# MC-GPU

Generated by Doxygen 1.7.6.1

Mon Dec 17 2012 13:26:39

# Contents

# 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

## 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)

# Chapter 2

# Class Index

## 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# Chapter 3

# File Index

## 3.1 File List

Here is a list of all files with brief descriptions:

# Chapter 4

# Class Documentation

## 4.1 compton_struct Struct Reference

Structure storing the data of the Compton interaction sampling model (equivalent to PENELOPE's common block /CGCO/).

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- float **fco** [**MAX_MATERIALS** $*$**MAX_SHELLS**]
- float **uico** [**MAX_MATERIALS** $*$**MAX_SHELLS**]
- float **fj0** [**MAX_MATERIALS** $*$**MAX_SHELLS**]
- int **noscco** [**MAX_MATERIALS**]

### 4.1.1 Detailed Description

Structure storing the data of the Compton interaction sampling model (equivalent to PENELOPE's common block /CGCO/).

Definition at line 235 of file MC-GPU_v1.3.h.

### 4.1.2 Member Data Documentation

#### 4.1.2.1 float compton_struct::fco[MAX_MATERIALS $*$MAX_SHELLS]

Definition at line 241 of file MC-GPU_v1.3.h.

Referenced by GCOa(), and load_material().

#### 4.1.2.2 float compton_struct::fj0[MAX_MATERIALS ∗MAX_SHELLS]

Definition at line 241 of file MC-GPU_v1.3.h.

Referenced by GCOa(), and load_material().

#### 4.1.2.3 int compton_struct::noscco[MAX_MATERIALS]

Definition at line 244 of file MC-GPU_v1.3.h.

Referenced by GCOa(), and load_material().

#### 4.1.2.4 float compton_struct::uico[MAX_MATERIALS ∗MAX_SHELLS]

Definition at line 241 of file MC-GPU_v1.3.h.

Referenced by GCOa(), and load_material().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.2 detector_struct Struct Reference

Structure storing the data defining the x-ray detector.

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- float **sdd**
- **float3 corner_min_rotated_to_Y**
- **float3 center**
- float **rot_inv** [9]
- float **width_X**
- float **height_Z**
- float **inv_pixel_size_X**
- float **inv_pixel_size_Z**
- **int2 num_pixels**
- int **total_num_pixels**
- int **rotation_flag**

### 4.2.1 Detailed Description

Structure storing the data defining the x-ray detector.

For a CT, the struct stores for each angle the detector location and the rotations to transport the detector to a plane perpendicular to +Y. To simulate multiple projections, an array of MAX_NUM_PROJECTIONS of instances of this struct have to be defined! !!DeBuG!! shared

Definition at line 188 of file MC-GPU_v1.3.h.

### 4.2.2 Member Data Documentation

#### 4.2.2.1 float3 detector_struct::center

Definition at line 195 of file MC-GPU_v1.3.h.

Referenced by read_input(), set_CT_trajectory(), and tally_image().

#### 4.2.2.2 float3 detector_struct::corner_min_rotated_to_Y

Definition at line 195 of file MC-GPU_v1.3.h.

Referenced by read_input(), set_CT_trajectory(), and tally_image().

#### 4.2.2.3 float detector_struct::height_Z

Definition at line 197 of file MC-GPU_v1.3.h.

Referenced by read_input(), and set_CT_trajectory().

#### 4.2.2.4 float detector_struct::inv_pixel_size_X

Definition at line 197 of file MC-GPU_v1.3.h.

Referenced by read_input(), report_image(), set_CT_trajectory(), and tally_image().

#### 4.2.2.5 float detector_struct::inv_pixel_size_Z

Definition at line 197 of file MC-GPU_v1.3.h.

Referenced by read_input(), report_image(), set_CT_trajectory(), and tally_image().

#### 4.2.2.6 int2 detector_struct::num_pixels

Definition at line 202 of file MC-GPU_v1.3.h.

Referenced by main(), read_input(), report_image(), set_CT_trajectory(), and tally_-image().

#### 4.2.2.7 float detector_struct::rot_inv[9]

Definition at line 197 of file MC-GPU_v1.3.h.

Referenced by read_input(), set_CT_trajectory(), and tally_image().

#### 4.2.2.8 int detector_struct::rotation_flag

Definition at line 203 of file MC-GPU_v1.3.h.

Referenced by read_input(), set_CT_trajectory(), source(), and tally_image().

#### 4.2.2.9 float detector_struct::sdd

Definition at line 194 of file MC-GPU_v1.3.h.

Referenced by main(), read_input(), and set_CT_trajectory().

#### 4.2.2.10 int detector_struct::total_num_pixels

Definition at line 203 of file MC-GPU_v1.3.h.

Referenced by read_input(), set_CT_trajectory(), and tally_image().

#### 4.2.2.11 float detector_struct::width_X

Definition at line 197 of file MC-GPU_v1.3.h.

Referenced by read_input(), and set_CT_trajectory().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.3 double2 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- double **x**
- double **y**

### 4.3.1 Detailed Description

Definition at line 134 of file MC-GPU_v1.3.h.

### 4.3.2 Member Data Documentation

#### 4.3.2.1 double double2::x

Definition at line 134 of file MC-GPU_v1.3.h.

#### 4.3.2.2 double double2::y

Definition at line 134 of file MC-GPU_v1.3.h.

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.4 double3 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- double **x**
- double **y**
- double **z**

### 4.4.1 Detailed Description

Definition at line 135 of file MC-GPU_v1.3.h.

### 4.4.2 Member Data Documentation

#### 4.4.2.1 double **double3::x**

Definition at line 135 of file MC-GPU_v1.3.h.

#### 4.4.2.2 double **double3::y**

Definition at line 135 of file MC-GPU_v1.3.h.

#### 4.4.2.3 double **double3::z**

Definition at line 135 of file MC-GPU_v1.3.h.

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.5 float2 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- float **x**
- float **y**

### 4.5.1 Detailed Description

Definition at line 132 of file MC-GPU_v1.3.h.

### 4.5.2 Member Data Documentation

#### 4.5.2.1 float **float2::x**

Definition at line 132 of file MC-GPU_v1.3.h.

Referenced by track_particles().

#### 4.5.2.2 float **float2::y**

Definition at line 132 of file MC-GPU_v1.3.h.

Referenced by report_voxels_dose(), and track_particles().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.6 float3 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- float **x**
- float **y**
- float **z**

### 4.6.1 Detailed Description

Definition at line 133 of file MC-GPU_v1.3.h.

### 4.6.2 Member Data Documentation

#### 4.6.2.1 float **float3::x**

Definition at line 133 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), move_to_bbox(), read_input(), report_image(), report_voxels_dose(), rotate_double(), set_CT_trajectory(), source(), tally_image(), and track_particles().

#### 4.6.2.2 float **float3::y**

Definition at line 133 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), move_to_bbox(), read_input(), report_image(), report_voxels_dose(), rotate_double(), set_CT_trajectory(), source(), tally_image(), and track_particles().

#### 4.6.2.3 float float3::z

Definition at line 133 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), move_to_bbox(), read_input(), report_image(), report_voxels_dose(), rotate_double(), set_CT_trajectory(), source(), tally_image(), and track_particles().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.7 int2 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- int **x**
- int **y**

### 4.7.1 Detailed Description

Definition at line 130 of file MC-GPU_v1.3.h.

### 4.7.2 Member Data Documentation

#### 4.7.2.1 int int2::x

Definition at line 130 of file MC-GPU_v1.3.h.

Referenced by init_PRNG(), main(), ranecu(), ranecu_double(), read_input(), report_-image(), and tally_image().

#### 4.7.2.2 int int2::y

Definition at line 130 of file MC-GPU_v1.3.h.

Referenced by init_PRNG(), main(), ranecu(), ranecu_double(), read_input(), report_-image(), and tally_image().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.8 int3 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- int **x**
- int **y**
- int **z**

### 4.8.1 Detailed Description

Definition at line 131 of file MC-GPU_v1.3.h.

### 4.8.2 Member Data Documentation

#### 4.8.2.1 int int3::x

Definition at line 131 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), and report_voxels_dose().

#### 4.8.2.2 int int3::y

Definition at line 131 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), and report_voxels_dose().

#### 4.8.2.3 int int3::z

Definition at line 131 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), and main().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.9 linear_interp Struct Reference

Structure with the basic data required by the linear interpolation of the mean free paths: number of values and energy grid.

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- int **num_values**
- float **e0**
- float **ide**

### 4.9.1 Detailed Description

Structure with the basic data required by the linear interpolation of the mean free paths: number of values and energy grid.

Definition at line 222 of file MC-GPU_v1.3.h.

### 4.9.2 Member Data Documentation

#### 4.9.2.1 float linear_interp::e0

Definition at line 229 of file MC-GPU_v1.3.h.

Referenced by load_material(), main(), and track_particles().

#### 4.9.2.2 float linear_interp::ide

Definition at line 229 of file MC-GPU_v1.3.h.

Referenced by load_material(), main(), and track_particles().

#### 4.9.2.3 int linear_interp::num_values

Definition at line 228 of file MC-GPU_v1.3.h.

Referenced by load_material(), and main().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.10 rayleigh_struct Struct Reference

Structure storing the data of the Rayleigh interaction sampling model (equivalent to P-ENELOPE's common block /CGRA/).

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- float **xco** [**NP_RAYLEIGH** ∗**MAX_MATERIALS**]
- float **pco** [**NP_RAYLEIGH** ∗**MAX_MATERIALS**]
- float **aco** [**NP_RAYLEIGH** ∗**MAX_MATERIALS**]
- float **bco** [**NP_RAYLEIGH** ∗**MAX_MATERIALS**]
- float **pmax** [**MAX_ENERGYBINS_RAYLEIGH** ∗**MAX_MATERIALS**]
- unsigned char **itlco** [**NP_RAYLEIGH** ∗**MAX_MATERIALS**]
- unsigned char **ituco** [**NP_RAYLEIGH** ∗**MAX_MATERIALS**]

### 4.10.1 Detailed Description

Structure storing the data of the Rayleigh interaction sampling model (equivalent to P-ENELOPE's common block /CGRA/).

Definition at line 248 of file MC-GPU_v1.3.h.

### 4.10.2 Member Data Documentation

#### 4.10.2.1 float rayleigh_struct::aco[NP_RAYLEIGH ∗MAX_MATERIALS]

Definition at line 254 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

#### 4.10.2.2 float rayleigh_struct::bco[NP_RAYLEIGH ∗MAX_MATERIALS]

Definition at line 254 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

#### 4.10.2.3 unsigned char rayleigh_struct::itlco[NP_RAYLEIGH ∗MAX_MATERIALS]

Definition at line 259 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

#### 4.10.2.4 unsigned char **rayleigh_struct::ituco[NP_RAYLEIGH ∗MAX_MATERIALS]**

Definition at line 259 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

#### 4.10.2.5 float **rayleigh_struct::pco[NP_RAYLEIGH ∗MAX_MATERIALS]**

Definition at line 254 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

#### 4.10.2.6 float **rayleigh_struct::pmax[MAX_ENERGYBINS_RAYLEIGH ∗MAX_MATERIALS]**

Definition at line 254 of file MC-GPU_v1.3.h.

Referenced by load_material(), and track_particles().

#### 4.10.2.7 float **rayleigh_struct::xco[NP_RAYLEIGH ∗MAX_MATERIALS]**

Definition at line 254 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.11 short3 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- short **x**
- short **y**
- short **z**

### 4.11.1 Detailed Description

Definition at line 136 of file MC-GPU_v1.3.h.

### 4.11.2 Member Data Documentation

#### 4.11.2.1 short short3::x

Definition at line 136 of file MC-GPU_v1.3.h.

Referenced by locate_voxel(), and tally_voxel_energy_deposition().

#### 4.11.2.2 short short3::y

Definition at line 136 of file MC-GPU_v1.3.h.

Referenced by locate_voxel(), and tally_voxel_energy_deposition().

#### 4.11.2.3 short short3::z

Definition at line 136 of file MC-GPU_v1.3.h.

Referenced by locate_voxel(), and tally_voxel_energy_deposition().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.12 source_energy_struct Struct Reference

Structure storing the source energy spectrum data to be sampled using the Walker aliasing algorithm.

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- int **num_bins_espc**
- float **espc** [**MAX_ENERGY_BINS**]
- float **espc_cutoff** [**MAX_ENERGY_BINS**]
- short int **espc_alias** [**MAX_ENERGY_BINS**]

### 4.12.1 Detailed Description

Structure storing the source energy spectrum data to be sampled using the Walker aliasing algorithm.

Definition at line 171 of file MC-GPU_v1.3.h.

### 4.12.2 Member Data Documentation

#### 4.12.2.1 float source_energy_struct::espc[MAX_ENERGY_BINS]

Definition at line 178 of file MC-GPU_v1.3.h.

Referenced by init_energy_spectrum(), main(), and source().

#### 4.12.2.2 short int source_energy_struct::espc_alias[MAX_ENERGY_BINS]

Definition at line 180 of file MC-GPU_v1.3.h.

Referenced by init_energy_spectrum(), and source().

#### 4.12.2.3 float source_energy_struct::espc_cutoff[MAX_ENERGY_BINS]

Definition at line 178 of file MC-GPU_v1.3.h.

Referenced by init_energy_spectrum(), and source().

#### 4.12.2.4 int source_energy_struct::num_bins_espc

Definition at line 177 of file MC-GPU_v1.3.h.

Referenced by init_energy_spectrum(), main(), and source().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.13 source_struct Struct Reference

Structure storing the data defining the source model (except for the energy spectrum).

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- **float3 position**
- **float3 direction**
- float **rot_fan** [9]
- float **cos_theta_low**
- float **phi_low**

- float **D_cos_theta**
- float **D_phi**
- float **max_height_at_y1cm**

### 4.13.1 Detailed Description

Structure storing the data defining the source model (except for the energy spectrum).

When a CT is simulated, multiple sources will be stored in an array (one source instance per projection angle).

Definition at line 153 of file MC-GPU_v1.3.h.

### 4.13.2 Member Data Documentation

#### 4.13.2.1 float source_struct::cos_theta_low

Definition at line 161 of file MC-GPU_v1.3.h.

Referenced by read_input(), and source().

#### 4.13.2.2 float source_struct::D_cos_theta

Definition at line 161 of file MC-GPU_v1.3.h.

Referenced by source().

#### 4.13.2.3 float source_struct::D_phi

Definition at line 161 of file MC-GPU_v1.3.h.

Referenced by source().

#### 4.13.2.4 float3 source_struct::direction

Definition at line 159 of file MC-GPU_v1.3.h.

Referenced by main(), read_input(), report_image(), set_CT_trajectory(), and tally_-image().

#### 4.13.2.5 float source_struct::max_height_at_y1cm

Definition at line 161 of file MC-GPU_v1.3.h.

Referenced by source().

#### 4.13.2.6 float source_struct::phi_low

Definition at line 161 of file MC-GPU_v1.3.h.

Referenced by source().

#### 4.13.2.7 float3 source_struct::position

Definition at line 159 of file MC-GPU_v1.3.h.

Referenced by main(), read_input(), report_image(), set_CT_trajectory(), and source().

#### 4.13.2.8 float source_struct::rot_fan[9]

Definition at line 161 of file MC-GPU_v1.3.h.

Referenced by source().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.14 ulonglong2 Struct Reference

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- unsigned long long int **x**
- unsigned long long int **y**

### 4.14.1 Detailed Description

Definition at line 137 of file MC-GPU_v1.3.h.

### 4.14.2 Member Data Documentation

#### 4.14.2.1 unsigned long long int ulonglong2::x

Definition at line 137 of file MC-GPU_v1.3.h.

Referenced by main(), report_voxels_dose(), tally_materials_dose(), and tally_voxel_-energy_deposition().

#### 4.14.2.2 unsigned long long int ulonglong2::y

Definition at line 137 of file MC-GPU_v1.3.h.

Referenced by main(), tally_materials_dose(), and tally_voxel_energy_deposition().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

## 4.15 voxel_struct Struct Reference

Structure defining a voxelized box with the back-lower corner at the coordinate origin.

```
#include <MC-GPU_v1.3.h>
```

### Public Attributes

- **int3 num_voxels**
- **float3 inv_voxel_size**
- **float3 size_bbox**

### 4.15.1 Detailed Description

Structure defining a voxelized box with the back-lower corner at the coordinate origin.

Definition at line 209 of file MC-GPU_v1.3.h.

### 4.15.2 Member Data Documentation

#### 4.15.2.1 float3 voxel_struct::inv_voxel_size

Definition at line 216 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), and report_voxels_dose().

#### 4.15.2.2 int3 voxel_struct::num_voxels

Definition at line 215 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), main(), and report_voxels_dose().

#### 4.15.2.3 float3 voxel_struct::size_bbox

Definition at line 216 of file MC-GPU_v1.3.h.

Referenced by load_voxels(), locate_voxel(), and source().

The documentation for this struct was generated from the following file:

- **MC-GPU_v1.3.h**

# Chapter 5

# File Documentation

## 5.1 MC-GPU_kernel_v1.3.cu File Reference

Definition of the CUDA GPU kernel for the simulation of x ray tracks in a voxelized geometry.

### Defines

- #define **LEAP_DISTANCE** 256

  *Upper limit of the number of random values sampled in a single track.*
- #define **a1_RANECU** 40014

  *Multipliers and moduli for the two MLCG in RANECU.*
- #define **m1_RANECU** 2147483563
- #define **a2_RANECU** 40692
- #define **m2_RANECU** 2147483399

### Functions

- void **track_particles** (int history_batch, int histories_per_thread, int num_p, int seed_input, unsigned long long int ∗image, **ulonglong2** ∗voxels_Edep, **float2** ∗voxel_mat_dens, **float2** ∗mfp_Woodcock_table, **float3** ∗mfp_table_a, **float3** ∗mfp_table_b, struct **rayleigh_struct** ∗rayleigh_table, struct **compton_struct** ∗compton_table, struct **detector_struct** ∗detector_data_array, struct **source_-struct** ∗source_data_array, **ulonglong2** ∗materials_dose)

  *Initialize the image array, ie, set all pixels to zero Essentially, this function has the same effect as the command: "cutilSafeCall(cudaMemcpy(image_device, image, image_-bytes, cudaMemcpyHostToDevice))";.*

- void **tally_voxel_energy_deposition** (float ∗Edep, **short3** ∗voxel_coord, **ulong-long2** ∗voxels_Edep)

    *Tally the dose deposited in the voxels.*
- void **tally_image** (float ∗energy, **float3** ∗position, **float3** ∗direction, signed char ∗scatter_state, unsigned long long int ∗image, struct **source_struct** ∗source_-data_SHARED, struct **detector_struct** ∗detector_data_SHARED)

    *Tally a radiographic projection image.*
- void **source** (**float3** ∗position, **float3** ∗direction, float ∗energy, **int2** ∗seed, int ∗absvox, struct **source_struct** ∗source_data_SHARED, struct **detector_struct** ∗detector_data_SHARED)

    *Source that creates primary x rays, according to the defined source model.*
- void **move_to_bbox** (**float3** ∗position, **float3** ∗direction, **float3** size_bbox, int ∗intersection_flag)

    *Functions that moves a particle inside the voxelized geometry bounding box.*
- void **init_PRNG** (int history_batch, int histories_per_thread, int seed_input, **int2** ∗seed)

    *Initialize the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).*
- int **abMODm** (int m, int a, int s)

    *Calculate "(a1∗a2) MOD m" with 32-bit integers and avoiding the possible overflow, using the Russian Peasant approach modulo m and the approximate factoring method, as described in: L'Ecuyer and Cote, ACM Trans.*
- float **ranecu** (**int2** ∗seed)

    *Pseudo-random number generator (PRNG) RANECU returning a float value (single precision version).*
- double **ranecu_double** (**int2** ∗seed)

    *Pseudo-random number generator (PRNG) RANECU returning a double value.*
- int **locate_voxel** (**float3** ∗position, **short3** ∗voxel_coord)

    *Find the voxel that contains the current position.*
- void **rotate_double** (**float3** ∗direction, double costh, double phi)

    *Rotates a vector; the rotation is specified by giving the polar and azimuthal angles in the "self-frame", as determined by the vector to be rotated.*
- void **GRAa** (float ∗energy, double ∗costh_Rayleigh, int ∗mat, float ∗pmax_-current, **int2** ∗seed, struct **rayleigh_struct** ∗cgra)

    *Sample a Rayleigh interaction using the sampling algorithm used in PENELOPE 2006.*
- void **GCOa** (float ∗energy, double ∗costh_Compton, int ∗mat, **int2** ∗seed, struct **compton_struct** ∗cgco_SHARED)

    *Random sampling of incoherent (Compton) scattering of photons, using the sampling algorithm from PENELOPE 2006: Relativistic impulse approximation with analytical one-electron Compton profiles.*
- void **tally_materials_dose** (float ∗Edep, int ∗material, **ulonglong2** ∗materials_-dose)

    *Tally the depose deposited inside each material.*

### 5.1.1 Detailed Description

Definition of the CUDA GPU kernel for the simulation of x ray tracks in a voxelized geometry. This kernel has been optimized to yield a good performance in the GPU but can still be compiled in the CPU without problems. All the CUDA especific commands are enclosed in pre-processor directives that are skipped if the parameter "USING_CUDA" is not defined at compilation time.

**Author**

Andreu Badal (`Andreu.Badal-Soler@fda.hhs.gov`)

**Date**

2012/12/12

Definition in file **MC-GPU_kernel_v1.3.cu**.

### 5.1.2 Define Documentation

#### 5.1.2.1 #define a1_RANECU 40014

Multipliers and moduli for the two MLCG in RANECU.

Definition at line 814 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG(), and update_seed_PRNG().

#### 5.1.2.2 #define a2_RANECU 40692

Definition at line 816 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG().

#### 5.1.2.3 #define LEAP_DISTANCE 256

Upper limit of the number of random values sampled in a single track.

Definition at line 812 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG(), and update_seed_PRNG().

#### 5.1.2.4 #define m1_RANECU 2147483563

Definition at line 815 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG(), and update_seed_PRNG().

#### 5.1.2.5 #define m2_RANECU 2147483399

Definition at line 817 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG().

### 5.1.3 Function Documentation

#### 5.1.3.1 int abMODm ( int *m,* int *a,* int *s* ) [inline]

Calculate "(a1∗a2) MOD m" with 32-bit integers and avoiding the possible overflow, using the Russian Peasant approach modulo m and the approximate factoring method, as described in: L'Ecuyer and Cote, ACM Trans.

Math. Soft. 17 (1991).

This function has been adapted from "seedsMLCG.f", see: Badal and Sempau, - Computer Physics Communications 175 (2006)

**Parameters**

| | | |
|---|---|---|
| in | *m,a,s* | MLCG parameters |

**Returns**

(a1∗a2) MOD m

Definition at line 919 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG(), and update_seed_PRNG().

#### 5.1.3.2 void GCOa ( float ∗ *energy,* double ∗ *costh_Compton,* int ∗ *mat,* int2 ∗ *seed,* struct compton_struct ∗ *cgco_SHARED* ) [inline]

Random sampling of incoherent (Compton) scattering of photons, using the sampling algorithm from PENELOPE 2006: Relativistic impulse approximation with analytical one-electron Compton profiles.

**Parameters**

| | | |
|---|---|---|
| in,out | *energy* | incident and final photon energy (eV) |
| out | *costh_-Compton* | cosine of the polar scattering angle |
| in | *material* | Current voxel material |
| in | *seed* | RANECU PRNG seed |

Definition at line 1287 of file MC-GPU_kernel_v1.3.cu.

References compton_struct::fco, compton_struct::fj0, MAX_MATERIALS, MAX_SHE-LLS, max_value, min_value, compton_struct::noscco, ranecu(), and compton_struct-::uico.

Referenced by track_particles().

#### 5.1.3.3 void **GRAa** ( float ∗ *energy,* double ∗ *costh_Rayleigh,* int ∗ *mat,* float ∗ *pmax_current,* int2 ∗ *seed,* struct **rayleigh_struct** ∗ *cgra* ) `[inline]`

Sample a Rayleigh interaction using the sampling algorithm used in PENELOPE 2006.

**Parameters**

| | | |
|---|---|---|
| `in` | *energy* | Particle energy (not modified with Rayleigh) |
| `out` | *costh_-Rayleigh* | Cosine of the angular deflection |
| `in` | *material* | Current voxel material |

Definition at line 1181 of file MC-GPU_kernel_v1.3.cu.

References rayleigh_struct::aco, rayleigh_struct::bco, rayleigh_struct::itlco, rayleigh_-struct::ituco, min_value, NP_RAYLEIGH, rayleigh_struct::pco, ranecu_double(), and rayleigh_struct::xco.

Referenced by track_particles().

#### 5.1.3.4 void **init_PRNG** ( int *history_batch,* int *histories_per_thread,* int *seed_input,* int2 ∗ *seed* ) `[inline]`

Initialize the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).

Each calculated seed initiates a consecutive and disjoint sequence of pseudo-random numbers with length LEAP_DISTANCE, that can be used to in a parallel simulation (-Sequence Splitting parallelization method). The basic equation behind the algorithm is: S(i+j) = (a∗∗j ∗ S(i)) MOD m = [(a∗∗j MOD m)∗S(i)] MOD m , which is described in: P L'Ecuyer, Commun. ACM 31 (1988) p.742

This function has been adapted from "seedsMLCG.f", see: A Badal and J Sempau, Computer Physics Communications 175 (2006) p. 440-450

**Parameters**

| | | |
|---|---|---|
| `in` | *history* | Particle bach number. |
| `in` | *seed_input* | Initial PRNG seed input (used to initiate both MLCGs in RA-NECU). |
| `out` | *seed* | Initial PRNG seeds for the present history. |

Definition at line 841 of file MC-GPU_kernel_v1.3.cu.

References a1_RANECU, a2_RANECU, abMODm(), LEAP_DISTANCE, m1_RANECU, m2_RANECU, int2::x, and int2::y.

Referenced by track_particles().

#### 5.1.3.5 int **locate_voxel** ( **float3** ∗ ***position,*** **short3** ∗ ***voxel_coord*** ) `[inline]`

Find the voxel that contains the current position.

Report the voxel absolute index and the x,y,z indices. The structure containing the voxel number and size is read from CONSTANT memory.

**Parameters**

| | | |
|---|---|---|
| `in` | *position* | Particle position |
| `out` | *voxel_coord* | Pointer to three integer values (short3∗) that will store the x,y and z voxel indices. |

**Returns**

Returns "absvox", the voxel number where the particle is located (negative if position outside the voxel bbox).

Definition at line 1033 of file MC-GPU_kernel_v1.3.cu.

References EPS_SOURCE, voxel_struct::inv_voxel_size, voxel_struct::num_voxels, voxel_struct::size_bbox, voxel_data_CONST, int3::x, float3::x, short3::x, int3::y, float3::y, short3::y, float3::z, and short3::z.

Referenced by track_particles().

#### 5.1.3.6 void **move_to_bbox** ( **float3** ∗ ***position,*** **float3** ∗ ***direction,*** **float3** ***size_bbox,*** **int** ∗ ***intersection_flag*** ) `[inline]`

Functions that moves a particle inside the voxelized geometry bounding box.

An EPSILON distance is added to make sure the particles will be clearly inside the bbox, not exactly on the surface.

This algorithm makes the following assumtions:

- The back lower vertex of the voxel bounding box is always located at the origin: (x0,y0,z0)=(0,0,0).
- The initial value of "position" corresponds to the focal spot location.

- When a ray is not pointing towards the bbox plane that it should cross according to the sign of the direction, I assign a distance to the intersection =0 instead of the real negative distance. The wall that will be crossed to enter the bbox is always the furthest and therefore a 0 distance will never be used except in the case of a ray starting inside the bbox or outside the bbox and not pointing to any of the 3 planes. In this situation the ray will be transported a 0 distance, meaning that it will stay at the focal spot.

(Interesting information on ray-box intersection: `http://tog.acm.org/resources/-GraphicsGems/gems/RayBox.c`)

**Parameters**

| | | |
|---|---|---|
| `in,out` | *position* | Particle position: initially set to the focal spot, returned transported inside the voxel bbox. |
| `out` | *direction* | Sampled particle direction (cosine vectors). |
| `out` | *intersection-_flag* | Set to $<0$ if particle outside bbox and will not cross the voxels, not changed otherwise. |
| `out` | *size_bbox* | Size of the bounding box. |

Definition at line 714 of file MC-GPU_kernel_v1.3.cu.

References EPS_SOURCE, NEG_EPS_SOURCE, NEG_INF, float3::x, float3::y, and float3::z.

Referenced by source().

#### 5.1.3.7 float ranecu ( int2 ∗ *seed* ) `[inline]`

Pseudo-random number generator (PRNG) RANECU returning a float value (single precision version).

**Parameters**

| | | |
|---|---|---|
| `in,out` | *seed* | PRNG seed (seed kept in the calling function and updated here). |

**Returns**

PRN double value in the open interval (0,1)

Definition at line 965 of file MC-GPU_kernel_v1.3.cu.

References int2::x, and int2::y.

Referenced by GCOa(), source(), and track_particles().

#### 5.1.3.8 double **ranecu_double** ( **int2** ∗ *seed* ) `[inline]`

Pseudo-random number generator (PRNG) RANECU returning a double value.

Definition at line 995 of file MC-GPU_kernel_v1.3.cu.

References int2::x, and int2::y.

Referenced by GRAa(), and track_particles().

#### 5.1.3.9 void **rotate_double** ( **float3** ∗ *direction,* double *costh,* double *phi* ) `[inline]`

Rotates a vector; the rotation is specified by giving the polar and azimuthal angles in the "self-frame", as determined by the vector to be rotated.

This function is a literal translation from Fortran to C of PENELOPE (v. 2006) subroutine "DIRECT".

**Parameters**

| | | |
|---|---|---|
| `in,out` | *(u,v,w)* | input vector (=d) in the lab. frame; returns the rotated vector components in the lab. frame |
| `in` | *costh* | cos(theta), angle between d before and after turn |
| `in` | *phi* | azimuthal angle (rad) turned by d in its self-frame |

Definition at line 1103 of file MC-GPU_kernel_v1.3.cu.

References float3::x, float3::y, and float3::z.

Referenced by track_particles().

#### 5.1.3.10 void **source** ( **float3** ∗ *position,* **float3** ∗ *direction,* float ∗ *energy,* **int2** ∗ *seed,* int ∗ *absvox,* struct **source_struct** ∗ *source_data_SHARED,* struct **detector_struct** ∗ *detector_data_SHARED* ) `[inline]`

Source that creates primary x rays, according to the defined source model.

The particles are automatically moved to the surface of the voxel bounding box, to start the tracking inside a real material. If the sampled particle do not enter the voxels, it is init in the focal spot and the main program will check if it arrives at the detector or not.

**Parameters**

| | | |
|---|---|---|
| `in` | *source_data* | Structure describing the source. |
| `in` | *source_-energy_-data_CONS-T* | Global variable in constant memory space describing the source energy spectrum. |

| | | |
|---|---|---|
| out | *position* | Initial particle position (particle transported inside the voxel bbox). |
| out | *direction* | Sampled particle direction (cosine vectors). |
| out | *energy* | Sampled energy of the new x ray. |
| in | *seed* | Current seed of the random number generator, requiered to sample the movement direction. |
| out | *absvox* | Set to <0 if primary particle will not cross the voxels, not changed otherwise (>0). |

Definition at line 626 of file MC-GPU_kernel_v1.3.cu.

References source_struct::cos_theta_low, source_struct::D_cos_theta, source_struct::-D_phi, source_energy_struct::espc, source_energy_struct::espc_alias, source_energy-_struct::espc_cutoff, source_struct::max_height_at_y1cm, move_to_bbox(), source-_energy_struct::num_bins_espc, source_struct::phi_low, source_struct::position, ranecu(), source_struct::rot_fan, detector_struct::rotation_flag, voxel_struct::size_bbox, source_energy_data_CONST, voxel_data_CONST, float3::x, float3::y, and float3::z.

Referenced by track_particles().

#### 5.1.3.11 void **tally_image** ( float ∗ *energy,* **float3** ∗ *position,* **float3** ∗ *direction,* signed char ∗ *scatter_state,* unsigned long long int ∗ *image,* struct **source_struct** ∗ *source_data_SHARED,* struct **detector_struct** ∗ *detector_data_SHARED* ) [inline]

Tally a radiographic projection image.

This function is called whenever a particle escapes the voxelized volume. The code checks if the particle would arrive at the detector if it kept moving in a straight line after exiting the voxels (assuming vacuum enclosure). An ideal image formation model is implemented: each pixel counts the total energy of the x rays that enter the pixel (100% detection efficiency for any energy). The image due to primaries and different kinds of scatter is tallied separately.

In the GPU, and atomicAdd() function is used to make sure that multiple threads do not update the same pixel at the same time, which would result in a lose of information. Since the atomicAdd function is only available for 'unsigned long long int' data, the float pixel values are scaled by a factor "SCALE_eV" defined in the header file (eg, #define SCALE_eV 10000.0f) and stored as unsigned long long integers in main memory.

WARNING! If the total tallied signal (for all particles) is larger than "1.8e19/SCALE_eV", there will be a bit overflow and the value will be reset to 0 giving bogus results.

WARNING! The detector plane should be located outside the voxels bounding box. -However, since the particles are moved outside the bbox in the last step, they could cross the detector plane anyway. If the particles are less than 2.0 cm behind the detector, they are moved back and detected. Therefore the detector can be a few cm inside the bbox and still work. If the Woodcock mean free path is larger than the distance from the bbox to the detector, we may lose some particles behind the detector!

**Parameters**

| | | |
|---|---|---|
| in | *energy* | X-ray energy |
| in | *position* | Particle position |
| in | *direction* | Particle direction (cosine vectors) |
| in | *scatter_state* | Flag marking primaries, single Compton, single Rayleigh or multiple scattered radiation |
| out | *image* | Integer array containing the image, ie, the pixel values (in tenths of meV) |

Definition at line 482 of file MC-GPU_kernel_v1.3.cu.

References detector_struct::center, detector_struct::corner_min_rotated_to_Y, source-_struct::direction, detector_struct::inv_pixel_size_X, detector_struct::inv_pixel_size_Z, detector_struct::num_pixels, detector_struct::rot_inv, detector_struct::rotation_flag, SC-ALE_eV, detector_struct::total_num_pixels, int2::x, float3::x, int2::y, float3::y, and float3-::z.

Referenced by track_particles().

#### 5.1.3.12 void **tally_materials_dose** ( float ∗ *Edep,* int ∗ *material,* **ulonglong2** ∗ *materials_dose* ) [inline]

Tally the depose deposited inside each material.

This function is called whenever a particle suffers a Compton or photoelectric interaction. The energy released in each interaction is added and later in the report function the total deposited energy is divided by the total mass of the material in the voxelized object to get the dose. This naturally accounts for multiple densities for voxels with the same material (not all voxels have same mass). Electrons are not transported in MC--GPU and therefore we are approximating that the dose is equal to the KERMA (energy released by the photons alone). This approximation is acceptable when there is electronic equilibrium and when the range of the secondary electrons is shorter than the organ size.

The function uses atomic functions for a thread-safe access to the GPU memory. We can check if this tally was disabled in the input file checking if the array materials_dose was allocated in the GPU (disabled if pointer = NULL).

**Parameters**

| | | |
|---|---|---|
| in | *Edep* | Energy deposited in the interaction |
| in | *material* | Current material id number |
| out | *materials_-dose* | **ulonglong2** (p. 28) array storing the mateials dose [in eV/g] and dose^2 (ie, uncertainty). |

Definition at line 1547 of file MC-GPU_kernel_v1.3.cu.

References SCALE_eV, ulonglong2::x, and ulonglong2::y.

Referenced by track_particles().

#### 5.1.3.13 void **tally_voxel_energy_deposition** ( float ∗ ***Edep***, **short3** ∗ ***voxel_coord***, **ulonglong2** ∗ ***voxels_Edep*** ) `[inline]`

Tally the dose deposited in the voxels.

This function is called whenever a particle suffers a Compton or photoelectric interaction. It is not necessary to call this function if the dose tally was disabled in the input file (ie, dose_ROI_x_max_CONST $< 0$). Electrons are not transported in MC-GPU and therefore we are approximating that the dose is equal to the KERMA (energy released by the photons alone). This approximation is acceptable when there is electronic equilibrium and when the range of the secondary electrons is shorter than the voxel size. Usually the doses will be acceptable for photon energies below 1 MeV. The dose estimates may not be accurate at the interface of low density volumes.

We need to use atomicAdd() in the GPU to prevent that multiple threads update the same voxel at the same time, which would result in a lose of information. This is very improbable when using a large number of voxels but gives troubles with a simple geometries with few voxels (in this case the atomicAdd will slow down the code because threads will update the voxel dose secuentially).

**Parameters**

| | | |
|---|---|---|
| `in` | *Edep* | Energy deposited in the interaction |
| `in` | *voxel_coord* | Voxel coordinates, needed to check if particle located inside the input region of interest (ROI) |
| `out` | *voxels_Edep* | **ulonglong2** (p. 28) array containing the 3D voxel dose and dose^2 (ie, uncertainty) as unsigned integers scaled by SCALE_eV. |

Definition at line 418 of file MC-GPU_kernel_v1.3.cu.

References dose_ROI_x_max_CONST, dose_ROI_x_min_CONST, dose_ROI_y_max_CONST, dose_ROI_y_min_CONST, dose_ROI_z_max_CONST, dose_ROI_z_min_CONST, SCALE_eV, short3::x, ulonglong2::x, short3::y, ulonglong2::y, and short3::z.

Referenced by track_particles().

#### 5.1.3.14 void track_particles ( int *history_batch*, int *histories_per_thread*, int *num_p*, int *seed_input*, unsigned long long int ∗ *image*, ulonglong2 ∗ *voxels_Edep*, float2 ∗ *voxel_mat_dens*, float2 ∗ *mfp_Woodcock_table*, float3 ∗ *mfp_table_a*, float3 ∗ *mfp_table_b*, struct rayleigh_struct ∗ *rayleigh_table*, struct compton_struct ∗ *compton_table*, struct detector_struct ∗ *detector_data_array*, struct source_struct ∗ *source_data_array*, ulonglong2 ∗ *materials_dose* )

Initialize the image array, ie, set all pixels to zero Essentially, this function has the same effect as the command: "cutilSafeCall(cudaMemcpy(image_device, image, image_-bytes, cudaMemcpyHostToDevice))";.

CUDA performs some initialization work the first time a GPU kernel is called. Therefore, calling a short kernel before the real particle tracking is performed may improve the accuracy of the timing measurements in the relevant kernel.

**Parameters**

| | | |
|---|---|---|
| `in,out` | *image* | Pointer to the image array. |
| `in` | *pixels_per_-image* | Number of pixels in the image (ie, elements in the array). Main function to simulate x-ray tracks inside a voxelized geometry. Secondary electrons are not simulated (in photoelectric and Compton events the energy is locally deposited). |

The following global variables, in the GPU __constant__ memory are used: voxel_data-_CONST, source_energy_data_CONST, detector_data_CONST, mfp_table_data_CO-NST.

**Parameters**

| | | |
|---|---|---|
| `in` | *history_-batch* | Particle batch number (only used in the CPU version when CUDA is disabled!, the GPU uses the built-in variable threadIdx) |
| `in` | *num_p* | Projection number in the CT simulation. This variable defines a specific angle and the corresponding source and detector will be used. |
| `in` | *histories_-per_thread* | Number of histories to simulate for each call to this function (ie, for GPU thread). |
| `in` | *seed_input* | Random number generator seed (the same seed is used to initialize the two MLCGs of RANECU). |
| `in` | *voxel_mat_-dens* | Pointer to the voxel densities and material vector (the voxelized geometry), stored in GPU glbal memory. |
| `in` | *mfp_-Woodcock_-table* | Two parameter table for the linear interpolation of the -Woodcock mean free path (MFP) (stored in GPU global memory). |
| `in` | *mfp_table_a* | First element for the linear interpolation of the interaction mean free paths (stored in GPU global memory). |

| | | |
|---|---|---|
| in | *mfp_table_b* | Second element for the linear interpolation of the interaction mean free paths (stored in GPU global memory). |
| in | *rayleigh_-table* | Pointer to the table with the data required by the Rayleigh interaction sampling, stored in GPU global memory. |
| in | *compton_-table* | Pointer to the table with the data required by the Compton interaction sampling, stored in GPU global memory. |
| in,out | *image* | Pointer to the image vector in the GPU global memory. |
| in,out | *dose* | Pointer to the array containing the 3D voxel dose (and its uncertainty) in the GPU global memory. |

Definition at line 135 of file MC-GPU_kernel_v1.3.cu.

References dose_ROI_x_max_CONST, linear_interp::e0, GCOa(), GRAa(), linear_-interp::ide, init_PRNG(), locate_voxel(), MAX_MATERIALS, mfp_table_data_CONS-T, rayleigh_struct::pmax, ranecu(), ranecu_double(), rotate_double(), source(), tally-_image(), tally_materials_dose(), tally_voxel_energy_deposition(), float2::x, float3::x, float2::y, float3::y, and float3::z.

Referenced by main().

## 5.2 MC-GPU_v1.3.cu File Reference

### Functions

- int **main** (int argc, char ∗∗argv)

  *Main program of MC-GPU: initialize the simulation enviroment, launch the GPU kernels that perform the x ray transport and report the final results.*

- void **read_input** (int argc, char ∗∗argv, int myID, unsigned long long int ∗total_histories, int ∗seed_input, int ∗gpu_id, int ∗num_threads_per_block, int ∗histories_per_thread, struct **detector_struct** ∗detector_data, unsigned long long int ∗∗image_ptr, int ∗image_bytes, struct **source_struct** ∗source_data, struct **source_energy_struct** ∗source_energy_data, char ∗file_name_voxels, char file_name_materials[**MAX_MATERIALS**][250], char ∗file_name_output, char ∗file_name_espc, int ∗num_projections, double ∗D_angle, double ∗angular-ROI_0, double ∗angularROI_1, double ∗initial_angle, **ulonglong2** ∗∗voxels_-Edep_ptr, int ∗voxels_Edep_bytes, char ∗file_dose_output, short int ∗dose_R-OI_x_min, short int ∗dose_ROI_x_max, short int ∗dose_ROI_y_min, short int ∗dose_ROI_y_max, short int ∗dose_ROI_z_min, short int ∗dose_ROI_z_max, double ∗SRotAxisD, double ∗vertical_translation_per_projection, int ∗flag_-material_dose)

  *Read the input file given in the command line and return the significant data.*

- void **trim_name** (char ∗input_line, char ∗file_name)

*Extract a file name from an input text line, trimming the initial blanks, trailing comment (#) and stopping at the first blank (the file name should not contain blanks).*

- char ∗ **fgets_trimmed** (char ∗trimmed_line, int num, FILE ∗file_ptr)

  *Read a line of text and trim initial blancks and trailing comments (#).*
- void **load_voxels** (int myID, char ∗file_name_voxels, float ∗density_max, struct **voxel_struct** ∗voxel_data, **float2** ∗∗voxel_mat_dens_ptr, unsigned int ∗voxel_mat_dens_bytes, short int ∗dose_ROI_x_max, short int ∗dose_ROI_y_max, short int ∗dose_ROI_z_max)

  *Read the voxel data and allocate the material and density matrix.*
- void **load_material** (int myID, char file_name_materials[**MAX_MATERIALS**][250], float ∗density_max, float ∗density_nominal, struct **linear_interp** ∗mfp_table_data, **float2** ∗∗mfp_Woodcock_table_ptr, int ∗mfp_Woodcock_table_bytes, **float3** ∗∗mfp_table_a_ptr, **float3** ∗∗mfp_table_b_ptr, int ∗mfp_table_bytes, struct **rayleigh_struct** ∗rayleigh_table_ptr, struct **compton_struct** ∗compton_table_ptr)

  *Read the material input files and set the mean free paths and the "linear_interp" structures.*
- int **report_image** (char ∗file_name_output, struct **detector_struct** ∗detector_data, struct **source_struct** ∗source_data, float mean_energy_spectrum, unsigned long long int ∗image, double time_elapsed, unsigned long long int total_histories, int current_projection, int num_projections, double D_angle, double initial_angle, int myID, int numprocs)

  *Report the tallied image in ASCII and binary form (32-bit floats).*
- int **report_voxels_dose** (char ∗file_dose_output, int num_projections, struct **voxel_struct** ∗voxel_data, **float2** ∗voxel_mat_dens, **ulonglong2** ∗voxels_Edep, double time_total_MC_init_report, unsigned long long int total_histories, short int dose_ROI_x_min, short int dose_ROI_x_max, short int dose_ROI_y_min, short int dose_ROI_y_max, short int dose_ROI_z_min, short int dose_ROI_z_max, struct **source_struct** ∗source_data)

  *Report the total tallied 3D voxel dose deposition for all projections.*
- int **report_materials_dose** (int num_projections, unsigned long long int total_histories, float ∗density_nominal, **ulonglong2** ∗materials_dose, double ∗mass_materials)

  *Report the tallied dose to each material number, accounting for different densities in different regions with the same material number.*
- void **set_CT_trajectory** (int myID, int num_projections, double D_angle, double angularROI_0, double angularROI_1, double SRotAxisD, struct **source_struct** ∗source_data, struct **detector_struct** ∗detector_data, double vertical_translation_per_projection)

  *Sets the CT trajectory: store in memory the source and detector rotations that are needed to calculate the multiple projections.*
- void **update_seed_PRNG** (int batch_number, unsigned long long int total_histories, int ∗seed)

*Initialize the first seed of the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).*

- void **init_energy_spectrum** (char ∗file_name_espc, struct **source_energy_-struct** ∗source_energy_data, float ∗mean_energy_spectrum)

  *Read the energy spectrum file and initialize the Walker aliasing sampling.*
- int **seeki_walker** (float ∗cutoff, short int ∗alias, float randno, int n)

  *Finds the interval (x(i),x(i+1)] containing the input value using Walker's aliasing method.*
- void **IRND0** (float ∗W, float ∗F, short int ∗K, int N)

  *Initialisation of Walker's aliasing algorithm for random sampling from discrete probability distributions.*

### 5.2.1 Detailed Description

**Author**

Andreu Badal (`Andreu.Badal-Soler@fda.hhs.gov`)

**Date**

2012/12/12 -- MC-GPU v.1.3: 2012/12/12 -- MC-GPU v.1.2: 2011/10/25 -- MC-GPU v.1.1: 2010/06/25 -- MC-GPU v.1.0: 2009/03/17

Definition in file **MC-GPU_v1.3.cu**.

### 5.2.2 Function Documentation

#### 5.2.2.1 char∗ fgets_trimmed ( char ∗ *trimmed_line,* int *num,* FILE ∗ *file_ptr* )

Read a line of text and trim initial blancks and trailing comments (#).

**Parameters**

| | | |
|---|---|---|
| `in` | *num* | Characters to read |
| `in` | *file_ptr* | Pointer to the input file stream |
| `out` | *trimmed_line* | Trimmed line from input file, skipping empty lines and comments |

Definition at line 1868 of file MC-GPU_v1.3.cu.

Referenced by init_energy_spectrum(), and read_input().

#### 5.2.2.2 void **init_energy_spectrum** ( char ∗ *file_name_espc,* struct **source_energy_struct** ∗ *source_energy_data,* float ∗ *mean_energy_spectrum* )

Read the energy spectrum file and initialize the Walker aliasing sampling.

**Parameters**

| | | |
|---|---|---|
| `in` | *file_name_-espc* | File containing the energy spectrum (lower energy value in each bin and its emission probability). |
| `in,out` | *source_-energy_data* | Energy spectrum and other source data. The Walker alias and cutoffs are initialized in this function. |
| `out` | *mean_-energy_-spectrum* | Mean energy in the input x-ray energy spectrum. |

!Walker!! Calling PENELOPE's function to init the Walker method

Definition at line 3398 of file MC-GPU_v1.3.cu.

References source_energy_struct::espc, source_energy_struct::espc_alias, source_-energy_struct::espc_cutoff, fgets_trimmed(), IRND0(), MAX_ENERGY_BINS, max_-value, and source_energy_struct::num_bins_espc.

Referenced by main().

#### 5.2.2.3 void **IRND0** ( float ∗ *W,* float ∗ *F,* short int ∗ *K,* int *N* )

Initialisation of Walker's aliasing algorithm for random sampling from discrete probability distributions.

Input arguments: N ........ number of different values of the random variable. W(1:-N) ... corresponding point probabilities (not necessarily normalised to unity). Output arguments: F(1:N) ... cutoff values. K(1:N) ... alias values.

This subroutine is part of the PENELOPE 2006 code developed by Francesc Salvat at the University of Barcelona. For more info: www.oecd-nea.org/science/pubs/2009/nea6416-penelope.pdf

Definition at line 3575 of file MC-GPU_v1.3.cu.

Referenced by init_energy_spectrum().

#### 5.2.2.4 void **load_material** ( int *myID,* char *file_name_materials[MAX_MATERIALS][250],* float ∗ *density_max,* float ∗ *density_nominal,* struct **linear_interp** ∗ *mfp_table_data,* **float2** ∗∗ *mfp_Woodcock_table_ptr,* int ∗ *mfp_Woodcock_table_bytes,* **float3** ∗∗ *mfp_table_a_ptr,* **float3** ∗∗ *mfp_table_b_ptr,* int ∗ *mfp_table_bytes,* struct **rayleigh_struct** ∗ *rayleigh_table_ptr,* struct **compton_struct** ∗ *compton_table_ptr* )

Read the material input files and set the mean free paths and the "linear_interp" structures.

Find the material nominal density. Set the Woodcock trick data.

**Parameters**

| | | |
|---|---|---|
| `in` | *file_name_-materials* | Array with the names of the material files. |
| `in` | *density_max* | maximum density in the geometry (needed to set Woodcock trick) |
| `out` | *density_-nominal* | Array with the nominal density of the materials read |
| `out` | *mfp_table_-data* | Constant values for the linear interpolation |
| `out` | *mfp_table_-a_ptr* | First element for the linear interpolation. |
| `out` | *mfp_table_-b_ptr* | Second element for the linear interpolation. |

Definition at line 2110 of file MC-GPU_v1.3.cu.

References rayleigh_struct::aco, rayleigh_struct::bco, linear_interp::e0, compton_-struct::fco, compton_struct::fj0, linear_interp::ide, rayleigh_struct::itlco, rayleigh_struct-::ituco, MASTER_THREAD, MAX_ENERGYBINS_RAYLEIGH, MAX_MATERIALS, MAX_SHELLS, compton_struct::noscco, NP_RAYLEIGH, linear_interp::num_values, rayleigh_struct::pco, rayleigh_struct::pmax, compton_struct::uico, and rayleigh_struct-::xco.

Referenced by main().

#### 5.2.2.5 void **load_voxels** ( int *myID,* char ∗ *file_name_voxels,* float ∗ *density_max,* struct **voxel_struct** ∗ *voxel_data,* **float2** ∗∗ *voxel_mat_dens_ptr,* unsigned int ∗ *voxel_mat_dens_bytes,* short int ∗ *dose_ROI_x_max,* short int ∗ *dose_ROI_y_max,* short int ∗ *dose_ROI_z_max* )

Read the voxel data and allocate the material and density matrix.

Also find and report the maximum density defined in the geometry.

**Parameters**

| | | |
|---|---|---|
| in | *file_name_-voxels* | Name of the voxelized geometry file. |
| out | *density_max* | Array with the maximum density for each material in the voxels. |
| out | *voxel_data* | Pointer to a structure containing the voxel number and size. |
| out | *voxel_mat_-dens_ptr* | Pointer to the vector with the voxel materials and densities. |
| in | *dose_ROI_-x/y/z_max* | Size of the dose ROI: can not be larger than the total number of voxels in the geometry. |

Definition at line 1929 of file MC-GPU_v1.3.cu.

References voxel_struct::inv_voxel_size, MASTER_THREAD, MAX_MATERIALS, min-_value, voxel_struct::num_voxels, voxel_struct::size_bbox, int3::x, float3::x, int3::y, float3::y, int3::z, and float3::z.

Referenced by main().

### 5.2.2.6 int main ( int *argc,* char ∗∗ *argv* )

Main program of MC-GPU: initialize the simulation enviroment, launch the GPU kernels that perform the x ray transport and report the final results.

This function reads the description of the simulation from an external file given in the command line. This input file defines the number of particles to simulate, the characteristics of the x-ray source and the detector, the number and spacing of the projections (if simulating a CT), the location of the material files containing the interaction mean free paths, and the location of the voxelized geometry file.

**Author**

Andreu Badal

Definition at line 377 of file MC-GPU_v1.3.cu.

References source_struct::direction, dose_ROI_x_max_CONST, dose_ROI_x_min_C-ONST, dose_ROI_y_max_CONST, dose_ROI_y_min_CONST, dose_ROI_z_max_CO-NST, dose_ROI_z_min_CONST, linear_interp::e0, source_energy_struct::espc, linear-_interp::ide, init_energy_spectrum(), voxel_struct::inv_voxel_size, load_material(), load-_voxels(), MASTER_THREAD, MAX_MATERIALS, mfp_table_data_CONST, source-_energy_struct::num_bins_espc, detector_struct::num_pixels, linear_interp::num_-values, voxel_struct::num_voxels, source_struct::position, RAD2DEG, read_input(), report_image(), report_materials_dose(), report_voxels_dose(), detector_struct::sdd, set_CT_trajectory(), source_energy_data_CONST, track_particles(), update_seed_-PRNG(), voxel_data_CONST, int2::x, int3::x, float3::x, ulonglong2::x, int2::y, int3::y, float3::y, ulonglong2::y, int3::z, and float3::z.

#### 5.2.2.7 void **read_input** ( int *argc,* char ∗∗ *argv,* int *myID,* unsigned long long int ∗ *total_histories,* int ∗ *seed_input,* int ∗ *gpu_id,* int ∗ *num_threads_per_block,* int ∗ *histories_per_thread,* struct **detector_struct** ∗ *detector_data,* unsigned long long int ∗∗ *image_ptr,* int ∗ *image_bytes,* struct **source_struct** ∗ *source_data,* struct **source_energy_struct** ∗ *source_energy_data,* char ∗ *file_name_voxels,* char *file_name_materials[MAX_MATERIALS][250],* char ∗ *file_name_output,* char ∗ *file_name_espc,* int ∗ *num_projections,* double ∗ *D_angle,* double ∗ *angularROI_0,* double ∗ *angularROI_1,* double ∗ *initial_angle,* **ulonglong2** ∗∗ *voxels_Edep_ptr,* int ∗ *voxels_Edep_bytes,* char ∗ *file_dose_output,* short int ∗ *dose_ROI_x_min,* short int ∗ *dose_ROI_x_max,* short int ∗ *dose_ROI_y_min,* short int ∗ *dose_ROI_y_max,* short int ∗ *dose_ROI_z_min,* short int ∗ *dose_ROI_z_max,* double ∗ *SRotAxisD,* double ∗ *vertical_translation_per_projection,* int ∗ *flag_material_dose* )

Read the input file given in the command line and return the significant data.

Example input file:

1000000 [Total number of histories to simulate] geometry.vox [Voxelized geometry file name] material.mat [Material data file name]

**Parameters**

| | | |
|---|---|---|
| `in` | *argc* | Command line parameters |
| `in` | *argv* | Command line parameters: name of input file |
| `out` | *total_-histories* | Total number of particles to simulate |
| `out` | *seed_input* | Input random number generator seed |
| `out` | *num_-threads_per-_block* | Number of CUDA threads for each GPU block |
| `out` | *detector_-data* | |
| `out` | *image* | |
| `out` | *source_data* | |
| `out` | *file_name_-voxels* | |
| `out` | *file_name_-materials* | |
| `out` | *file_name_-output* | |

Definition at line 1242 of file MC-GPU_v1.3.cu.

References detector_struct::center, detector_struct::corner_min_rotated_to_Y, source-_struct::cos_theta_low, DEG2RAD, source_struct::direction, fgets_trimmed(), detector-_struct::height_Z, detector_struct::inv_pixel_size_X, detector_struct::inv_pixel_size_-Z, MASTER_THREAD, MAX_MATERIALS, MAX_NUM_PROJECTIONS, detector-

_struct::num_pixels, PI, source_struct::position, RAD2DEG, detector_struct::rot_inv, detector_struct::rotation_flag, detector_struct::sdd, detector_struct::total_num_pixels, trim_name(), detector_struct::width_X, int2::x, float3::x, int2::y, float3::y, and float3::z.

Referenced by main().

#### 5.2.2.8 int **report_image** ( char ∗ *file_name_output,* struct **detector_struct** ∗ *detector_data,* struct **source_struct** ∗ *source_data,* float *mean_energy_spectrum,* unsigned long long int ∗ *image,* double *time_elapsed,* unsigned long long int *total_histories,* int *current_projection,* int *num_projections,* double *D_angle,* double *initial_angle,* int *myID,* int *numprocs* )

Report the tallied image in ASCII and binary form (32-bit floats).

Separate images for primary and scatter radiation are generated.

**Parameters**

| | | |
|---|---|---|
| `in` | *file_name_-output* | File where tallied image is reported |
| `in` | *detector_-data* | Detector description read from the input file (pointer to **detector_struct** (p. 14)) |
| `in` | *image* | Tallied image (in meV per pixel) |
| `in` | *time_-elapsed* | Time elapsed during the main loop execution (in seconds) |
| `in` | *total_-histories* | Total number of x-rays simulated |

Definition at line 2715 of file MC-GPU_v1.3.cu.

References source_struct::direction, detector_struct::inv_pixel_size_X, detector_struct-::inv_pixel_size_Z, detector_struct::num_pixels, source_struct::position, RAD2DEG, S-CALE_eV, int2::x, float3::x, int2::y, float3::y, and float3::z.

Referenced by main().

#### 5.2.2.9 int **report_materials_dose** ( int *num_projections,* unsigned long long int *total_histories,* float ∗ *density_nominal,* **ulonglong2** ∗ *materials_dose,* double ∗ *mass_materials* )

Report the tallied dose to each material number, accounting for different densities in different regions with the same material number.

**Parameters**

| | | |
|---|---|---|
| `in` | *num_-projections* | Number of projections simulated |

| | | |
|---|---|---|
| `in` | *total_-histories* | Total number of x-rays simulated per projection |
| `out` | *density_-nominal* | Array with the nominal densities of materials given in the input file; -1 for materials not defined. Used to report only defined materials. |
| `in` | *materials_-dose* | Tallied dose and dose^2 arrays |

Definition at line 3128 of file MC-GPU_v1.3.cu.

References MAX_MATERIALS, and SCALE_eV.

Referenced by main().

#### 5.2.2.10 int **report_voxels_dose** ( char ∗ *file_dose_output,* int *num_projections,* struct **voxel_struct** ∗ *voxel_data,* **float2** ∗ *voxel_mat_dens,* **ulonglong2** ∗ *voxels_Edep,* double *time_total_MC_init_report,* unsigned long long int *total_histories,* short int *dose_ROI_x_min,* short int *dose_ROI_x_max,* short int *dose_ROI_y_min,* short int *dose_ROI_y_max,* short int *dose_ROI_z_min,* short int *dose_ROI_z_max,* struct **source_struct** ∗ *source_data* )

Report the total tallied 3D voxel dose deposition for all projections.

The voxel doses in the input ROI and their respective uncertainties are reported in binary form (32-bit floats) in two separate .raw files. The dose in a single plane at the level of the focal spot is also reported in ASCII format for simple visualization with GNUPLOT. The total dose deposited in each different material is reported to the standard output. The material dose is calculated adding the energy deposited in the individual voxels within the dose ROI, and dividing by the total mass of the material in the ROI.

**Parameters**

| | | |
|---|---|---|
| `in` | *file_dose_-output* | File where tallied image is reported |
| `in` | *detector_-data* | Detector description read from the input file (pointer to **detector_struct** (p. 14)) |
| `in` | *image* | Tallied image (in meV per pixel) |
| `in` | *time_-elapsed* | Time elapsed during the main loop execution (in seconds) |
| `in` | *total_-histories* | Total number of x-rays simulated |
| `in` | *source_data* | Data required to compute the voxel plane to report in ASCII format: Z at the level of the source, 1st projection |

Definition at line 2890 of file MC-GPU_v1.3.cu.

References voxel_struct::inv_voxel_size, MAX_MATERIALS, voxel_struct::num_voxels, SCALE_eV, int3::x, float3::x, ulonglong2::x, int3::y, float2::y, float3::y, and float3::z.

Referenced by main().

#### 5.2.2.11 int seeki_walker ( float ∗ *cutoff,* short int ∗ *alias,* float *randno,* int *n* ) [inline]

Finds the interval (x(i),x(i+1)] containing the input value using Walker's aliasing method.

Input: cutoff(1..n) -> interval cutoff values for the Walker method cutoff(1..n) -> alias for the upper part of each interval randno -> point to be located n -> no. of data points Output: index i of the semiopen interval where randno lies Comments: -> The cutoff and alias values have to be previously initialised calling the penelope subroutine IRND0.

Algorithm implementation based on the PENELOPE code developed by - Francesc Salvat at the University of Barcelona. For more info: www.oecd-nea.-org/science/pubs/2009/nea6416-penelope.pdf

Definition at line 3526 of file MC-GPU_v1.3.cu.

#### 5.2.2.12 void set_CT_trajectory ( int *myID,* int *num_projections,* double *D_angle,* double *angularROI_0,* double *angularROI_1,* double *SRotAxisD,* struct source_struct ∗ *source_data,* struct detector_struct ∗ *detector_data,* double *vertical_translation_per_projection* )

Sets the CT trajectory: store in memory the source and detector rotations that are needed to calculate the multiple projections.

The first projection (0) was previously initialized in function "read_input".

ASSUMPTIONS: the CT scan plane must be perpendicular to the Z axis, ie, the initial direction of the particles must have w=0!

Definition at line 3194 of file MC-GPU_v1.3.cu.

References detector_struct::center, detector_struct::corner_min_rotated_to_Y, source_struct::direction, detector_struct::height_Z, detector_struct::inv_pixel_size_X, detector_struct::inv_pixel_size_Z, MASTER_THREAD, MAX_NUM_PROJECTIONS, detector_struct::num_pixels, PI, source_struct::position, RAD2DEG, detector_struct-::rot_inv, detector_struct::rotation_flag, detector_struct::sdd, detector_struct::total_-num_pixels, detector_struct::width_X, float3::x, float3::y, and float3::z.

Referenced by main().

#### 5.2.2.13 void **trim_name** ( char ∗ *input_line,* char ∗ *file_name* )

Extract a file name from an input text line, trimming the initial blanks, trailing comment (#) and stopping at the first blank (the file name should not contain blanks).

**Parameters**

| | | |
|---|---|---|
| `in` | *input_line* | Input sentence with blanks and a trailing comment |
| `out` | *file_name* | Trimmed file name |

Definition at line 1840 of file MC-GPU_v1.3.cu.

Referenced by read_input().

#### 5.2.2.14 void **update_seed_PRNG** ( int *batch_number,* unsigned long long int *total_histories,* int ∗ *seed* ) `[inline]`

Initialize the first seed of the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).

This function is equivalent to "init_PRNG" but only updates one of the seeds.

Note that if we use the same seed number to initialize the 2 MLCGs of the PRNG we can only warranty that the first MLCG will be uncorrelated for each value generated by "update_seed_PRNG". There is a tiny chance that the final PRNs will be correlated because the leap frog on the first MLCG will probably go over the repetition cycle of the MLCG, which is much smaller than the full RANECU. But any correlataion is extremely unlikely. Function "init_PRNG" doesn't have this issue.

**Parameters**

| | | |
|---|---|---|
| `in` | *batch_-number* | Elements to skip (eg, MPI thread_number). |
| `in` | *total_-histories* | Histories to skip. |
| `in,out` | *seed* | Initial PRNG seeds; returns the updated seed. |

Definition at line 3356 of file MC-GPU_v1.3.cu.

References a1_RANECU, abMODm(), LEAP_DISTANCE, and m1_RANECU.

Referenced by main().

## 5.3 MC-GPU_v1.3.h File Reference

Header file containing the declarations for the MC-GPU code.

## Classes

- struct **int2**
- struct **int3**
- struct **float2**
- struct **float3**
- struct **double2**
- struct **double3**
- struct **short3**
- struct **ulonglong2**
- struct **source_struct**

  *Structure storing the data defining the source model (except for the energy spectrum).*
- struct **source_energy_struct**

  *Structure storing the source energy spectrum data to be sampled using the Walker aliasing algorithm.*
- struct **detector_struct**

  *Structure storing the data defining the x-ray detector.*
- struct **voxel_struct**

  *Structure defining a voxelized box with the back-lower corner at the coordinate origin.*
- struct **linear_interp**

  *Structure with the basic data required by the linear interpolation of the mean free paths: number of values and energy grid.*
- struct **compton_struct**

  *Structure storing the data of the Compton interaction sampling model (equivalent to PENELOPE's common block /CGCO/).*
- struct **rayleigh_struct**

  *Structure storing the data of the Rayleigh interaction sampling model (equivalent to PENELOPE's common block /CGRA/).*

## Defines

- #define **MASTER_THREAD** if(0==myID)

  *MPI macro: mark commands to be executed only by the master thread (myID==0).*
- #define **MAX_NUM_PROJECTIONS** 720

  *Maximum number of projections allowed in the CT simulation (not limited by the constant memory because stored in global and shared memory):*
- #define **MAX_MATERIALS** 15

  *Constants values for the Compton and Rayleigh models:*
- #define **MAX_SHELLS** 30
- #define **NP_RAYLEIGH** 128
- #define **MAX_ENERGYBINS_RAYLEIGH** 25005

- #define **MAX_ENERGY_BINS** 256

  *Maximum number of energy bins in the input x-ray energy spectrum.*
- #define **PI** 3.14159265358979323846
- #define **RAD2DEG** 180.0/**PI**
- #define **DEG2RAD** **PI**/180.0
- #define **SCALE_eV** 100.0f

  *Value to scale the deposited energy in the pixels so that it can be stored as a long long integer instead of a double precision float.*
- #define **EPS_SOURCE** 0.000015f

  *Offset value to make sure the particles are completely inside, or outside, the voxel bounding box.*
- #define **NEG_EPS_SOURCE** -**EPS_SOURCE**
- #define **INF** 500000.0f
- #define **INF_minus1** 499999.0f
- #define **NEG_INF** -500000.0f
- #define **max_value**(a, b) ( ((a) > (b)) ? (a) : (b) )

  *Preprocessor macro to calculate maximum and minimum values:*
- #define **min_value**(a, b) ( ((a) < (b)) ? (a) : (b) )

## Typedefs

- typedef struct **int2 int2**
- typedef struct **int3 int3**
- typedef struct **float2 float2**
- typedef struct **float3 float3**
- typedef struct **double2 double2**
- typedef struct **double3 double3**
- typedef struct **short3 short3**
- typedef struct **ulonglong2 ulonglong2**

## Functions

- void **read_input** (int argc, char ∗∗argv, int myID, unsigned long long int ∗total_histories, int ∗gpu_id, int ∗seed_input, int ∗num_threads_per_block, int ∗histories_per_thread, struct **detector_struct** ∗detector_data, unsigned long long int ∗∗image_ptr, int ∗image_bytes, struct **source_struct** ∗source_data, struct **source_energy_struct** ∗source_energy_data, char ∗file_name_voxels, char file_name_materials[**MAX_MATERIALS**][250], char ∗file_name_output, char ∗file_name_espc, int ∗num_projections, double ∗D_angle, double ∗angular-ROI_0, double ∗angularROI_1, double ∗initial_angle, **ulonglong2** ∗∗voxels_-Edep_ptr, int ∗voxels_Edep_bytes, char ∗file_dose_output, short int ∗dose_R-OI_x_min, short int ∗dose_ROI_x_max, short int ∗dose_ROI_y_min, short int

∗dose_ROI_y_max, short int ∗dose_ROI_z_min, short int ∗dose_ROI_z_max, double ∗SRotAxisD, double ∗vertical_translation_per_projection, int ∗flag_-material_dose)

*Read the input file given in the command line and return the significant data.*

- void **load_voxels** (int myID, char ∗file_name_voxels, float ∗density_max, struct **voxel_struct** ∗voxel_data, **float2** ∗∗voxel_mat_dens_ptr, unsigned int ∗voxel-_mat_dens_bytes, short int ∗dose_ROI_x_max, short int ∗dose_ROI_y_max, short int ∗dose_ROI_z_max)

  *Read the voxel data and allocate the material and density matrix.*

- void **load_material** (int myID, char file_name_materials[**MAX_MATERIAL-S**][250], float ∗density_max, float ∗density_nominal, struct **linear_interp** ∗mfp-_table_data, **float2** ∗∗mfp_Woodcock_table, int ∗mfp_Woodcock_table_bytes, **float3** ∗∗mfp_table_a_ptr, **float3** ∗∗mfp_table_b_ptr, int ∗mfp_table_bytes, struct **rayleigh_struct** ∗rayleigh_table_ptr, struct **compton_struct** ∗compton_table_-ptr)

  *Read the material input files and set the mean free paths and the "linear_interp" structures.*

- void **trim_name** (char ∗input_line, char ∗file_name)

  *Extract a file name from an input text line, trimming the initial blanks, trailing comment (#) and stopping at the first blank (the file name should not contain blanks).*

- char ∗ **fgets_trimmed** (char ∗trimmed_line, int num, FILE ∗file_ptr)

  *Read a line of text and trim initial blancks and trailing comments (#).*

- int **report_image** (char ∗file_name_output, struct **detector_struct** ∗detector-_data, struct **source_struct** ∗source_data, float mean_energy_spectrum, unsigned long long int ∗image, double time_elapsed, unsigned long long int total-_histories, int current_projection, int num_projections, double D_angle, double initial_angle, int myID, int numprocs)

  *Report the tallied image in ASCII and binary form (32-bit floats).*

- void **set_CT_trajectory** (int myID, int num_projections, double D_angle, double angularROI_0, double angularROI_1, double SRotAxisD, struct **source_-struct** ∗source_data, struct **detector_struct** ∗detector_data, double vertical_-translation_per_projection)

  *Sets the CT trajectory: store in memory the source and detector rotations that are needed to calculate the multiple projections.*

- int **report_voxels_dose** (char ∗file_dose_output, int num_projections, struct **voxel_struct** ∗voxel_data, **float2** ∗voxel_mat_dens, **ulonglong2** ∗voxels_Edep, double time_elapsed_total, unsigned long long int total_histories, short int dose-_ROI_x_min, short int dose_ROI_x_max, short int dose_ROI_y_min, short int dose_ROI_y_max, short int dose_ROI_z_min, short int dose_ROI_z_max, struct **source_struct** ∗source_data)

  *Report the total tallied 3D voxel dose deposition for all projections.*

- void **init_energy_spectrum** (char ∗file_name_espc, struct **source_energy_-struct** ∗source_energy_data, float ∗mean_energy_spectrum)

*Read the energy spectrum file and initialize the Walker aliasing sampling.*

- void **update_seed_PRNG** (int batch_number, unsigned long long int total_-histories, int ∗seed)

  *Initialize the first seed of the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).*

- void **IRND0** (float ∗W, float ∗F, short int ∗K, int N)

  *Initialisation of Walker's aliasing algorithm for random sampling from discrete probability distributions.*

- int **report_materials_dose** (int num_projections, unsigned long long int total_-histories, float ∗density_nominal, **ulonglong2** ∗materials_dose, double ∗mass_-materials)

  *Report the tallied dose to each material number, accounting for different densities in different regions with the same material number.*

- int **seeki_walker** (float ∗cutoff, short int ∗alias, float randno, int n)

  *Finds the interval (x(i),x(i+1)] containing the input value using Walker's aliasing method.*

- void **track_particles** (int history_batch, int histories_per_thread, int num_p, int seed_input, unsigned long long int ∗image, **ulonglong2** ∗voxels_Edep, **float2** ∗voxel_mat_dens, **float2** ∗mfp_Woodcock_table, **float3** ∗mfp_table_a, **float3** ∗mfp_table_b, struct **rayleigh_struct** ∗rayleigh_table, struct **compton_struct** ∗compton_table, struct **detector_struct** ∗detector_data_array, struct **source_-struct** ∗source_data_array, **ulonglong2** ∗materials_dose)

  *Initialize the image array, ie, set all pixels to zero Essentially, this function has the same effect as the command: "cutilSafeCall(cudaMemcpy(image_device, image, image_-bytes, cudaMemcpyHostToDevice))";.*

- void **source** (**float3** ∗position, **float3** ∗direction, float ∗energy, **int2** ∗seed, int ∗absvox, struct **source_struct** ∗source_data_SHARED, struct **detector_struct** ∗detector_data_SHARED)

  *Source that creates primary x rays, according to the defined source model.*

- void **move_to_bbox** (**float3** ∗position, **float3** ∗direction, **float3** size_bbox, int ∗intersection_flag)

  *Functions that moves a particle inside the voxelized geometry bounding box.*

- void **tally_image** (float ∗energy, **float3** ∗position, **float3** ∗direction, signed char ∗scatter_state, unsigned long long int ∗image, struct **source_struct** ∗source_-data_SHARED, struct **detector_struct** ∗detector_data_SHARED)

  *Tally a radiographic projection image.*

- void **init_PRNG** (int history_batch, int histories_per_thread, int seed_input, **int2** ∗seed)

  *Initialize the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).*

- int **abMODm** (int m, int a, int s)

  *Calculate "(a1∗a2) MOD m" with 32-bit integers and avoiding the possible overflow, using the Russian Peasant approach modulo m and the approximate factoring method, as described in: L'Ecuyer and Cote, ACM Trans.*

- float **ranecu** (**int2** ∗seed)

  *Pseudo-random number generator (PRNG) RANECU returning a float value (single precision version).*
- double **ranecu_double** (**int2** ∗seed)

  *Pseudo-random number generator (PRNG) RANECU returning a double value.*
- int **locate_voxel** (**float3** ∗position, **short3** ∗voxel_coord)

  *Find the voxel that contains the current position.*
- void **rotate_double** (**float3** ∗direction, double cos_theta, double phi)

  *Rotates a vector; the rotation is specified by giving the polar and azimuthal angles in the "self-frame", as determined by the vector to be rotated.*
- void **GRAa** (float ∗energy, double ∗costh_Rayleigh, int ∗mat, float ∗pmax_-current, **int2** ∗seed, struct **rayleigh_struct** ∗cgra)

  *Sample a Rayleigh interaction using the sampling algorithm used in PENELOPE 2006.*
- void **GCOa** (float ∗energy, double ∗costh_Compton, int ∗mat, **int2** ∗seed, struct **compton_struct** ∗cgco_SHARED)

  *Random sampling of incoherent (Compton) scattering of photons, using the sampling algorithm from PENELOPE 2006: Relativistic impulse approximation with analytical one-electron Compton profiles.*
- void **tally_voxel_energy_deposition** (float ∗Edep, **short3** ∗voxel_coord, **ulong-long2** ∗dose)

  *Tally the dose deposited in the voxels.*
- void **tally_materials_dose** (float ∗Edep, int ∗material, **ulonglong2** ∗materials_-dose)

  *Tally the depose deposited inside each material.*

## Variables

- short int **dose_ROI_x_min_CONST**

  *Global variable to be stored in the GPU constant memory defining the coordinates of the dose deposition region of interest.*
- short int **dose_ROI_x_max_CONST**
- short int **dose_ROI_y_min_CONST**
- short int **dose_ROI_y_max_CONST**
- short int **dose_ROI_z_min_CONST**
- short int **dose_ROI_z_max_CONST**
- struct **voxel_struct voxel_data_CONST**

  *Global variable to be stored in the GPU constant memory defining the size of the voxel phantom.*
- struct **source_struct source_data_CONST**

  *Global variable to be stored in the GPU constant memory defining the x-ray source.*
- struct **detector_struct detector_data_CONST**

*Global variable to be stored in the GPU constant memory defining the x-ray detector.*

- struct **linear_interp mfp_table_data_CONST**

  *Global variable to be stored in the GPU constant memory defining the linear interpolation data.*
- struct **source_energy_struct source_energy_data_CONST**

  *Global variable to be stored in the GPU constant memory defining the source energy spectrum.*

### 5.3.1 Detailed Description

Header file containing the declarations for the MC-GPU code. This file declares all the host and device functions and structures, the library files to include in the compilation, various constants parameters of the simulation, pre-processor macro functions, etc.

**Author**

Andreu Badal (`Andreu.Badal-Soler@fda.hhs.gov`)

**Date**

2012/12/12

Definition in file **MC-GPU_v1.3.h**.

### 5.3.2 Define Documentation

#### 5.3.2.1 #define DEG2RAD PI/180.0

Definition at line 73 of file MC-GPU_v1.3.h.

Referenced by read_input().

#### 5.3.2.2 #define EPS_SOURCE 0.000015f

Offset value to make sure the particles are completely inside, or outside, the voxel bounding box.

For example, to start a particle track the source may have to translate a particle from the focal spot to the plane Y=0, but in reality the particle will be moved to Y=EPS_SOURCE to make sure it is unmistakenly located inside a voxel. If "EPS_SOURCE" is too small an artifact with weird concentric circular shadows may appear at the center of the simulated image.

Definition at line 85 of file MC-GPU_v1.3.h.

Referenced by locate_voxel(), and move_to_bbox().

#### 5.3.2.3 #define INF 500000.0f

Definition at line 88 of file MC-GPU_v1.3.h.

#### 5.3.2.4 #define INF_minus1 499999.0f

Definition at line 89 of file MC-GPU_v1.3.h.

#### 5.3.2.5 #define MASTER_THREAD if(0==myID)

MPI macro: mark commands to be executed only by the master thread (myID==0).

Definition at line 56 of file MC-GPU_v1.3.h.

Referenced by load_material(), load_voxels(), main(), read_input(), and set_CT_-trajectory().

#### 5.3.2.6 #define MAX_ENERGY_BINS 256

Maximum number of energy bins in the input x-ray energy spectrum.

Definition at line 68 of file MC-GPU_v1.3.h.

Referenced by init_energy_spectrum().

#### 5.3.2.7 #define MAX_ENERGYBINS_RAYLEIGH 25005

Definition at line 65 of file MC-GPU_v1.3.h.

Referenced by load_material().

#### 5.3.2.8 #define MAX_MATERIALS 15

Constants values for the Compton and Rayleigh models:

Definition at line 62 of file MC-GPU_v1.3.h.

Referenced by GCOa(), load_material(), load_voxels(), main(), read_input(), report_-materials_dose(), report_voxels_dose(), and track_particles().

#### 5.3.2.9 #define MAX_NUM_PROJECTIONS 720

Maximum number of projections allowed in the CT simulation (not limited by the constant memory because stored in global and shared memory):

Definition at line 59 of file MC-GPU_v1.3.h.

Referenced by read_input(), and set_CT_trajectory().

#### 5.3.2.10 #define MAX_SHELLS 30

Definition at line 63 of file MC-GPU_v1.3.h.

Referenced by GCOa(), and load_material().

#### 5.3.2.11 #define max_value( *a, b* ) ( ((a) > (b)) ? (a) : (b) )

Preprocessor macro to calculate maximum and minimum values:

Definition at line 93 of file MC-GPU_v1.3.h.

Referenced by GCOa(), and init_energy_spectrum().

#### 5.3.2.12 #define min_value( *a, b* ) ( ((a) < (b)) ? (a) : (b) )

Definition at line 94 of file MC-GPU_v1.3.h.

Referenced by GCOa(), GRAa(), and load_voxels().

#### 5.3.2.13 #define NEG_EPS_SOURCE -EPS_SOURCE

Definition at line 87 of file MC-GPU_v1.3.h.

Referenced by move_to_bbox().

#### 5.3.2.14 #define NEG_INF -500000.0f

Definition at line 90 of file MC-GPU_v1.3.h.

Referenced by move_to_bbox().

#### 5.3.2.15 #define NP_RAYLEIGH 128

Definition at line 64 of file MC-GPU_v1.3.h.

Referenced by GRAa(), and load_material().

#### 5.3.2.16 #define PI 3.14159265358979323846

Definition at line 71 of file MC-GPU_v1.3.h.

Referenced by read_input(), and set_CT_trajectory().

#### 5.3.2.17 #define RAD2DEG 180.0/PI

Definition at line 72 of file MC-GPU_v1.3.h.

Referenced by main(), read_input(), report_image(), and set_CT_trajectory().

#### 5.3.2.18 #define SCALE_eV 100.0f

Value to scale the deposited energy in the pixels so that it can be stored as a long long integer instead of a double precision float.

The integer values have to be used in order to use the atomicAdd function in CUDA.

Definition at line 79 of file MC-GPU_v1.3.h.

Referenced by report_image(), report_materials_dose(), report_voxels_dose(), tally_-image(), tally_materials_dose(), and tally_voxel_energy_deposition().

### 5.3.3 Typedef Documentation

#### 5.3.3.1 typedef struct double2 double2

Definition at line 134 of file MC-GPU_v1.3.h.

#### 5.3.3.2 typedef struct double3 double3

Definition at line 135 of file MC-GPU_v1.3.h.

#### 5.3.3.3 typedef struct float2 float2

Definition at line 132 of file MC-GPU_v1.3.h.

#### 5.3.3.4 typedef struct float3 float3

Definition at line 133 of file MC-GPU_v1.3.h.

#### 5.3.3.5 typedef struct **int2 int2**

Definition at line 130 of file MC-GPU_v1.3.h.

#### 5.3.3.6 typedef struct **int3 int3**

Definition at line 131 of file MC-GPU_v1.3.h.

#### 5.3.3.7 typedef struct **short3 short3**

Definition at line 136 of file MC-GPU_v1.3.h.

#### 5.3.3.8 typedef struct **ulonglong2 ulonglong2**

Definition at line 137 of file MC-GPU_v1.3.h.

### 5.3.4 Function Documentation

#### 5.3.4.1 int **abMODm** ( int *m*, int *a*, int *s* ) `[inline]`

Calculate "(a1∗a2) MOD m" with 32-bit integers and avoiding the possible overflow, using the Russian Peasant approach modulo m and the approximate factoring method, as described in: L'Ecuyer and Cote, ACM Trans.

Math. Soft. 17 (1991).

This function has been adapted from "seedsMLCG.f", see: Badal and Sempau, - Computer Physics Communications 175 (2006)

**Parameters**

| | | |
|---|---|---|
| `in` | *m,a,s* | MLCG parameters |

**Returns**

(a1∗a2) MOD m

Definition at line 919 of file MC-GPU_kernel_v1.3.cu.

Referenced by init_PRNG(), and update_seed_PRNG().

#### 5.3.4.2 char∗ **fgets_trimmed** ( char ∗ *trimmed_line*, int *num*, FILE ∗ *file_ptr* )

Read a line of text and trim initial blancks and trailing comments (#).

**Parameters**

| | | |
|---|---|---|
| `in` | *num* | Characters to read |
| `in` | *file_ptr* | Pointer to the input file stream |
| `out` | *trimmed_line* | Trimmed line from input file, skipping empty lines and comments |

Definition at line 1868 of file MC-GPU_v1.3.cu.

Referenced by init_energy_spectrum(), and read_input().

#### 5.3.4.3 void **GCOa** ( float ∗ *energy,* double ∗ *costh_Compton,* int ∗ *mat,* **int2** ∗ *seed,* struct **compton_struct** ∗ *cgco_SHARED* ) `[inline]`

Random sampling of incoherent (Compton) scattering of photons, using the sampling algorithm from PENELOPE 2006: Relativistic impulse approximation with analytical one-electron Compton profiles.

**Parameters**

| | | |
|---|---|---|
| `in,out` | *energy* | incident and final photon energy (eV) |
| `out` | *costh_-Compton* | cosine of the polar scattering angle |
| `in` | *material* | Current voxel material |
| `in` | *seed* | RANECU PRNG seed |

Definition at line 1287 of file MC-GPU_kernel_v1.3.cu.

References compton_struct::fco, compton_struct::fj0, MAX_MATERIALS, MAX_SHELLS, max_value, min_value, compton_struct::noscco, ranecu(), and compton_struct::uico.

Referenced by track_particles().

#### 5.3.4.4 void **GRAa** ( float ∗ *energy,* double ∗ *costh_Rayleigh,* int ∗ *mat,* float ∗ *pmax_current,* **int2** ∗ *seed,* struct **rayleigh_struct** ∗ *cgra* ) `[inline]`

Sample a Rayleigh interaction using the sampling algorithm used in PENELOPE 2006.

**Parameters**

| | | |
|---|---|---|
| `in` | *energy* | Particle energy (not modified with Rayleigh) |
| `out` | *costh_-Rayleigh* | Cosine of the angular deflection |
| `in` | *material* | Current voxel material |

Definition at line 1181 of file MC-GPU_kernel_v1.3.cu.

References rayleigh_struct::aco, rayleigh_struct::bco, rayleigh_struct::itlco, rayleigh_-struct::ituco, min_value, NP_RAYLEIGH, rayleigh_struct::pco, ranecu_double(), and rayleigh_struct::xco.

Referenced by track_particles().

#### 5.3.4.5 void init_energy_spectrum ( char ∗ *file_name_espc,* struct source_energy_struct ∗ *source_energy_data,* float ∗ *mean_energy_spectrum* )

Read the energy spectrum file and initialize the Walker aliasing sampling.

**Parameters**

| | | |
|---|---|---|
| in | *file_name_-espc* | File containing the energy spectrum (lower energy value in each bin and its emission probability). |
| in,out | *source_-energy_data* | Energy spectrum and other source data. The Walker alias and cutoffs are initialized in this function. |
| out | *mean_-energy_-spectrum* | Mean energy in the input x-ray energy spectrum. |

!Walker!! Calling PENELOPE's function to init the Walker method

Definition at line 3398 of file MC-GPU_v1.3.cu.

References source_energy_struct::espc, source_energy_struct::espc_alias, source_-energy_struct::espc_cutoff, fgets_trimmed(), IRND0(), MAX_ENERGY_BINS, max_-value, and source_energy_struct::num_bins_espc.

Referenced by main().

#### 5.3.4.6 void init_PRNG ( int *history_batch,* int *histories_per_thread,* int *seed_input,* int2 ∗ *seed* ) [inline]

Initialize the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).

Each calculated seed initiates a consecutive and disjoint sequence of pseudo-random numbers with length LEAP_DISTANCE, that can be used to in a parallel simulation (-Sequence Splitting parallelization method). The basic equation behind the algorithm is: S(i+j) = (a∗∗j ∗ S(i)) MOD m = [(a∗∗j MOD m)∗S(i)] MOD m , which is described in: P L'Ecuyer, Commun. ACM 31 (1988) p.742

This function has been adapted from "seedsMLCG.f", see: A Badal and J Sempau, Computer Physics Communications 175 (2006) p. 440-450

**Parameters**

| | | |
|---|---|---|
| in | *history* | Particle bach number. |
| in | *seed_input* | Initial PRNG seed input (used to initiate both MLCGs in RANECU). |
| out | *seed* | Initial PRNG seeds for the present history. |

Definition at line 841 of file MC-GPU_kernel_v1.3.cu.

References a1_RANECU, a2_RANECU, abMODm(), LEAP_DISTANCE, m1_RANECU, m2_RANECU, int2::x, and int2::y.

Referenced by track_particles().

#### 5.3.4.7 void **IRND0** ( float ∗ *W,* float ∗ *F,* short int ∗ *K,* int *N* )

Initialisation of Walker's aliasing algorithm for random sampling from discrete probability distributions.

Input arguments: N ........ number of different values of the random variable. W(1:-N) ... corresponding point probabilities (not necessarily normalised to unity). Output arguments: F(1:N) ... cutoff values. K(1:N) ... alias values.

This subroutine is part of the PENELOPE 2006 code developed by Francesc Salvat at the University of Barcelona. For more info: www.oecd-nea.org/science/pubs/2009/nea6416-penelope.pdf

Definition at line 3575 of file MC-GPU_v1.3.cu.

Referenced by init_energy_spectrum().

#### 5.3.4.8 void **load_material** ( int *myID,* char *file_name_materials[MAX_MATERIALS][250],* float ∗ *density_max,* float ∗ *density_nominal,* struct **linear_interp** ∗ *mfp_table_data,* **float2** ∗∗ *mfp_Woodcock_table_ptr,* int ∗ *mfp_Woodcock_table_bytes,* **float3** ∗∗ *mfp_table_a_ptr,* **float3** ∗∗ *mfp_table_b_ptr,* int ∗ *mfp_table_bytes,* struct **rayleigh_struct** ∗ *rayleigh_table_ptr,* struct **compton_struct** ∗ *compton_table_ptr* )

Read the material input files and set the mean free paths and the "linear_interp" structures.

Find the material nominal density. Set the Woodcock trick data.

**Parameters**

| | | |
|---|---|---|
| in | *file_name_-materials* | Array with the names of the material files. |
| in | *density_max* | maximum density in the geometry (needed to set Woodcock trick) |

| | | |
|---|---|---|
| out | *density_-nominal* | Array with the nominal density of the materials read |
| out | *mfp_table_-data* | Constant values for the linear interpolation |
| out | *mfp_table_-a_ptr* | First element for the linear interpolation. |
| out | *mfp_table_-b_ptr* | Second element for the linear interpolation. |

Definition at line 2110 of file MC-GPU_v1.3.cu.

References rayleigh_struct::aco, rayleigh_struct::bco, linear_interp::e0, compton_-struct::fco, compton_struct::fj0, linear_interp::ide, rayleigh_struct::itlco, rayleigh_struct-::ituco, MASTER_THREAD, MAX_ENERGYBINS_RAYLEIGH, MAX_MATERIALS, MAX_SHELLS, compton_struct::noscco, NP_RAYLEIGH, linear_interp::num_values, rayleigh_struct::pco, rayleigh_struct::pmax, compton_struct::uico, and rayleigh_struct-::xco.

Referenced by main().

#### 5.3.4.9 void **load_voxels** ( int *myID,* char ∗ *file_name_voxels,* float ∗ *density_max,* struct **voxel_struct** ∗ *voxel_data,* **float2** ∗∗ *voxel_mat_dens_ptr,* unsigned int ∗ *voxel_mat_dens_bytes,* short int ∗ *dose_ROI_x_max,* short int ∗ *dose_ROI_y_max,* short int ∗ *dose_ROI_z_max* )

Read the voxel data and allocate the material and density matrix.

Also find and report the maximum density defined in the geometry.

**Parameters**

| | | |
|---|---|---|
| in | *file_name_-voxels* | Name of the voxelized geometry file. |
| out | *density_max* | Array with the maximum density for each material in the vox-els. |
| out | *voxel_data* | Pointer to a structure containing the voxel number and size. |
| out | *voxel_mat_-dens_ptr* | Pointer to the vector with the voxel materials and densities. |
| in | *dose_ROI_-x/y/z_max* | Size of the dose ROI: can not be larger than the total number of voxels in the geometry. |

Definition at line 1929 of file MC-GPU_v1.3.cu.

References voxel_struct::inv_voxel_size, MASTER_THREAD, MAX_MATERIALS, min-_value, voxel_struct::num_voxels, voxel_struct::size_bbox, int3::x, float3::x, int3::y, float3::y, int3::z, and float3::z.

Referenced by main().

#### 5.3.4.10 int **locate_voxel** ( **float3** ∗ *position,* **short3** ∗ *voxel_coord* ) `[inline]`

Find the voxel that contains the current position.

Report the voxel absolute index and the x,y,z indices. The structure containing the voxel number and size is read from CONSTANT memory.

**Parameters**

| | | |
|---|---|---|
| `in` | *position* | Particle position |
| `out` | *voxel_coord* | Pointer to three integer values (short3∗) that will store the x,y and z voxel indices. |

**Returns**

Returns "absvox", the voxel number where the particle is located (negative if position outside the voxel bbox).

Definition at line 1033 of file MC-GPU_kernel_v1.3.cu.

References EPS_SOURCE, voxel_struct::inv_voxel_size, voxel_struct::num_voxels, voxel_struct::size_bbox, voxel_data_CONST, int3::x, float3::x, short3::x, int3::y, float3::y, short3::y, float3::z, and short3::z.

Referenced by track_particles().

#### 5.3.4.11 void **move_to_bbox** ( **float3** ∗ *position,* **float3** ∗ *direction,* **float3** *size_bbox,* **int** ∗ *intersection_flag* ) `[inline]`

Functions that moves a particle inside the voxelized geometry bounding box.

An EPSILON distance is added to make sure the particles will be clearly inside the bbox, not exactly on the surface.

This algorithm makes the following assumtions:

- The back lower vertex of the voxel bounding box is always located at the origin: (x0,y0,z0)=(0,0,0).
- The initial value of "position" corresponds to the focal spot location.
- When a ray is not pointing towards the bbox plane that it should cross according to the sign of the direction, I assign a distance to the intersection =0 instead of the real negative distance. The wall that will be crossed to enter the bbox is always the furthest and therefore a 0 distance will never be used except in the case of a

ray starting inside the bbox or outside the bbox and not pointing to any of the 3 planes. In this situation the ray will be transported a 0 distance, meaning that it will stay at the focal spot.

(Interesting information on ray-box intersection: `http://tog.acm.org/resources/-GraphicsGems/gems/RayBox.c`)

**Parameters**

| | | |
|---|---|---|
| `in,out` | *position* | Particle position: initially set to the focal spot, returned transported inside the voxel bbox. |
| `out` | *direction* | Sampled particle direction (cosine vectors). |
| `out` | *intersection-_flag* | Set to <0 if particle outside bbox and will not cross the voxels, not changed otherwise. |
| `out` | *size_bbox* | Size of the bounding box. |

Definition at line 714 of file MC-GPU_kernel_v1.3.cu.

References EPS_SOURCE, NEG_EPS_SOURCE, NEG_INF, float3::x, float3::y, and float3::z.

Referenced by source().

#### 5.3.4.12 float ranecu ( int2 ∗ *seed* ) `[inline]`

Pseudo-random number generator (PRNG) RANECU returning a float value (single precision version).

**Parameters**

| | | |
|---|---|---|
| `in,out` | *seed* | PRNG seed (seed kept in the calling function and updated here). |

**Returns**

PRN double value in the open interval (0,1)

Definition at line 965 of file MC-GPU_kernel_v1.3.cu.

References int2::x, and int2::y.

Referenced by GCOa(), source(), and track_particles().

#### 5.3.4.13 double ranecu_double ( int2 ∗ *seed* ) `[inline]`

Pseudo-random number generator (PRNG) RANECU returning a double value.

Definition at line 995 of file MC-GPU_kernel_v1.3.cu.

References int2::x, and int2::y.

Referenced by GRAa(), and track_particles().

#### 5.3.4.14 void **read_input** ( int *argc*, char ∗∗ *argv*, int *myID*, unsigned long long int ∗ *total_histories*, int ∗ *seed_input*, int ∗ *gpu_id*, int ∗ *num_threads_per_block*, int ∗ *histories_per_thread*, struct **detector_struct** ∗ *detector_data*, unsigned long long int ∗∗ *image_ptr*, int ∗ *image_bytes*, struct **source_struct** ∗ *source_data*, struct **source_energy_struct** ∗ *source_energy_data*, char ∗ *file_name_voxels*, char *file_name_materials[MAX_MATERIALS][250]*, char ∗ *file_name_output*, char ∗ *file_name_espc*, int ∗ *num_projections*, double ∗ *D_angle*, double ∗ *angularROI_0*, double ∗ *angularROI_1*, double ∗ *initial_angle*, **ulonglong2** ∗∗ *voxels_Edep_ptr*, int ∗ *voxels_Edep_bytes*, char ∗ *file_dose_output*, short int ∗ *dose_ROI_x_min*, short int ∗ *dose_ROI_x_max*, short int ∗ *dose_ROI_y_min*, short int ∗ *dose_ROI_y_max*, short int ∗ *dose_ROI_z_min*, short int ∗ *dose_ROI_z_max*, double ∗ *SRotAxisD*, double ∗ *vertical_translation_per_projection*, int ∗ *flag_material_dose* )

Read the input file given in the command line and return the significant data.

Example input file:

1000000 [Total number of histories to simulate] geometry.vox [Voxelized geometry file name] material.mat [Material data file name]

**Parameters**

| | | |
|---|---|---|
| in | *argc* | Command line parameters |
| in | *argv* | Command line parameters: name of input file |
| out | *total_-histories* | Total number of particles to simulate |
| out | *seed_input* | Input random number generator seed |
| out | *num_-threads_per-_block* | Number of CUDA threads for each GPU block |
| out | *detector_-data* | |
| out | *image* | |
| out | *source_data* | |
| out | *file_name_-voxels* | |
| out | *file_name_-materials* | |
| out | *file_name_-output* | |

Definition at line 1242 of file MC-GPU_v1.3.cu.

References detector_struct::center, detector_struct::corner_min_rotated_to_Y, source-_struct::cos_theta_low, DEG2RAD, source_struct::direction, fgets_trimmed(), detector-_struct::height_Z, detector_struct::inv_pixel_size_X, detector_struct::inv_pixel_size_-Z, MASTER_THREAD, MAX_MATERIALS, MAX_NUM_PROJECTIONS, detector-_struct::num_pixels, PI, source_struct::position, RAD2DEG, detector_struct::rot_inv, detector_struct::rotation_flag, detector_struct::sdd, detector_struct::total_num_pixels, trim_name(), detector_struct::width_X, int2::x, float3::x, int2::y, float3::y, and float3::z.

Referenced by main().

#### 5.3.4.15 int **report_image** ( char ∗ *file_name_output,* struct **detector_struct** ∗ *detector_data,* struct **source_struct** ∗ *source_data,* float *mean_energy_spectrum,* unsigned long long int ∗ *image,* double *time_elapsed,* unsigned long long int *total_histories,* int *current_projection,* int *num_projections,* double *D_angle,* double *initial_angle,* int *myID,* int *numprocs* )

Report the tallied image in ASCII and binary form (32-bit floats).

Separate images for primary and scatter radiation are generated.

**Parameters**

| | | |
|---|---|---|
| in | *file_name_-output* | File where tallied image is reported |
| in | *detector_-data* | Detector description read from the input file (pointer to **detector_struct** (p. 14)) |
| in | *image* | Tallied image (in meV per pixel) |
| in | *time_-elapsed* | Time elapsed during the main loop execution (in seconds) |
| in | *total_-histories* | Total number of x-rays simulated |

Definition at line 2715 of file MC-GPU_v1.3.cu.

References source_struct::direction, detector_struct::inv_pixel_size_X, detector_struct-::inv_pixel_size_Z, detector_struct::num_pixels, source_struct::position, RAD2DEG, S-CALE_eV, int2::x, float3::x, int2::y, float3::y, and float3::z.

Referenced by main().

#### 5.3.4.16 int **report_materials_dose** ( int *num_projections*, unsigned long long int *total_histories*, float ∗ *density_nominal*, **ulonglong2** ∗ *materials_dose*, double ∗ *mass_materials* )

Report the tallied dose to each material number, accounting for different densities in different regions with the same material number.

**Parameters**

| | | |
|---|---|---|
| `in` | *num_-projections* | Number of projections simulated |
| `in` | *total_-histories* | Total number of x-rays simulated per projection |
| `out` | *density_-nominal* | Array with the nominal densities of materials given in the input file; -1 for materials not defined. Used to report only defined materials. |
| `in` | *materials_-dose* | Tallied dose and dose^2 arrays |

Definition at line 3128 of file MC-GPU_v1.3.cu.

References MAX_MATERIALS, and SCALE_eV.

Referenced by main().

#### 5.3.4.17 int **report_voxels_dose** ( char ∗ *file_dose_output*, int *num_projections*, struct **voxel_struct** ∗ *voxel_data*, **float2** ∗ *voxel_mat_dens*, **ulonglong2** ∗ *voxels_Edep*, double *time_total_MC_init_report*, unsigned long long int *total_histories*, short int *dose_ROI_x_min*, short int *dose_ROI_x_max*, short int *dose_ROI_y_min*, short int *dose_ROI_y_max*, short int *dose_ROI_z_min*, short int *dose_ROI_z_max*, struct **source_struct** ∗ *source_data* )

Report the total tallied 3D voxel dose deposition for all projections.

The voxel doses in the input ROI and their respective uncertainties are reported in binary form (32-bit floats) in two separate .raw files. The dose in a single plane at the level of the focal spot is also reported in ASCII format for simple visualization with GNUPLOT. The total dose deposited in each different material is reported to the standard output. The material dose is calculated adding the energy deposited in the individual voxels within the dose ROI, and dividing by the total mass of the material in the ROI.

**Parameters**

| | | |
|---|---|---|
| `in` | *file_dose_-output* | File where tallied image is reported |
| `in` | *detector_-data* | Detector description read from the input file (pointer to **detector_struct** (p. 14)) |

| | | |
|---|---|---|
| in | *image* | Tallied image (in meV per pixel) |
| in | *time_-elapsed* | Time elapsed during the main loop execution (in seconds) |
| in | *total_-histories* | Total number of x-rays simulated |
| in | *source_data* | Data required to compute the voxel plane to report in ASCII format: Z at the level of the source, 1st projection |

Definition at line 2890 of file MC-GPU_v1.3.cu.

References voxel_struct::inv_voxel_size, MAX_MATERIALS, voxel_struct::num_voxels, SCALE_eV, int3::x, float3::x, ulonglong2::x, int3::y, float2::y, float3::y, and float3::z.

Referenced by main().

#### 5.3.4.18 void **rotate_double** ( **float3** ∗ ***direction,*** double ***costh,*** double ***phi*** ) `[inline]`

Rotates a vector; the rotation is specified by giving the polar and azimuthal angles in the "self-frame", as determined by the vector to be rotated.

This function is a literal translation from Fortran to C of PENELOPE (v. 2006) subroutine "DIRECT".

**Parameters**

| | | |
|---|---|---|
| in,out | *(u,v,w)* | input vector (=d) in the lab. frame; returns the rotated vector components in the lab. frame |
| in | *costh* | cos(theta), angle between d before and after turn |
| in | *phi* | azimuthal angle (rad) turned by d in its self-frame |

Definition at line 1103 of file MC-GPU_kernel_v1.3.cu.

References float3::x, float3::y, and float3::z.

Referenced by track_particles().

#### 5.3.4.19 int **seeki_walker** ( float ∗ ***cutoff,*** short int ∗ ***alias,*** float ***randno,*** int ***n*** ) `[inline]`

Finds the interval (x(i),x(i+1)] containing the input value using Walker's aliasing method.

Input: cutoff(1..n) -> interval cutoff values for the Walker method cutoff(1..n) -> alias for the upper part of each interval randno -> point to be located n -> no. of data points Output: index i of the semiopen interval where randno lies Comments: -> The cutoff and alias values have to be previously initialised calling the penelope subroutine IRND0.

Algorithm implementation based on the PENELOPE code developed by -

Francesc Salvat at the University of Barcelona. For more info: www.oecd-nea.-org/science/pubs/2009/nea6416-penelope.pdf

Definition at line 3526 of file MC-GPU_v1.3.cu.

#### 5.3.4.20 void **set_CT_trajectory** ( int *myID,* int *num_projections,* double *D_angle,* double *angularROI_0,* double *angularROI_1,* double *SRotAxisD,* struct **source_struct** ∗ *source_data,* struct **detector_struct** ∗ *detector_data,* double *vertical_translation_per_projection* )

Sets the CT trajectory: store in memory the source and detector rotations that are needed to calculate the multiple projections.

The first projection (0) was previously initialized in function "read_input".

ASSUMPTIONS: the CT scan plane must be perpendicular to the Z axis, ie, the initial direction of the particles must have w=0!

Definition at line 3194 of file MC-GPU_v1.3.cu.

References detector_struct::center, detector_struct::corner_min_rotated_to_Y, source_struct::direction, detector_struct::height_Z, detector_struct::inv_pixel_size_X, detector_struct::inv_pixel_size_Z, MASTER_THREAD, MAX_NUM_PROJECTIONS, detector_struct::num_pixels, PI, source_struct::position, RAD2DEG, detector_struct-::rot_inv, detector_struct::rotation_flag, detector_struct::sdd, detector_struct::total_-num_pixels, detector_struct::width_X, float3::x, float3::y, and float3::z.

Referenced by main().

#### 5.3.4.21 void **source** ( **float3** ∗ *position,* **float3** ∗ *direction,* float ∗ *energy,* **int2** ∗ *seed,* int ∗ *absvox,* struct **source_struct** ∗ *source_data_SHARED,* struct **detector_struct** ∗ *detector_data_SHARED* ) `[inline]`

Source that creates primary x rays, according to the defined source model.

The particles are automatically moved to the surface of the voxel bounding box, to start the tracking inside a real material. If the sampled particle do not enter the voxels, it is init in the focal spot and the main program will check if it arrives at the detector or not.

**Parameters**

| | | |
|---|---|---|
| `in` | *source_data* | Structure describing the source. |
| `in` | *source_-energy_-data_CONS-T* | Global variable in constant memory space describing the source energy spectrum. |
| `out` | *position* | Initial particle position (particle transported inside the voxel bbox). |

| | | |
|---|---|---|
| out | *direction* | Sampled particle direction (cosine vectors). |
| out | *energy* | Sampled energy of the new x ray. |
| in | *seed* | Current seed of the random number generator, requiered to sample the movement direction. |
| out | *absvox* | Set to <0 if primary particle will not cross the voxels, not changed otherwise (>0). |

Definition at line 626 of file MC-GPU_kernel_v1.3.cu.

References source_struct::cos_theta_low, source_struct::D_cos_theta, source_struct::-D_phi, source_energy_struct::espc, source_energy_struct::espc_alias, source_energy-_struct::espc_cutoff, source_struct::max_height_at_y1cm, move_to_bbox(), source-_energy_struct::num_bins_espc, source_struct::phi_low, source_struct::position, ranecu(), source_struct::rot_fan, detector_struct::rotation_flag, voxel_struct::size_bbox, source_energy_data_CONST, voxel_data_CONST, float3::x, float3::y, and float3::z.

Referenced by track_particles().

#### 5.3.4.22 void **tally_image** ( float ∗ *energy,* **float3** ∗ *position,* **float3** ∗ *direction,* signed char ∗ *scatter_state,* unsigned long long int ∗ *image,* struct **source_struct** ∗ *source_data_SHARED,* struct **detector_struct** ∗ *detector_data_SHARED* ) [inline]

Tally a radiographic projection image.

This function is called whenever a particle escapes the voxelized volume. The code checks if the particle would arrive at the detector if it kept moving in a straight line after exiting the voxels (assuming vacuum enclosure). An ideal image formation model is implemented: each pixel counts the total energy of the x rays that enter the pixel (100% detection efficiency for any energy). The image due to primaries and different kinds of scatter is tallied separately.

In the GPU, and atomicAdd() function is used to make sure that multiple threads do not update the same pixel at the same time, which would result in a lose of information. Since the atomicAdd function is only available for 'unsigned long long int' data, the float pixel values are scaled by a factor "SCALE_eV" defined in the header file (eg, #define SCALE_eV 10000.0f) and stored as unsigned long long integers in main memory.

WARNING! If the total tallied signal (for all particles) is larger than "1.8e19/SCALE_eV", there will be a bit overflow and the value will be reset to 0 giving bogus results.

WARNING! The detector plane should be located outside the voxels bounding box. -However, since the particles are moved outside the bbox in the last step, they could cross the detector plane anyway. If the particles are less than 2.0 cm behind the detector, they are moved back and detected. Therefore the detector can be a few cm inside the bbox and still work. If the Woodcock mean free path is larger than the distance from the bbox to the detector, we may lose some particles behind the detector!

**Parameters**

| | | |
|---|---|---|
| in | *energy* | X-ray energy |
| in | *position* | Particle position |
| in | *direction* | Particle direction (cosine vectors) |
| in | *scatter_state* | Flag marking primaries, single Compton, single Rayleigh or multiple scattered radiation |
| out | *image* | Integer array containing the image, ie, the pixel values (in tenths of meV) |

Definition at line 482 of file MC-GPU_kernel_v1.3.cu.

References detector_struct::center, detector_struct::corner_min_rotated_to_Y, source-_struct::direction, detector_struct::inv_pixel_size_X, detector_struct::inv_pixel_size_Z, detector_struct::num_pixels, detector_struct::rot_inv, detector_struct::rotation_flag, SC-ALE_eV, detector_struct::total_num_pixels, int2::x, float3::x, int2::y, float3::y, and float3-::z.

Referenced by track_particles().

#### 5.3.4.23 void **tally_materials_dose** ( float ∗ *Edep,* int ∗ *material,* ulonglong2 ∗ *materials_dose* ) [inline]

Tally the depose deposited inside each material.

This function is called whenever a particle suffers a Compton or photoelectric interaction. The energy released in each interaction is added and later in the report function the total deposited energy is divided by the total mass of the material in the voxelized object to get the dose. This naturally accounts for multiple densities for voxels with the same material (not all voxels have same mass). Electrons are not transported in MC--GPU and therefore we are approximating that the dose is equal to the KERMA (energy released by the photons alone). This approximation is acceptable when there is electronic equilibrium and when the range of the secondary electrons is shorter than the organ size.

The function uses atomic functions for a thread-safe access to the GPU memory. We can check if this tally was disabled in the input file checking if the array materials_dose was allocated in the GPU (disabled if pointer = NULL).

**Parameters**

| | | |
|---|---|---|
| in | *Edep* | Energy deposited in the interaction |
| in | *material* | Current material id number |
| out | *materials_-dose* | **ulonglong2** (p. 28) array storing the mateials dose [in eV/g] and dose^2 (ie, uncertainty). |

Definition at line 1547 of file MC-GPU_kernel_v1.3.cu.

References SCALE_eV, ulonglong2::x, and ulonglong2::y.

Referenced by track_particles().

#### 5.3.4.24 void **tally_voxel_energy_deposition** ( float ∗ *Edep,* short3 ∗ *voxel_coord,* ulonglong2 ∗ *voxels_Edep* ) [inline]

Tally the dose deposited in the voxels.

This function is called whenever a particle suffers a Compton or photoelectric interaction. It is not necessary to call this function if the dose tally was disabled in the input file (ie, dose_ROI_x_max_CONST $< 0$). Electrons are not transported in MC-GPU and therefore we are approximating that the dose is equal to the KERMA (energy released by the photons alone). This approximation is acceptable when there is electronic equilibrium and when the range of the secondary electrons is shorter than the voxel size. Usually the doses will be acceptable for photon energies below 1 MeV. The dose estimates may not be accurate at the interface of low density volumes.

We need to use atomicAdd() in the GPU to prevent that multiple threads update the same voxel at the same time, which would result in a lose of information. This is very improbable when using a large number of voxels but gives troubles with a simple geometries with few voxels (in this case the atomicAdd will slow down the code because threads will update the voxel dose secuentially).

**Parameters**

| | | |
|---|---|---|
| in | *Edep* | Energy deposited in the interaction |
| in | *voxel_coord* | Voxel coordinates, needed to check if particle located inside the input region of interest (ROI) |
| out | *voxels_Edep* | **ulonglong2** (p. 28) array containing the 3D voxel dose and dose^2 (ie, uncertainty) as unsigned integers scaled by S-CALE_eV. |

Definition at line 418 of file MC-GPU_kernel_v1.3.cu.

References dose_ROI_x_max_CONST, dose_ROI_x_min_CONST, dose_ROI_y_max-_CONST, dose_ROI_y_min_CONST, dose_ROI_z_max_CONST, dose_ROI_z_min_-CONST, SCALE_eV, short3::x, ulonglong2::x, short3::y, ulonglong2::y, and short3::z.

Referenced by track_particles().

#### 5.3.4.25 void **track_particles** ( int *history_batch,* int *histories_per_thread,* int *num_p,* int *seed_input,* unsigned long long int ∗ *image,* **ulonglong2** ∗ *voxels_Edep,* **float2** ∗ *voxel_mat_dens,* **float2** ∗ *mfp_Woodcock_table,* **float3** ∗ *mfp_table_a,* **float3** ∗ *mfp_table_b,* struct **rayleigh_struct** ∗ *rayleigh_table,* struct **compton_struct** ∗ *compton_table,* struct **detector_struct** ∗ *detector_data_array,* struct **source_struct** ∗ *source_data_array,* **ulonglong2** ∗ *materials_dose* )

Initialize the image array, ie, set all pixels to zero Essentially, this function has the same effect as the command: "cutilSafeCall(cudaMemcpy(image_device, image, image_-bytes, cudaMemcpyHostToDevice))";.

CUDA performs some initialization work the first time a GPU kernel is called. Therefore, calling a short kernel before the real particle tracking is performed may improve the accuracy of the timing measurements in the relevant kernel.

**Parameters**

| | | |
|---|---|---|
| in,out | *image* | Pointer to the image array. |
| in | *pixels_per_-image* | Number of pixels in the image (ie, elements in the array). Main function to simulate x-ray tracks inside a voxelized geometry. Secondary electrons are not simulated (in photoelectric and Compton events the energy is locally deposited). |

The following global variables, in the GPU __constant__ memory are used: voxel_data-_CONST, source_energy_data_CONST, detector_data_CONST, mfp_table_data_CO-NST.

**Parameters**

| | | |
|---|---|---|
| in | *history_-batch* | Particle batch number (only used in the CPU version when CUDA is disabled!, the GPU uses the built-in variable threadIdx) |
| in | *num_p* | Projection number in the CT simulation. This variable defines a specific angle and the corresponding source and detector will be used. |
| in | *histories_-per_thread* | Number of histories to simulate for each call to this function (ie, for GPU thread). |
| in | *seed_input* | Random number generator seed (the same seed is used to initialize the two MLCGs of RANECU). |
| in | *voxel_mat_-dens* | Pointer to the voxel densities and material vector (the voxelized geometry), stored in GPU glbal memory. |
| in | *mfp_-Woodcock_-table* | Two parameter table for the linear interpolation of the -Woodcock mean free path (MFP) (stored in GPU global memory). |
| in | *mfp_table_a* | First element for the linear interpolation of the interaction mean free paths (stored in GPU global memory). |

| | | |
|---|---|---|
| in | *mfp_table_b* | Second element for the linear interpolation of the interaction mean free paths (stored in GPU global memory). |
| in | *rayleigh_-table* | Pointer to the table with the data required by the Rayleigh interaction sampling, stored in GPU global memory. |
| in | *compton_-table* | Pointer to the table with the data required by the Compton interaction sampling, stored in GPU global memory. |
| in,out | *image* | Pointer to the image vector in the GPU global memory. |
| in,out | *dose* | Pointer to the array containing the 3D voxel dose (and its uncertainty) in the GPU global memory. |

Definition at line 135 of file MC-GPU_kernel_v1.3.cu.

References dose_ROI_x_max_CONST, linear_interp::e0, GCOa(), GRAa(), linear_-interp::ide, init_PRNG(), locate_voxel(), MAX_MATERIALS, mfp_table_data_CONS-T, rayleigh_struct::pmax, ranecu(), ranecu_double(), rotate_double(), source(), tally-_image(), tally_materials_dose(), tally_voxel_energy_deposition(), float2::x, float3::x, float2::y, float3::y, and float3::z.

Referenced by main().

#### 5.3.4.26 void **trim_name** ( char ∗ *input_line,* char ∗ *file_name* )

Extract a file name from an input text line, trimming the initial blanks, trailing comment (#) and stopping at the first blank (the file name should not contain blanks).

**Parameters**

| | | |
|---|---|---|
| in | *input_line* | Input sentence with blanks and a trailing comment |
| out | *file_name* | Trimmed file name |

Definition at line 1840 of file MC-GPU_v1.3.cu.

Referenced by read_input().

#### 5.3.4.27 void **update_seed_PRNG** ( int *batch_number,* unsigned long long int *total_histories,* int ∗ *seed* ) [inline]

Initialize the first seed of the pseudo-random number generator (PRNG) RANECU to a position far away from the previous history (leap frog technique).

This function is equivalent to "init_PRNG" but only updates one of the seeds.

Note that if we use the same seed number to initialize the 2 MLCGs of the PRNG we can only warranty that the first MLCG will be uncorrelated for each value generated by "update_seed_PRNG". There is a tiny chance that the final PRNs will be correlated because the leap frog on the first MLCG will probably go over the repetition cycle of the

MLCG, which is much smaller than the full RANECU. But any correlataion is extremely unlikely. Function "init_PRNG" doesn't have this issue.

**Parameters**

| | | |
|---|---|---|
| `in` | *batch_-number* | Elements to skip (eg, MPI thread_number). |
| `in` | *total_-histories* | Histories to skip. |
| `in,out` | *seed* | Initial PRNG seeds; returns the updated seed. |

Definition at line 3356 of file MC-GPU_v1.3.cu.

References a1_RANECU, abMODm(), LEAP_DISTANCE, and m1_RANECU.

Referenced by main().

### 5.3.5 Variable Documentation

#### 5.3.5.1 struct detector_struct detector_data_CONST

Global variable to be stored in the GPU constant memory defining the x-ray detector.

Definition at line 329 of file MC-GPU_v1.3.h.

#### 5.3.5.2 short int dose_ROI_x_max_CONST

Definition at line 311 of file MC-GPU_v1.3.h.

Referenced by main(), tally_voxel_energy_deposition(), and track_particles().

#### 5.3.5.3 short int dose_ROI_x_min_CONST

Global variable to be stored in the GPU constant memory defining the coordinates of the dose deposition region of interest.

Definition at line 311 of file MC-GPU_v1.3.h.

Referenced by main(), and tally_voxel_energy_deposition().

#### 5.3.5.4 short int dose_ROI_y_max_CONST

Definition at line 311 of file MC-GPU_v1.3.h.

Referenced by main(), and tally_voxel_energy_deposition().

#### 5.3.5.5 short int dose_ROI_y_min_CONST

Definition at line 311 of file MC-GPU_v1.3.h.

Referenced by main(), and tally_voxel_energy_deposition().

#### 5.3.5.6 short int dose_ROI_z_max_CONST

Definition at line 311 of file MC-GPU_v1.3.h.

Referenced by main(), and tally_voxel_energy_deposition().

#### 5.3.5.7 short int dose_ROI_z_min_CONST

Definition at line 311 of file MC-GPU_v1.3.h.

Referenced by main(), and tally_voxel_energy_deposition().

#### 5.3.5.8 struct linear_interp mfp_table_data_CONST

Global variable to be stored in the GPU constant memory defining the linear interpolation data.

Definition at line 335 of file MC-GPU_v1.3.h.

Referenced by main(), and track_particles().

#### 5.3.5.9 struct source_struct source_data_CONST

Global variable to be stored in the GPU constant memory defining the x-ray source.

Definition at line 323 of file MC-GPU_v1.3.h.

#### 5.3.5.10 struct source_energy_struct source_energy_data_CONST

Global variable to be stored in the GPU constant memory defining the source energy spectrum.

Definition at line 341 of file MC-GPU_v1.3.h.

Referenced by main(), and source().

#### 5.3.5.11 struct voxel_struct voxel_data_CONST

Global variable to be stored in the GPU constant memory defining the size of the voxel phantom.

Definition at line 317 of file MC-GPU_v1.3.h.

Referenced by locate_voxel(), main(), and source().