Introduction

This laboratory exercise will introduce Scientific Visualization and Information Visualization methods in a single environment. At the end of this lab, you will have hands-on experience with Raycasting, a volume rendering technique (Part 1), collected data from a volume (Part 2), implemented a scatter plot (Part 3) and parallel coordinates plot (Part 4), two commonly used Information Visualization techniques. An information reduction technique will have been implemented in Part 4 as well as Brushing & Linking (Part 5).

The task in this lab consists of a multi-view linked system that displays a volume rendering together with plotted volume information. The plotted data consists of voxel information (intensity value, average intensity of all 3D neighbors, standard deviation of these values, and the gradient magnitude), which needs to be extracted from the volume first.

Important: The labs require many more hours than available in the scheduled lab sessions. It is paramount that you work on the assignments outside of sessions in order to be able to finish the sessions in time.

Documentation

This documentation is not meant as a comprehensive list of every necessary webpage, but rather as a starting point for your own literature research.

- Voreen Documentation. (http://voreen.org/doc/3.0.1/)


- GLSL Quick Reference Guide. (http://mew.cx/glsl_quickref.pdf)
Figure 1: The initial screen of VoreenVE.


**Part 0 - The Setup**

The lab will be done in the framework Voreen that was partially developed here at Linköping University and which is also in daily use in research. Functional units in Voreen, called *processors*, are grouped into *modules*. On the course homepage you can download a module called *tnm093* that contains stubs for the different classes that you will implement during the lab.

To start working on the labs, you have to follow these steps (applicable to the computers in the lab):

1. Copy the Voreen source code from `/usr/src/voreen-tnm093` into the `/tmp` directory.
2. Copy the *tnm093* module (either the one from the homepage or the one from a previous session) to `voreen-tnm093/modules`.
3. Run `qmake` from the `voreen-tnm093` folder to generate a Makefile.
4. Compile the source code using `make -j4` (`-j4` suggest to `make` to use all 4 available threads for compiling)
5. Start editing
6. Important: Before logging out, copy the `/tmp/voreen-tnm093/modules/tnm093` folder into your home directory. Every file in `tmp` will be deleted when a user logs out.

To compile Voreen for your own computer, have a look at the document at ([http://webstaff.itn.liu.se/~alebo68/courses/voreen/voreeninstructions.pdf](http://webstaff.itn.liu.se/~alebo68/courses/voreen/voreeninstructions.pdf)). Before compiling and running `qmake`, please change the `config.txt` to include the line `VRN_MODULES += tnm093` if it is not already present.
After compiling, you can run the graphical environment, which is called VoreenVE by calling `start_voreenve.sh` (for the lab computers) or executing `bin/VoreenVE` (from home). You will be able to see the initial screen VoreenVE as shown in Figure 1. You can ignore any warnings or errors that are related to missing XML documentation or font files. You can choose any workspace (a collection of Processors, connections, and values) from the Templates submenu to familiarize yourself with the program. Each processor has Properties that encapsulate member variables and provide automatic GUI elements so that you can change each of the values. In the default setup the list of properties appear to the right, when a processor is selected. Each processor has one or more ports that are used for information exchange between processors. They have varying types and allow downstream processors to access data generated by upstream data sources.

All the files that you will need to edit are in the folders `voreen/modules/tnm093/include` and `voreen/modules/tnm093/src`. Change the other files at your own risk.

The workspace that you will use for this lab is called `tnm093.vws` and is located in `voreen/modules/tnm093/workspaces`. The view that you will see consists of four parts, the top left will contain your rendering of a Parallel Coordinates Plot (PCP), the top right will contain your ScatterPlot, while the lower left shows a raycasting and the lower right a slice-based rendering of the example data set. That data set is a micro-CT scan of a Walnut, acquired at the European Institute for Molecular Imaging in Münster, Germany.

**Part 1 - Raycasting**

In this first part you will get experience with raycasting and modify a provided stub for a volume raycaster. The stub already provides an unshaded Maximum Intensity Projection (MIP) and it is your task to to enhance this to Phong-shaded front-to-back alpha compositing. As the Phong-shading requires gradients, it will be necessary to implement a gradient calculation callend central differences.

In the provided workspace, you can find a processor `TNMRaycaster`, which you will modify to implement the necessary techniques. The raycaster has a property called Compositing that can be switched between front-to-back alpha compositing (DVR) and MIP. The Transfer Function property is used to change the transfer function and change the visual appearance of the rendering. The transfer function is available in the shader as a one dimensional texture.

**Task 1:**
Familiarize yourself with the working of a raycaster and have a look at the GLSL shader at `voreen/modules/tnm093/glsl/rc_raycaster.frag`. Then implement front-to-back alpha compositing in the `rc_raycaster.frag` shader.

**Task 2:**
Implement the method stub `calculateGradient(vec3)` to return the gradient at the provided sampling position using central differences.

**Hint:** Take care of border cases and perform sensible measures to handle these.

**Hint:** \( h = \text{volumeStruct.}\_\text{datasetDimensionsRCP.}\_x|y|z \)

**Task 3:**
Implement the Phong-shading technique into the method stub `applyPhongShading(vec3, vec3, vec3, vec3, vec3)`. 

3
Figure 2: One exemplary result for the front-to-back alpha compositing. Please note that your results might vary slightly.

Figure 2 shows a possible result. A different transfer function can lead to completely different visual outcomes.

Before you log out (even if it is just for a second) copy the voreen-tnm093/modules/tnm093 folder to your home drive or it will be permanently deleted!

**Part 2 - Data Collection**

In this part you will edit a processor (voreen/modules/tnm093/src/tnmvolumeinformation.cpp) that analyzes a volumetric data set and extracts information for each voxel of the volume. It is this information that you will show and render in the following two parts.

The structure is as follows: In a volume, each voxel has a unique identifier, which is determined by its position in the volume and the dimensions of the volume. The `TNMVolumeInformation` processor has one input port (inport), which is the volume that is to be analyzed, and one output port (outport) that contains the analyzed data. The analyzed data is in the form of a vector of structs:

```cpp
struct DataItem {
    unsigned int voxelIndex;
    std::vector<float> data;
};
typedef std::vector<DataItem> Data;
```

The `TNMVolumeInformation` processor generates one `Data` object that is transmitted to the plot-generating processors. The `index` member is the identifier of the voxel to which the `data` belongs.
Task 4:
Edit the TNMVolumeInformation processor to extract pieces of data for each voxel and provide them in the Data outport.
Generate (at least) the following measures:
1. The intensity of the voxel.
2. The average taken of all 26 neighboring voxels and the voxel in question.
3. The standard deviation of these values.
4. The gradient magnitude of the gradient which was calculated using any gradient scheme (forward, central, or backward calculation).

Hint: The intensity value of a voxel is available through the VolumeUInt16 object's voxel(int) method, which takes the index of the voxel and returns a 16bit integer value. Hint: Border cases have to be taken into account for the averaging and gradient calculation. Hint: The VolumeInformation processor already resizes the data vector to the correct size. Not push_back is necessary.

Before you log out (even if it is just for a second) copy the voreen-tnm093/modules/tnm093 folder to your home drive or it will be permanently deleted!

Part 3 - ScatterPlot

In this task, you will implement a ScatterPlot by editing the TNMScatterPlot processor at voreen/modules/tnm093/src/tnmscatterplot.cpp.

In the scatter plot, the user chooses two axis and all the data values are shown as markers (for example, dots) on the resulting 2D plane. Figure 3 shows an example of a scatterplot.

Task 5:
Edit the TNMScatterPlot to implement a scatter plot visualization. Perform the rendering operations in the process method, in which, for all intents and purposes you can assume that a valid OpenGL context is present.

Hint: Each axis in the canvas has the coordinates [-1,1] with the origin in the lower left. Hint: glBegin, glEnd, glVertex2f
Part 4 - Parallel Coordinates Plot

Here, you will implement a parallel coordinates plot visualization, which shows all of the multidimensional data at once. Whereas in a scatter plot two axes are selected and the data values are represented as markers, parallel coordinates plots show all axes at once and data values are represented as lines, connecting the axis with the intersection points denoting the value of that dimension (see Figure 4 for an example rendering).

One important feature of parallel coordinates is to limit the range of each dimension as a method of filtering out data values. This is usually implemented as a doubleslider that can be moved up and down each axis.

**Task 6:**
Implement a parallel coordinates visualization in the TNMParallelCoordinates processor at voreen/modules/tnm093/src/tnm_parallelcoordinates.cpp. Stub methods have been added for rendering the data values (renderLines) that need to me implemented. Finish the rendering before going over to the next task of enable interaction.

The interaction handling is performed in the methods handleMouseClick and handleMouseMove. It is up to you how you want to use these methods, but dragging a handle up and down should change the position along the axis and filter the values. The handles are of the class TNMParallelCoordinates::AxisHandle.

**Task 7:**
Create pairs of AxisHandles in the constructor, assign them unique identifiers and the correct positions, and add them to the _handles vector.
Use these identifiers in the interaction handling methods to change only that handle, which is currently manipulated. The id variable returns the identifier of the handle that is underneath the mouse cursor.

The details of the interaction are up to you, but it should include the following possibilities:

Before you log out (even if it is just for a second) copy the voreen-tnm093/modules/tnm093 folder to your home drive or it will be permanently deleted!
• Clicking and dragging a handle moves that handle and the lines that are rendered are filtered based on the position of the handles.

• The top handle should never be below the bottom handle and vice versa.

**Task 8:**
Implement interaction with the parallel coordinates visualization. It should be possible to drag the handles along the axes and the only those lines should be rendered that lie between the handles of all axes (i.e., as soon as a value on any axis is smaller than the value of the lower handle or bigger than the value of the upper handle, it is not rendered).

**Before you log out** (even if it is just for a second) copy the voreen-tnm093/modules/tnm093 folder to your home drive or it will be permanently deleted!

**Part 5 – Data Reduction**

In many cases it is not feasible to display all of the collected data, as it clutters the display and reduces readability. One way of solving this problem is to discard parts of the collected data. In this task, you will edit the processor TNMDataReduction and implement one data reduction technique based on random sampling.

The processor has one `Data` inport with the unfiltered data and one `Data` outport, where the filtered data is stored. While removing data values, take care that the same line is not removed twice.

You can add the processor into the workflow by rearranging the arrows that are connecting the VolumeInformation to the TNMParallelCoordinates and TNMScatterPlot processors.

**Task 9:**
Implement the TNMDataReduction processor. In the `process` method, you have access to the `_percentage` property, that should be used to determine how many percent of the data values should be removed.

**Changelog**

**11/09:** Minor textual changes, added Qt requirements, updated tnm_parallelcoordinates files, changed `_selection` to `_brushing`
**18/09:** Minor textual changes, added reference to `make`, updated tnm_raycaster
**24/09:** Updated rc_raycaster.frag to include `datasetDimensionsRCP_` and updated tnm093.vws to include ground truth of front-to-back compositing and shading, add hint about `datasetDimensionsRCP_`, outsourced documentation about how to compile Voreen on own machine
**30/09:** Removed part 6 (Brushing & Linking)