IN4089 - Data Visualization - Project 2: Volume Visualization Framework Introduction

Introduction

First see if you can compile the project. For this, please download the source code. We rely on CMAKE, which allows you to produce a solution file for the code editor of your choice; Visual Studio, Visual Studio Code (runs on all platforms) or Clion (free for students). A documentation of configuring CMAKE can be found in the project folder. It should be possible to run the project out-of-the-box on Mac OS X, Linux, and Windows systems. Once you have created a solution file for your editor, you should be able to compile and execute the program. If you are a beginner, we suggest that you use Visual Studio Code or Visual Studio 2019, a setup guide can be found in the project folder. Beware that downloading and installation may take some time.

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, you do not need deep knowledge of C++, as we do provide a code skeleton that will support you. The syntax should be similar to most other programming languages that you might have used before but you might need some time to get used to it.

You do not need to go through all the files and classes. We will list the ones that are relevant below.

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. In order to make his possible, Hence, you are not allowed to modify the existing functions, besides their code body and classes can only extend the original classes but need to implement the original functions as outlined in this document.

Code Overview

As stated before, you need to focus on the parts of the skeleton code that are needed for your implementation. It is not the idea that you fully understand the functioning of the whole framework. Being a C++ project, the code is divided in header (`.h`) and source (`.cpp`) files. The class structure, member variables and function names are defined in the header files, while the actual implementation is done in the source files. The skeleton code is divided into four folders, `ImGui`, `Render`, `UI` and `Volume`. Except for the last part of the assignment (extension of the program), you only need to concern yourself with `Render` and `Volume`.

- Dear **ImGui** The interface is built using the Dear ImGui library, this folder contains the code necessary for this to work.
- **Render** The class Renderer (`render/renderer.h`) is where most of the raycasting algorithms/functions are done and it is the class that you will need to update and implement. To help you orient yourself, it currently generates a view-plane aligned slice through the center of the volume data and a maximum intensity projection rendering. The renderer also draws a bounding box to give visual feedback of the orientation of the data set.
- **UI** This folder contains the definition of the program interface (`ui/menu.h`), provides a virtual trackball ui/trackball that allows you to move around the volume with your mouse. It also contains `ui/transfer_func` and `ui/transfer_func2d`. Both have a `draw` function to display the UI (using imgui) and `updateRenderConfig` function updates the RenderConfig struct, which stores all rendering configuration variables. Unless you want to implement an extension there is no need to change these classes. You will instead use them to get access to the user designed transfer functions.

- **Volume** The class Volume (`volume/volume.h`) stores the volume data and contains functions to retrieve values from the volume. It is an important class where you will also need to implement some of the methods for interpolation.
  The class GradientVolume (`volume/gradient_volume.h`) stores and provides methods to compute and retrieve gradient vectors, some of which you need to implement yourself.

glm Library

For **vector math**, we will rely on the glm library. It is a powerful collection of classes to perform vector and matrix math and has become a standard for 3D applications. You can find documentation for glm at https://openframeworks.cc/documentation/glm/.

Automatic Gading

Because your code will partially be graded automatically (we run test cases with reference solutions and compare it to the result of your functions), it is very important that you **do not modify or remove existing code in the header files**. Addition of new functions or classes is allowed. You can always check whether your code is still valid by running the integrity check provided with the project.
For the extension assignment, you should submit a separate solution that will not be graded automatically. Therefore, feel free to make changes as you please.

Datasets

We provide several data sets to work with for this assignment. We recommend you to use the carp8.fld for testing, since the size is small and some of the processes we will be dealing with can be quite time and resource consuming.

First Run

Let's get an overview of the framework by compiling the framework and check the code and running app.

- Run the app through Visual Studio (Code) and see what functionality is already available (e.g., zoom in the slides and change the resolution).
- The central class we will be working with is the Renderer class, found at `render/renderer.h` and `render/renderer.cpp`. Several functions are part of this class but you should focus on the ones that we mention and are needed for the assignment.
- To start, study the method `traceRaySlice(...)` in the class Renderer. We will use this class as a basis to develop a raycaster. Try to understand what is happening in the function, also based on what you see changing when you change the resolution.
- The code already provides a transfer function editor so you can more easily specify colors and opacities to be used.
- Go back to the code and look at the function `traceRaySlice(...)` and activate the Transfer Function (TF) in order to understand how it is used.
- Run the code and change the transfer function (left mouse button adds control points, right mouse button removes them)
- Study the function `render(...)` in the class `Renderer` to see how the different types of visualizations can be activated and the initial direction and position of the ray are calculated.
- explore which of these methods are already working in the app.
- MIP (Maximum Intensity Projection) already works. Study the MIP Code (function `traceRayMIP(...)`) to see how the ray traversal works. This function only contains code for traversing a single ray. Setting up the rays for the complete image is already taken care of.