Chapter 1

MC-GPU v1.3

```
Andreu Badal, PhD (Andreu.Badal-Soler{at}fda.hhs.gov)

   Division of Imaging and Applied Mathematics
   Office of Science and Engineering Laboratories
   Center for Devices and Radiological Health
   U.S. Food and Drug Administration

Code release date: 2012/12/12
```

MC-GPU [1-4] is a Monte Carlo simulation code that can generate synthetic radiographic images and computed tomography (CT) scans of realistic models of the human anatomy using the computational power of commodity Graphics Processing Unit (GPU) cards. The code implements a massively multi-threaded Monte Carlo simulation algorithm for the transport of x rays in a voxelized geometry. The x ray interaction models and material properties have been adapted from **PENELOPE 2006** [5].

MC-GPU was developed using the **CUDA** programming model from **NVIDIA** [6] to achieve maximum performance on NVIDIA GPUs. The code can also be compiled with a standard C compiler to be executed in a regular CPU. In a typical medical imaging simulation, the use of GPU computing with MC-GPU has been shown to provide a speed up of between 20 and 40 times, compared to the execution on a single CPU core.

The MC-GPU code has been described in different scientific publications [1-4]. The main reference of this work, which the users should cite, is the following [1]:

```
Andreu Badal and Aldo Badano, "Accelerating Monte Carlo simulations of
photon transport in a voxelized geometry using a massively parallel
Graphics Processing Unit", Medical Physics 36, pp. 4878-4880 (2009)
```

The main developer of MC-GPU is **Andreu Badal**, working at the U.S. **Food and Drug Administration** (Center for Devices and Radiological Health, Office of Science and -Engineering Laboratories, Division of Imaging and Applied Mathematics). The source code of MC-GPU is free and open software in the public domain, as explained in the

Disclaimer section below. The source code of MC-GPU and its auxiliary files are distributed from the website: `http://code.google.com/`.

This documentation has been automatically generated by **Doxygen** parsing the comments in the MC-GPU source code. This code is still in development, please report to the author any issue/bug that you may encounter. Feel free to suggest improvements to the code too!

1.1 List of modifications in different versions of the code

1.1.1 Version 1.3 (release date: 2012/12/12)

- Code upgraded to CUDA 5.0 (not compatible with previous versions of CUDA!).
- Removed limit on the amount of projection images that can be simulated per CT scan (source and detector parameters now stored in global memory and transferring to shared memory at run time to avoid using the limited constant memory).
- New material dose tally implemented to estimate the dose deposited in each material independently of the voxel dose tally (the voxel dose tally measures the dose in each material adding the energy deposited in each voxel of that material within the defined voxelized region-of-interest).
- Interaction loop re-organized to maximize performance (virtual interactions simulated before real ones).
- Improvements and small corrections in the source sampling and tally routines.
- Allow input of material and voxel geometry files compressed with gzip (zlib library now required for compilation).

1.1.2 Version 1.2 (release date: 2011/10/25)

- Implemented a voxel dose tally.
- Polyenergetic source model.
- MPI support for simulating individual projections.
- Simulation by time limit.
- Improved flexibility of the CT trajectories, helical scans.

1.2 Disclaimer

This software and documentation (the "Software") were developed at the Food and - Drug Administration (FDA) by employees of the Federal Government in the course of their official duties. Pursuant to Title 17, Section 105 of the United States Code, this work is not subject to copyright protection and is in the public domain. Permission is hereby granted, free of charge, to any person obtaining a copy of the Software, to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, or sell copies of the Software or derivatives, and to permit persons to whom the Software is furnished to do so. FDA assumes no responsibility whatsoever for use by other parties of the Software, its source code, documentation or compiled executables, and makes no guarantees, expressed or implied, about its quality, reliability, or any other characteristic. Further, use of this code in no way implies endorsement by the FDA or confers any advantage in regulatory decisions. Although this software can be redistributed and/or modified freely, we ask that any derivative works bear some notice that they are derived from it, and any modified versions bear some notice that they have been modified.

1.3 Code features

In this section we provide a brief description of the features of the MC-GPU code. A more complete description of the code can be found in our published articles. important information regarding the operation of the code is provided as comments in the input files of the sample simulations provided with the MC-GPU package. Detailed information on each function of the code can be found in the complete Doxygen documentation of the source code

The basic operation of the code consists in adapting the simulation input file to describe the location and characteristics of the x ray source, define the CT trajectory (if any), list the materials to be used in the simulation, define the geometry of the x ray detector and, finally, specify the voxelized object file to be used as the simulation material universe. In the first line of the input file, the user can fix the total number of x rays that have to be simulated ($>$ 1e5 histories) or the total simulation time (maximum 1e5 seconds).

The coordinate system of the simulated world is determined by the input voxelized geometry. The origin of coordinates is assumed to be located at the lower-back corner of the voxelized volume, and the axis are located on the vertices of the voxelized volume. This means that the lower-back corner of the first voxel is on the origin and the following voxels are located along the positive X, Y and Z axis (first quadrant).

To simulate the atomic interactions, MC-GPU uses a database of material properties based on the database from PENELOPE. A PENELOPE 2006 material file can be converted into an MC-GPU material file using the auxiliary utility "MC-GPU_create_-material_data.f" provided with the MC-GPU package. Pre-defined material files for a set of materials typically used in medical imaging simulations are already provided in the folder "MC-GPU_material_files".

The code includes two tally options: an **image tally** that creates projection x-ray images, and a radiation **dose tally** that estimates the dose deposited inside the patient model. MC-GPU does not currently simulate the transport of electrons and therefore the dose deposition tally (KERMA tally rigorously) will not be accurate for high energies or near material interfaces and small voxels. In the image tally the images are formed by counting the energy that enters a user-defined 2D grid of pixels, which is a simple approximation to a noise-free flat-panel detector with 100% detection efficiency. The pixel values have units of eV/cm^2. Four different images are reported at the end of the simulation, corresponding to the signal produced by x rays that did not interact between the source and the detector (non-scattered), x rays that suffered a single Compton (inelastic) interaction, a single Rayleigh (elastic) interaction, and multi-scattered x rays. The dose tally counts the energy deposited by each x ray track inside each voxel of the geometry, within a user-defined volumetric region-of-interest (ROI). The average dose deposited inside each voxel and in each material (and the associated statistical uncertainties) are reported at the end of the simulation.

MC-GPU can simulate a single projection image or a full CT scan. The CT is simulated generating many projection images around the static voxelized geometry. Currently, the code is limited to perform a simple CT trajectory rotating around the Z axis. The user can specify the angular shift and longitudinal translation (pitch) of the source between each projection and also the distance between the source and the axis of rotation (the axis is assumed to be parallel to the Z axis). By now, the code does not simulate some relevant components of a CT scanner such as the anti-scatter grid, a bow-tie filter or a curved detector (flat-panel detector only).

The x ray source is defined as a point source emitting x rays with an energy randomly sampled from the user-provided energy spectrum. The polyenergetic spectrum is efficiently sampled using the Walker aliasing algorithm. The emitted cone beam is computationally collimated to produce a rectangular field on the detector plane, within the azimuthal and polar angles specified by the user. The detector plane is automatically located at the specified distance right in front of the source focal spot, with the collimated cone beam pointing towards the geometric center of the detector.

In order to optimize the particle tracking algorithm (ray-tracing) and minimize the accesses to the slow GPU main memory, the photon trajectories across the voxels are computed using the Woodcock tracking algorithm. With this technique the photons perceive the geometry as a uniform medium composed of the material of the most attenuating voxel. In this way, the voxel boundaries do not have to be explicitly calculated and multiple voxels can be crossed in a single step. To keep the simulation unbiased, some of the interactions are considered "virtual" (i.e., do not change the photon energy or direction of movement), depending on the x ray energy and the actual material at the interaction site. In typical medical imaging simulations where the most attenuating material is cortical bone, the Woodcock tracking algorithm gives an speed up of almost one order of magnitude compared to computing voxel boundaries all the time. However, if the geometry includes a high density voxel, such as a metallic implant, the performance of the code can be severely reduced because a large fraction of the sampled interactions will be virtual.

The random number generator used in PENELOPE [5], RANECU, is also used in the GPU program. To ensure that the simulated tracks are not correlated, each thread initializes the generator to a unique position in the random sequence, far enough from the other threads, using the algorithm implemented in the seedsMLCG code [7].

In a typical simulation, several thousand threads are launched simultaneously in the GPU, each one of them simulating a batch of several x ray tracks. If the code is compiled with MPI support (see below), multiple GPUs can be used in parallel. The code will perform a short speed test to estimate the relative speed of each GPU used in the simulation and then distribute the number of particles among the available GPUs correspondingly. If the user specified a time limit in the simulation, all the GPUs will simulate in parallel for the allowed time. Since the code is already optimized to scale well in thousands of GPU threads, it scales almost linearly with the number of GPUs in most situations, with only a few seconds of overhead in the initialization of the multiple GPUs and in the reduction of the final results.

1.4 Code output

At the end of the simulation the code reports the tallied 3D dose distribution and the final simulated images in RAW binary form, as 32-bits float values. The image data is provided as a collection of five consecutive images corresponding to: total image (scatter+primaries), primary particles, Compton, Rayleigh and multi-scatter. The dose data is reported as two RAW files with the mean dose and twice the standard deviation of the dose in each voxel of the geometry respectively, within the input ROI. The average dose deposited in each material of the geometry is also reported to the standard output. Organ doses can be obtained by post-processing the output dose file, knowing which voxel corresponds to each organ. The pixel and voxel dose data values are stored with the X coordinate incrementing first, the Y coordinate incrementing second, and the Z coordinate incrementing last.

The program also reports the simulated images and the dose at the Z plane at the level of the x ray source as ASCII text files. The ASCII output can be readily visualized with the GNUPLOT scripts distributed with MC-GPU. The header section at the beginning of these text files provides the information required to easily read the RAW binary files with IMAGEJ, OCTAVE or other programs.

1.5 Code compilation and execution

MC-GPU has been developed and tested only in the Linux operating system. A Makefile script is provided to compile the MC-GPU code in Linux. The CUDA libraries and the GNU GCC compiler must be previously installed. The Makefile may have to be edited to modify the library path. The code requires the "zlib.h" library to be able to open gzipped input files.

MC-GPU uses CUDA to access NVIDIA GPUs but all the actual computations are coded in standard C and the CUDA-specific commands are enclosed within preprocessor "if" statements. Defining the pre-processor variable "USING_CUDA" (i.e., compiling with "-DUSING_CUDA") the particle transport routines are compiled to simulate many x ray histories in parallel in an NVIDIA GPU using CUDA. Otherwise, the code is sequentially executed in the CPU. The same coding approach has been used to allow the use of multiple GPUs. Defining the pre-processor variable "USING_MPI" (i.e., compiling with "-DUSING_MPI"), Message Passing Interface (MPI) library calls are used to share information between multiple CPU threads in different computers. Each MPI thread gets a unique id in the CPU and addresses a unique GPU. At the end of the simulation the images and doses tallied by the different GPUs are reduced to form single output file equivalent to a sequential simulation of the same number of particles.

The code can be easily compiled executing the command "make" or running the provided "./make.sh" script. Optionally, the code can be executed from the command line with a command like this (example using CUDA and MPI, openMPI library in this case):

```
nvcc -DUSING_CUDA -DUSING_MPI MC-GPU_v1.3.cu -o MC-GPU_v1.3.x -O3
 -use_fast_math -L/usr/lib/ -I. -I/usr/local/cuda/include
 -I/usr/local/cuda/samples/common/inc -I/usr/local/cuda/samples/shared/inc/
 -I/usr/include/openmpi  -lmpi -lz --ptxas-options=-v
 -gencode=arch=compute_20,code=sm_20 -gencode=arch=compute_30,code=sm_30
```

The same source code can also be compiled for a regular CPU using:

```
gcc -x c -O3 MC-GPU_v1.3.cu -o MC-GPU_v1.3_CPU.x -I./ -lm -lz
```

To run a simulation (and keep the information reported to the standard output in an external file) the compiled code can be executed as:

```
./MC-GPU_v1.3.x MC-GPU_v1.3.in | tee MC-GPU_v1.3.out
```

All simulation can be executed in the same way using the code compiled for the CPU or the GPU (however, the number of histories should be reduced for the CPU to finish the simulation in a reasonable time). To run the simulation in parallel with MPI in multiple GPUs (or CPU cores) in the current computer the user can execute:

```
mpirun -n 4 ./MC-GPU_v1.3.x MC-GPU_v1.3.in
```

To use GPUs in different computers, the user must make sure all computers can access the simulation files and that the libraries are correctly set up in all nodes. To execute a simulation (with verbose MPI information being reported):

```
mpirun --tag-output -v -x LD_LIBRARY_PATH -hostfile myhostfile.txt -n 8
/fullPath/MC-GPU_v1.3.x /fullPath/MC-GPU_v1.3.in | tee MC-GPU_v1.3.out
```

The text file 'hostfile' lists the IP addresses and number of computing slots (GPUs) of the computers collaborating in the simulation. This file is not necessary when using multiple GPUs in a single workstation. When using multiple computers, the simulation files should be located in a shared drive to make sure every node can access the input data. The different workstations must have different host names in order to be differentiated by the MPI threads. The multiple threads communicate to each other to make sure they don't use the same GPU in the same workstation.

1.6 Known issues

In extremely long simulations, it is theoretically possible to cause an overflow of the counters estimating the mean and standard deviation of the material or voxel doses. If this happen, the results will be incorrect and even negative or nan values can be reported.

1.7 References

1. A. Badal and A. Badano, Accelerating Monte Carlo simulations of photon transport in a voxelized geometry using a massively parallel Graphics Processing Unit, Med. Phys. 36, p. 4878-4880 (2009)

2. A. Badal and A. Badano, Monte Carlo Simulation of X-Ray Imaging Using a - Graphics Processing Unit, IEEE NSC-MIC, Conference Record , HP3–1, p. 4081-4084 (2009)

3. A. Badal, I. Kyprianou, D. Sharma and A. Badano, Fast cardiac CT simulation using a Graphics Processing Unit-accelerated Monte Carlo code, Proc. SPIE Medical Imaging Conference 7622, p. 762231 (2010)

4. A. Badal and A. Badano, Fast Simulation of Radiographic Images Using a Monte Carlo X-Ray Transport Algorithm Implemented in CUDA, Chapter 50 of GPU - Computing Gems (Emerald Edition), p. 813-830, editor Wen-mei W. Hwu, publisher Morgan Kaufmann (Elsevier), Burlington MA, 2010

5. F. Salvat, J. M. Fernandez-Varea and J. Sempau, PENELOPE – A code system for Monte Carlo simulation of electron and photon transport, NEA-OECD, Issy-les-Moulineaux, available at www.nea.fr/html/dbprog/peneloperef.html (2006)

6. NVIDIA Corporation, NVIDIA CUDA(TM) Programming Guide, Technical Report available at www.nvidia.com/cuda (2011)

7. A. Badal and J. Sempau, A package of Linux scripts for the parallelization of Monte Carlo simulations, Comput. Phys. Commun. 175 (6), p. 440-450 (2006)