TNM093 2023 — SciVis Mini-Project

Project examiner/assistants: Alexander Bock, Emma Broman, Emma Nilsson

1 Intended Learning Outcomes

The goal of this mini-project is to provide an improved and practical familiarity with the concept of volume rendering. This includes understanding of the raycasting steps, the application of composition methods, the generation of transfer functions, and the ability to analyse a dataset interactively. These techniques should be implemented in a framework that does not require much overhead and is easily extensible to be able to focus on these individual goals, rather than requiring the development of an end-to-end solution.

The student shall, after finishing this mini-project, be able to:

- qualitatively understand the visualization technique of direct volume rendering,
- in detail understand three of the composition techniques (front-to-back, first hit point, maximum intensity projection) and possess the ability to reason about interesting new composition methods
- comprehend the concept of a transfer function in volume exploration and have the ability to design and implement an editor for manipulating the transfer function interactively to aid exploration
- be able to implement these techniques with web technologies such as JavaScript, WebGL2, and, for the transfer function, using any other libraries as determined by the student.

2 The Environment

For this mini-project, a simple JavaScript + WebGL environment is provided that should be used as the starting point for the implementation. The framework, including the data, can be found in the Lisam course page or at http://www.itn.liu.se/ alebo68/lectures/tnm093/2020/scivis-miniproject.zip. The code alone can also be found as a GitHub template at https://github.com/alexanderbock/TNM093.

The environment contains code that creates a WebGL window on the webpage, loads a dataset, renders the entry/exit bounding boxes, and initiates the volume rendering. It uses a simple, predefined transfer function to show initial results. As it requires WebGL2, the browser that you are using to view the webpage has to support the WebGL2 feature. This should be the case for most modern browsers, but if in doubt, check https://caniuse.com/#feat=webgl2 for the different version when WebGL2 was supported for the different browser vendors. Particularly Safari on MacOS does *not* implement this feature and we'd recommend Firefox or Chrome on that operating system.

The environment has a number of rendering parameter that can be changed. The step size between samples along the different rays can be changed interactively, the compositing methods can be changed using radio buttons, the camera can be moved around the volume, and the output rendering can be changed for debugging purposes and can show the volume rendering, the entry or exit points, the ray direction, the currently employed transfer function, or show a single slice of the volume.

2.1 Hosting the page

The environment requires two external files to be served locally, which means that just opening the webpage will *not* work and only produce an empty white rendering and an error message in the browser log. You will need to host a webserver and then go to the address of that webserver to see the rendering and work with it.

We recommend using the Live Server extension for Visual Studio Code for hosting your files. Then, you can simply press the "Go Live" button in the editor and the webpage will be hosted on whatever port is set per default. Alternatively, you can use Python for a simple webhosting service by executing: `python -m http.server`[1] in the folder where the `index.html`, `gl-matrix.js`, and `pig.raw` files are located. This directory will then be served and be accessible by visiting http://localhost:8000 on your machine.

3 The Tasks

- Familiarize yourself with the provided framework and conceptually understand everything that is happening in the initialization and the rendering loop before starting to implement anything.
- Implement the front-to-back compositing, first hit points, and maximum intensity projection compositing methods in the volume rendering `traverseRay` function.
- Using whichever techniques you want, implement a way to interactively update the transfer function to change the results of the volume rendering by manipulating the way the volume samples are mapped to color + transparency. It is recommended to stick to either a widget-based or node-based transfer function editor as shown in the lecture and not implement multiple methods. See the checklist below for a more detailed list of requirements. The transfer function is updated in the `updateTransferFunction` function. **Important**: Don't forget to call the `triggerTransferFunctionUpdate` function after anything has changed in your transfer function editor or otherwise the rendering will not be notified of your new transfer function.

The following is a Todo list that needs to be completed with a lab assistant to have successfully passed the lab. **Important**: The amount of time required for this mini-project is *much* more than what is allotted in the lab sessions. There are 8 h of lab sessions per group, but the mini-project is scheduled for 48 h of work in total. If you only implement things while being in the lab itself, you will *not* be able to finish the project.

Please note: If some time has passed between finishing the project and showing it to the lab assistants, take some time to refamiliarize yourself with the code you have written before calling them over. "I don't remember, this has been a while" is *not* a good answer to any of the questions below.

- ☐ Read the *entire* instructions carefully
- ☐ Interact with the camera controls and explain what the step size does and how volume rendering works and how you get a color for a pixel. For this explanation you can assume that you already have an entry and an exit point, there is no need to explain anything prior to that. Do this on a piece of paper by drawing a sketch.
- ☐ Show that the front-to-back compositing method works, show the implementation, and conceptually explain how it works
- ☐ Show that the first hit-point compositing method works, show the implementation, and explain how this compositing method works
- ☐ Show that the maximum intensity projection compositing method works, show the implementation, and explain how this works
- ☐ Using the front-to-back compositing method, use your transfer function editor to interactively manipulate the transfer function to show the lab assisstant what is on the inside of the dataset. Your editor must be capable of the following:

[1] Note that this is the command for Python 3. For earlier Python versions, the command is `python -m SimpleHTTPServer`

- ☐ Change the transfer function without needing to reload the webpage
- ☐ Have the ability to change the color and opacity mapping for volume samples, to any color
- ☐ Be able to interact with the editor and change the transfer function settings using the mouse
- ☐ Have the ability to simulate isosurface rendering by being able to create "spiky" transfer functions that have a high opacity only for a few values with different colors
- ☐ Apply smooth (e.g. linear) transitions between opacity values
- ☐ Be able to individually highlight at least three different materials in the volume, with different colors and opacity

☐ Write a one-page summary documenting your findings while interacting with your transfer function widget, what you have learned from this lab, and what findings were unexpected. Add the descriptions for the front-to-back, first hit-point, and maximum intensity projection composition methods. Add a short reflection about the situations in which to prefer InfoVis and SciVis visualization methods.