

## Lab Assignment 2 – CSE 564, Spring 2011

*Due: Monday, March 14, 11:59pm*

This lab will introduce you to X-ray, MIP,  $\alpha$ -composited, and shaded iso-surface volume rendering. Use the CUDA/FLTK base-code provided on blackboard (and introduced in class). The code has been tested with CUDA 3.2 (32 bit) and FLTK 1.1.10. It contains an exe file with the basic functionality (in the bin folder) and the various source files (in the projects folder).

Upon completion of the assignment please submit the following via blackboard:

- the complete software (all that is needed to build the executable: source code, project files, etc)
- an executable (.exe file) of your work
- all screenshots should be of your entire user interface with the processing result shown in the window(s)
- a comprehensive report that illustrates with screenshots, narrative text and code snippets of all aspects of your work

All of these components are equally important. Please note the policies posted on the class webpage.

The overall theme of this project is to visualize volume datasets with raycasting. Use the volume datasets given on the lab webpage. The provided code performs back-to-front  $\alpha$ -composited rendering and the transfer function is set to fixed setting (in `initCuda()` in `volumeRender.cu`). The rendering is restricted to a fixed volume (Bucky.raw), which is read in `initialize()` in `DisplayWindow.cpp`.

Your first step should be run the exe file to make sure your computer is setup and CUDA-capable. You should be able to interact with the buckyball (rotate with left mouse button down, zoom with the right mouse button down). Then look through the software and get familiar with it. You will have to add new code in various places so better get an overview first. There are various cpp files. Important are `TransFuncEditor.cpp` that deals with transfer function control, `DisplayWindow.cpp` that deals with rendering control, and `Application.cpp` that deals with general setup. The file `gui.cpp` is generated by the GUI editor Fluid which is part of the FLTK package. The file `volumeRender_kernel.cu` is where you will specify your CUDA kernel code.

In the following two tasks, look for comment lines labeled with ‘TASK’ in the various cpp files.

### 1. Write a function to read the volume data

The header file of the volumes posted on the lab webpage is similar to the ppm header. It is as follows:

P7 – indicates a volume data file

nx ny nz – the dimensions

255 – the density resolution

data (there will be one byte per data element, not the three (RGB) bytes you had for the images)

It uses the identifier P7 to decide on read time that you will be dealing with a volume (as opposed to an image). The data are stored in row-major, that is, the first nx data points are due to the first row of voxels,

and the first  $n_x \cdot n_y$  data points are due to the first volume slice, and so on. In the present version the Buckyball volume is read in by default and the volume size is fixed to a cube (that is,  $n_x = n_y = n_z$ ). The volumes you will be reading in using the file chooser do not necessarily fit in a cube. The task is therefore to write a function that can read in arbitrary volumes and can fill the respective data arrays for rendering, both on the CPU and the GPU. These volumes should then render in the colors of the Buckyball (because the transfer function is still fixed).

## 2. Add transfer function editing

You are now asked to specify your own transfer function (using the existing editor). There are two tasks: (1) editing the transfer function in the editor window on the left, and (2) copying the transfer function array to the GPU. This is described in the following.

First, currently the transfer function editing is only enabled for the red (R) channel. You now need to extend this to the blue, green, and  $\alpha$ -channels (GBA). You could add extra selector buttons on the GUI canvas (via FLTK) which would be most convenient, but you could also just control the selected transfer function via a depressed key (see the `handle(int event)` function in `DisplayWindow.cpp`).

Next, you need to copy the transfer function array as a texture to GPU memory using `cudaMemcpyToArray()`. You do this in the `handle` function in `TransFuncEditor.cpp`. There are two arrays:

On the CPU: `TransferFunction transFunc[4]`

On the GPU: `cudaArray *d_transferFuncArray`

If you did this correctly, then every time you modify the transfer function it gets copied to the GPU and the rendering result will change accordingly.

The following three tasks require you to write three additional CUDA kernels.

## 3. Implement X-ray rendering

You could use a key-event handler to change (select) the rendering mode to X-ray, or add a dedicated selector button on the GUI canvas. Since X-ray may produce values greater than 255, you need to divide the values by a suitable value. Here, you could choose some constant (modified using the arrow keys), or add a respective slider bar on the GUI canvas. This will allow the image to get brighter or darker. Make sure to account for ray step size in the X-ray accumulation, or you get very large values for small step sizes. Just multiply the result by the step size, this will compute the accurate Riemann sum.

## 4. Implement MIP rendering

Choose this rendering mode similar to the previous. No normalization is needed here – you are only keeping the maximum density value.

## 5. Implement shaded iso-surface rendering

This rendering mode uses the  $\alpha$ -compositing from before. Just now you use the A-channel of the transfer function to enhance surfaces by specifying a step function (from 0 to 1) at the desired density. Slightly

rounding the step-type transfer function on the bottom and top typically produces better looking results. Try this with the existing  $\alpha$ -compositing.

Next add shading. For this you need to estimate a gradient vector and perform the lighting. For the former use simple central differencing (interpolate the corresponding values with `tex3d()`) and for the latter assume the light is at the same location than the viewer. Thus, the view vector and the light vector are identical. If you wish you could change this for different effects.

## **6. Generate 3 showcase images that make you proud**

Play with the color and alpha transfer functions. See if you can enhance certain volume features with it. See also if you can inspire your inner artist and generate some ‘pretty’ pictures. Add the best of these to your report.