CSE471 Introduction to Computer Graphics (Fall 2019)
Instructor: Prof. Won-Ki Jeong
Due date: Dec 8, 2019, 11:59 pm.

Assignment 4: Volume Rendering

In this assignment, you will implement direct volume rendering. Specifically, you are required to implement a ray casting algorithm using alpha compositing. The goal of this assignment is to learn how to use 3D texture and alpha blending to make 3D rendering. An example of expected volume rendering result is as follows:

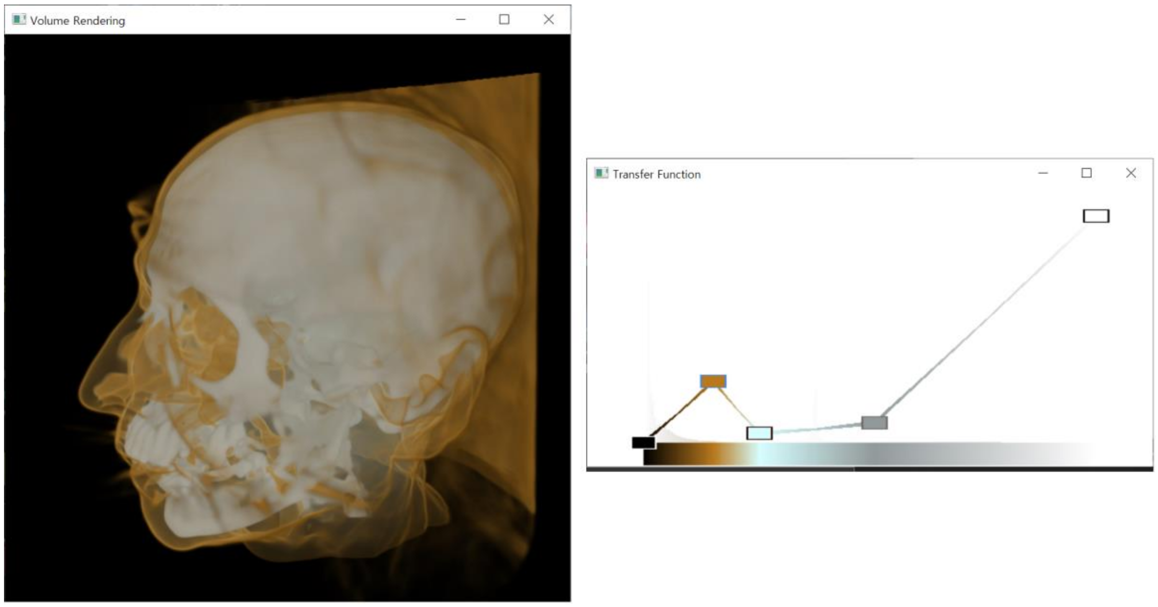


Figure 1. Left : Volume Rendering Window, Right: 1D Transfer Function Editor

The provided source code already has complete implementation of the viewer with virtual trackball and the transfer function editor. You only need to implement `volumeRendering.frag.` Do not change other code except the fragment shader code.

1. Overview of the skeleton code

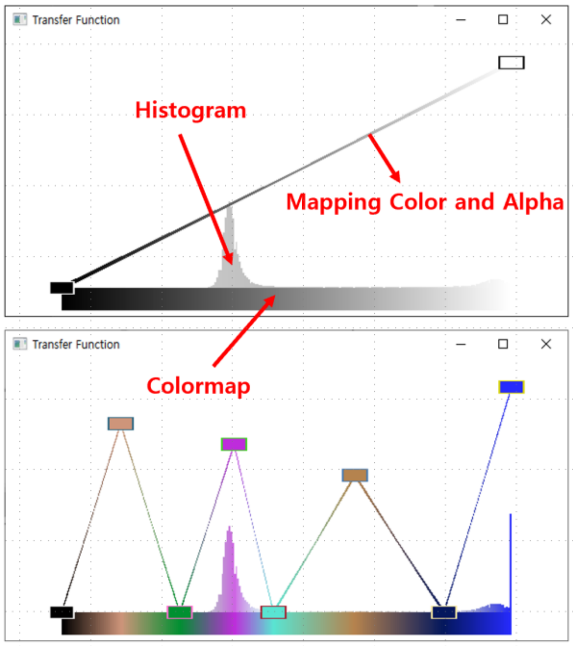
The provided volume rendering viewer accepts mouse interactions. Left-click and move is rotation, and right-click and move is zooming. The following variables in the fragment shader is the viewer-related parameters:

`Vec3 eyePosition`: location of the viewer

`Vec3 up` : Up vector of the viewer
`Vec3 objectMin` : the smallest x/y/z of the input volume
`Vec3 objectMax` : the largest x/y/z of the input volume

Note that the viewpoint is rotated (not the volume itself) in the provided code. Therefore, when you rotate the volume, only `eyePosition` and `up` are changed (`objectMin/Max` are defined when the input volume is loaded and will not be changed after that).

The below is the instruction of transfer function editor:



Transfer Function axis

- X: voxel data value (0~255, 8bit)
- Y: alpha
- Node color: R,G,B
- Edge color: color interpolation of end nodes

Transfer Function control

- Mouse left button drag: change node position
- Mouse Right button click: change node color (randomly assign color when it clicks)
- Shift key + Mouse left button click: create new node
- Shift key + Mouse right button click: delete existing node

2. Ray casting (80 points)

To make the volume renderer working, you need to complete the partially filled fragment shader. There are two parts you need to implement as follows:

2.1 Ray-cube intersection

In this part, you need to compute intersection between the viewing ray and six sides of the cube to find two endpoints, one is the entry and the other is the exit point, which is defined as an array `vec3 endPoint[2]`.

2.2 Color compositing

Once the entry and exit points are found, then we need to walk on the viewing ray, starting from `endpoint[0]` to `endpoint[1]`. The number of samples (or the step dt) can be defined on your own. You should implement two color compositing algorithms: maximum intensity projection (MIP), and alpha compositing.

For maximum intensity projection, you need to pick the maximum intensity among the sampled values.

For alpha compositing, first you should read the volume intensity at the given sampling location using a 3D texture (`sampler3D tex`), and then you should use the transfer function texture (`sampler1D transferFunction`) to convert the voxel intensity to RGBA value. Using RGBA, you should apply alpha blending. You should use front-to-back compositing and early termination (if alpha value is close to 1.0 then terminate). You also need to adjust alpha to allow easy control of transparency (alpha^x will make alpha value smaller).

Switching between two rendering modes can be done by commenting out #define MIP in the fragment shader code.

Grading:
- MIP works correctly (30 pts)
- Alpha blending works correctly (50 pts)

3. Report (10 pts)

Submit a report describing your code, implementation details, and results.

4. Rendering results (10 pts)

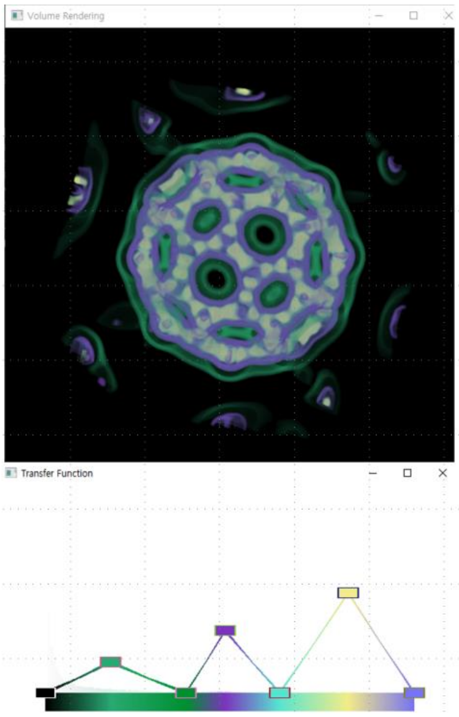
There are 5 volume data (.raw files) in the provided code. Make the most beautiful rendering images (one per volume data) and submit them.

What to submit:

- VolumeRendering.frag file (Do not submit other files, such as cpp or h)
- A report (pdf format)
- 5 rendering images (one per each volume data)

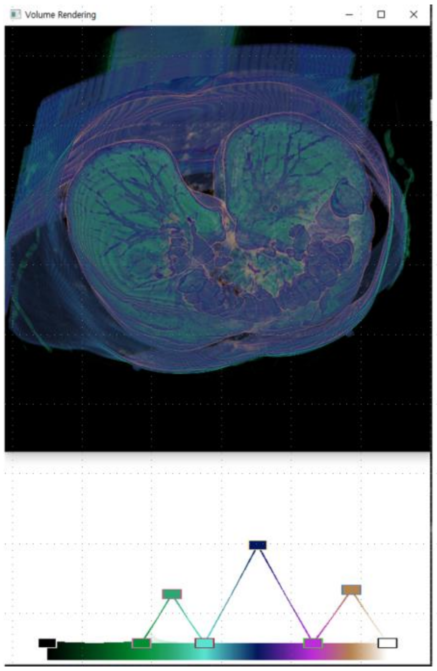
Good luck!!!

Examples:



Bucky_32_32_32.raw

Dimension (x, y, z): 32, 32, 32



lung_256_256_128.raw

Dimension (x, y, z): 256, 256, 128