User's Guide: pCT Image Reconstruction Program

Structural Design, Appropriate Settings, and Detailed Usage

Blake Edward Schultze

June 18, 2014

(1) Specifying location of input data and where output is to be written.

First, specify the directory where the input data is stored (“input_directory”) and the directory where output data should be written (“output_directory”). Note that since the various data sets are typically stored in separate folders in the same directory, the program is designed as such, so do not include the folder where the data is located in the directory name, the folder name should be written to the variable “input_folder”. Similarly, output for various data sets are typically in separate folders in a single directory, so do not include the output folder name in the output directory name, write the folder name into the variable “output_folder”. The filenames for the data files should then be written to the variable “input_base_name” and the file extension to “file_extension” (e.g. .bin, .dat, .txt, etc). Note that some compiler recognize the backslash character “\” as an escape character, e.g. “\n” is new line, “\t” is a tab, etc. To tell the compiler not to interpret this literally, you must use “\\”.

The filename format is expected to be “[input_base_name]_xxx.[file_extension]”, where “xxx” is a 3 digit number specifying the gantry angle from which the corresponding data was acquired. This completes the input/output path specifications. Since data has been written in various formats in the past/present, you must specify if the data is written in data format Version 0, Version 1, or the format prior to Version 0 by setting the corresponding boolean to true and the others to false (VERSION_0, VERSION_1, and VERSION_OLD, respectively). Similarly, since there have been inconsistent units used in the past (i.e. mm/cm), two booleans have been added to provide compatibility by setting the “SSD_IN_MM” and “DATA_IN_MM” booleans appropriately. In the past, many of the data sets also came with a config file (scan.cfg) which specified the u coordinates of the tracker planes, so if you are using any such data set, set the CONFIG_FILE boolean to true. Otherwise, you must set it false.

If the program cannot find a file, whether it be data or a config file, an error message is printed to the console window and program execution is halted. To allow one to see the error message, an user input request is initiated. Once you are finished reading the error message, hit any key to exit the program.

(2) Hull-Detection (i.e. determining object and its boundary for use in MLP)

There are currently 4 methods of hull-detection: FBP, Space/Silhouette Carving (SC), Modified Space/Silhouette Carving (MSC), and Space/Silhouette Modeling (SM). These can be turned on/off using the variables FBP_ON, SC_ON, MSC_ON, and SM_ON, respectively. These can typically all be turned on at the same time so their results can be compared, but this is not always possible; if the data sets are large, since each of these algorithms consume GPU memory (particularly SM), it is possible to exhaust the available GPU memory. In most cases this does not raise an execution halting exception, but subsequent processing generates erroneous results. This can be seen from the text output to the console window which notifies the user of how many protons passed through the reconstruction volume and, thus, were not cut from the data set: if memory is exhausted, the program will indicate that 0 histories pass reconstruction volume intersection detection for the remainder of gantry angles not yet processed.

(3) Writing Data to File

There are several functions provided for writing data to disk and the appropriate choice depends on the data type (boolean, integer, float), its container type (C array, vector), and if the data is to be written to a single file or separated into several files. Three parameters dictate the structure of these files, the first two specify the number of rows and columns and the last dictates either the number of files with this structure or the number of times this row/column structure is repeated below each other in a single file (depending on if writing to one or multiple files). This structure was designed with images in mind, but this does not preclude using them for any other type of data. These functions can be added anywhere in the program, though if the data desired resides on the GPU, it must first be transferred to the host using cudaMemcpy(device array name, host array name, data_size, cudaMemcpyHostToDevice).

(4) Host/GPU computation and structure information

A proton history contains (12) u/t/v coordinate measurements, a WEPL measurement, and the gantry angle at which these measurements were acquired, so for most data sets and on most computer systems, the storage capacity of the GPU is not large enough to store the entire data set at one time. Thus, we can only process a subset of the data set at one time and then repeat until the entire data set has been processed. The number of proton histories in these subsets is controlled with the constant MAX_GPU_HISTORIES and its upper limit depends on the amount of GPU memory; eventually, the program will query the system to determine the GPU's specifications and determine this value automatically. However, because the program is still in development and it is unclear exactly how much memory is needed for basic operation and how much additional memory is required per proton history, MAX_GPU_HISTORIES must be set manually.

At the moment, the program begins by determining the number of proton histories in each file and then on each iteration, it determines how many files it can process at once before exceeding MAX_GPU_HISTORIES. To provide forward compatibility and prevent the need for structural redesign in the future, the preprocessing routines are capable of processing any number of proton histories at once. This not only allows us to reach maximum efficiency by processing as many proton histories at once as possible, but it also allows us to begin preprocessing as data becomes available during acquisition, before completing the scan. In other words, the software does not limit the number of histories that can be processed simultaneously, the only restriction on this number is the GPU capacity. However, for convenience, if a file cannot be read in entirety without exceeding MAX_GPU_HISTORIES, the file is not included in the current iteration. Since this restriction is not enforced by any of the processing routines, it is isolated in the "while" loop in the main function (and to a lesser degree the function used to read the data) so it can easily be removed in the future if necessary.

During early development, it was found that 4-5 million proton histories could be processed simultaneously (though this may have dropped by now and obviously depends on each file's size), corresponding to around 5-20 files depending on the number of histories in each file. On the other hand, it was also found that increasing the number of histories processed simultaneously had negligible effect on execution time.

Although initially counterintuitive, this is due to the fact that computation accounts for a very small fraction of execution; almost the entire execution time is associated with data transfers to/from the GPU. Thus, it is primarily the total size of the particular set of data to be transferred that dictates the transfer time and it makes very little difference whether it is transferred in one piece or broken into smaller pieces. Of course, there is some overhead cost associated with data transfers, but this accounts for an even smaller portion of execution time than computation does. Therefore, since there are some minor complications with partially reading files and it is convenient for development verification to read a single file (i.e. gantry angle) at a time, the current implementation does not allow partial reading of files.

Although multiple files can be read simultaneously, provided that the total number of histories does not exceed MAX_GPU_HISTORIES, this is not recommended. The hull-detection algorithms consume memory as well (particularly SM), and if multiple hull-detection algorithms are to be used simultaneously so their results can be compared, exceeding 1-2 files at a time can potentially exhaust the available GPU memory (refer to hull-detection section on indications of memory exhaustion). As their is no significant benefit to processing more than 1 file at a time, MAX_GPU_HISTORIES is typically set to a value slightly larger than the maximum number of histories per file. As long as the number of histories per file (gantry angle) is relatively constant, this will enforce single file processing. This approach is particularly useful during development for identifying problems with data and/or computation, especially the more insidious issues with unstable numerics that only occur in certain situations or under particular circumstances, as these can generate computational errors that either go unnoticed or are difficult to locate. For example, trig computations near vertical angles, where cos -¿ 0 =¿ tan -¿ infinity, causes computational errors that become quite large the closer to vertical the angle is. This particular issue was encountered several times during development, both in simulated data sets and with preprocessing computations; the simulated data associated with vertical projection angles was invalid due to numerical errors calculating proton paths and the preprocessing program inaccurately calculated the points of intersection of the proton path with the reconstruction volume for high slope paths, with errors increasing in magnitude as angles approach vertical. These numerical instabilities occurred infrequently and may have gone unnoticed if each gantry angle was not processed individually. In fact, the original pCT reconstruction program contained numerical instabilities (e.g. the same intersection inaccuracy), but because its effects were rarely encountered, the propagation of the error was not dramatic enough to be immediately noticeable (contact me if interested in details).