

In this assignment, you will implement—in teams of two, if you like—a parallel volume rendering algorithm using Pthreads. You will parallelize the volume rendering algorithm using the code template we provide. This code template contains most needed functions to implement volume rendering.

## 1 Background

Volume rendering is a technique for visualizing discrete three-dimensional (3D) data, typically after being projected onto two dimensions. Volume rendering is used in many fields, including medical imaging for visualizing MRI (Magnetic Resonance Imaging) and CT (Computed Tomography) scans.

## 2 The Algorithm

Many algorithms have been proposed for volume rendering. The specific algorithm that you will use for this assignment is described by Marc Levoy's seminal paper, "Efficient Ray Tracing of Volume Data," which is available on the class website. The paper presents a brute-force sequential algorithm for volume rendering, followed by two optimized sequential algorithms. We suggest you start by parallelizing the brute-force algorithm. Then, for extra credit, you can implement parallel versions of the optimized algorithm(s).

As with Assignment 1, you will use the AIX machine, `champion.tacc.utexas.edu` for this assignment.

## 3 Details

The pthreads implementation of the brute-force algorithm should be straightforward. Your first goal is to implement this brute-force algorithm without any of the following optimizations/restrictions. Once you complete that, to make the assignment more interesting, we will add a couple of twists. First, we will place a limit on the amount of main memory that you can access, so you *cannot* assume that the entire input fits into a single shared array. This memory restriction only holds for the shared array holding the input dataset. The memory needed for holding the output image pixels can be allocated independent of this memory limit. Ideally, your implementation will be parameterized to accept different memory limits and different input image sizes.

Second, we will provide inputs in which the work is not balanced evenly across the 3D space. So, some threads will complete soon and idle while others have more work to do. You should think about ways to minimize this load imbalance.

Of course, as usual, we suggest that you start with simplifying assumptions and incrementally add the memory constraints, load balancing strategies, and optimized algorithms.

### 3.1 Simplifying Assumptions

In order to simplify this assignment, we will make the following assumptions:

1. The paper describes what are called "Transfer Functions," which basically map an 8-bit value (or a 12-bit or 16-bit value depending on the data set) to either an opacity or a color. The opacity or the color will be in the range from 0 to 1. You should use the simple transfer functions that we have provided in the code template.
2. You should assume that the volume data is stored as shown in Table 1.
3. We will assume that the volume is axis-aligned with the coordinate system and that the camera is looking directly into the volume. This simplifies all of the following functions mentioned in the paper: **First, Last, Image and Object**. We have provided implementations of these functions in the code template.

Byte Number	Z-Index	Y-Index	X-Index
0	0	0	0
1	0	0	1
2	0	0	2
.			
.			
N	0	0	MAX-X - 1
N+1	0	1	0
N+2	0	1	1
.			
M	0	1	MAX-X - 1
M+1	0	2	0
.			
.			
L	0	MAX-Y - 1	MAX-X - 1
L+1	1	0	0
.			
.			
.			

Table 1: Indexing the Volume Data

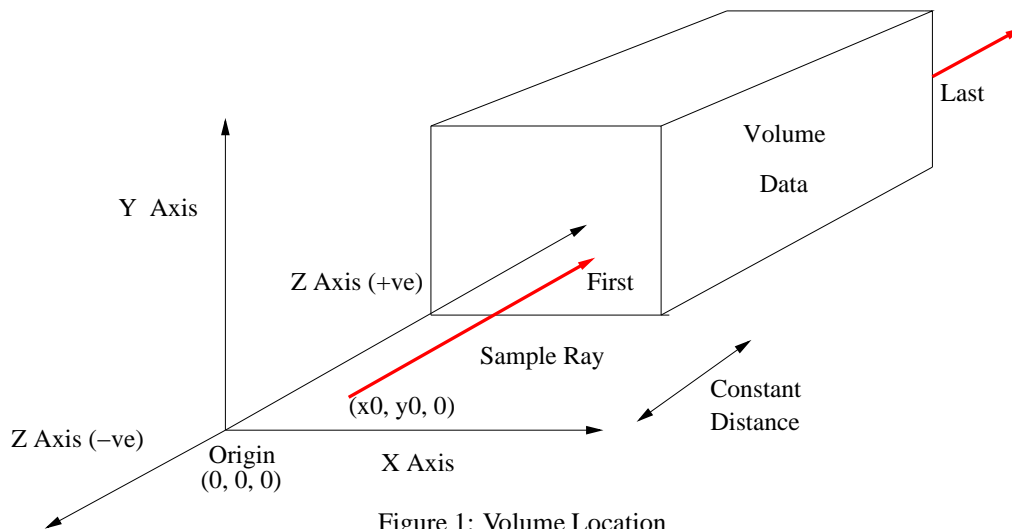


Figure 1: Volume Location

Figure 1 describes the assumptions about the placement of the volume. The volume data (typically a cuboid) is axis-aligned and placed at a constant distance along the positive Z-axis from the origin. The  $Z=0$  plane will be the starting point for all the rays in the algorithm. The red ray shows a sample ray shot from the point  $(0, 0, 0)$ . The ray enters and exits the volume at the points First and Last, respectively (our code template provides these functions for this camera setup). Your implementation should traverse the Volume data along the path of each ray from the point at which it enters the volume to the point at which it exits the volume. You should increase the value of Z in increments of 0.5 while stepping along the Z-axis (this value should ideally be provided via a command line argument, see below).

## 3.2 Code Template

We will provide a code template for this assignment that contains most of the functionality needed for volume rendering. The template will include a makefile and a few C files. The file named “seq\_vol\_render.c” contains all the needed functionality for building the volume renderer, including the transfer functions, First, Last, Image and Object functions, and the function to create the final output image after the algorithm completes.

Please see the **print\_usage** function in “seq\_vol\_render.c” for the various command-line parameters your code should support. You can add additional command-line parameters if you like.

### 3.2.1 Inputs and Outputs

We will provide raw volume data as inputs. These files will be named as input1.raw, input2.raw and so on. We will also provide the dimensions of the volume data. Your program will take the volume data, perform the actual volume rendering using the provided transfer functions, and write the output image to a file. The output image will be in the PPM format (Portable Pixel Map). On a linux system, you can use the “display” command to see this file. Or you can use software packages like gimp to see the resulting PPM file.

To test your implementation, we will provide a few test input datasets and their corresponding output images, which were generated by a sequential implementation of the algorithm. These images will be named sample\_outputX.ppm, where ‘X’ is the input dataset number. We will include a few large datasets that you can use to measure performance. These large datasets also will test how well you have parameterized your pthreads implementation to handle large amounts of memory. We will also include a few additional inputs for which we will not provide the corresponding output images. Finally, we will also test your implementation on a few blind datasets (which will not be provided to you).

### 3.2.2 Parameterization

Your code should be (ideally) parameterized (via the command-line) enough to handle the following:

1. input datasets of 8, 12 or 16 bits. That is, each voxel can be 8, 12 or 16 bits depending on the input dataset. If the dataset is not 8 bits, you can run into byte-ordering issues. We will keep you posted on this.
2. the maximum memory allowed (in megabytes) for storing the input dataset. This parameter will control how much memory you can malloc to store the input dataset. Our largest datasets could only be a few hundred MB in size. So, we will test your implementation with different values for this parameter. For example, if the entire input dataset is 16 MB in size, we can test with a value of 8 MB for this parameter. Note that this restriction only applies for the input dataset. You are allowed to consume “reasonable” amount of memory for the other data structures (holding the final image.)
3. As mentioned above, you will first start with a step-value of 0.5 for advancing along the Z-axis. You should make this step also a parameter, the values can be from 0.1 to 0.8.

## What to Turn In

This assignment is due at 11:59pm on the due date. Use the **turnin** program to submit your solution, which should include the following:

1. A written report of the assignment in either plain text or PDF format. This is your chance to explain your approach, any insights gained, problems encountered, etc. Your report should include performance results for your solution.

2. Your source code, instructions to build it on Champion, and instructions to run the programs (along with the arguments, etc).
3. A file called `submit.info`. This file should have three lines:

Line 1: Full name of first student

Line 2: Full name of second student (leave a blank line if there is no second student)

Line 3: email id of first student, email id of second student

If you want to use any late submission days, please email the TA, because the turnin program will be disabled at midnight of the due date, so you will have to mail your work to the TA to turn it in late.