement this in `getTF2DOpacity(...)` and call this function during your ray traversal. Hint: Start with constant values inside the triangle. After that set the values on the vertical line through the apex of the triangle to an opacity of 1 and from there towards the diagonal borders fall off to an opacity of zero by creating a linear transition that is aligned horizontally.

Figure 4 shows an example for 2D transfer function based compositing. Note how the surfaces of internal structures like the air bladder become clearly visible.

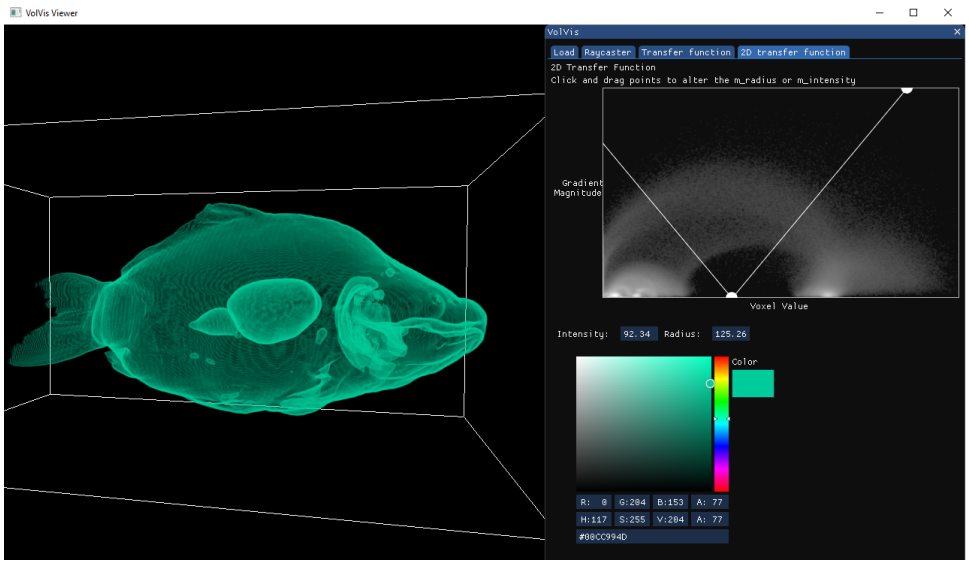

Figure 4: Result of the composite raycasting on carp8 data set with a 2D transfer function.

Individual: Extensions

For your individual grade, implement at least one extension. Examples for possible extensions are given below, but others are possible. You should submit the corresponding code in a separate submission so feel free to ignore any warnings about breaking existing functionality or the automatic grading here.

Example Extensions.

- We have discussed cubic interpolation in class. Function stubs are already available, so you could implement it in those.
- More advanced 2D transfer functions are possible, examples are the extra parameters (e.g., 2nd order derivatives) presented by Kniss et al. [1] or completely custom widgets presented in the same paper.
- Rheingans and Ebert [2] present some illustrative methods that could be added.
- Gooch et al. [3] present cool to warm shading for surface rendering.
- You could add shadows by using secondary rays.
- Be creative and think about your own extension.

References

[1] J. Kniss, G. Kindlmann, and C. Hansen, "Multidimensional transfer functions for interactive volume rendering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 8, no. 3, pp. 270–285, 2002.

[2] P. Rheingans and D. Ebert, "Volume illustration: nonphotorealistic rendering of volume models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 7, no. 3, pp. 253–264, 2001.

[3] A. Gooch, B. Gooch, P. Shirley, and E. Cohen, "A non-photorealistic lighting model for automatic technical illustration," pp. 447–452, 1998.