GGEMS

version 1.2

GGEMS Collaboration

avril 19, 2022

Contents

Welcome to GGEMS Documentation

GGEMS is an advanced Monte Carlo simulation platform using CPU and GPU architecture targeting medical applications (imaging and particle therapy). This code is based on the well-validated Geant4 physics model and capable to be executed in both CPU and GPU devices using the OpenCL library.

This documentation is divided into three parts:

First, an introduction to GGEMS and the informations are given in order to install your GGEMS environment.

Second, for a standard user, informations about all GGEMS potentials are given. Examples and tools are also illustrated and explained. Command lines are listed using both C++ and python instructions.

And finally, a more detailed description concerning GGEMS core for advanced user. The purpose of this part is to give enough informations to an user to implement a custom part of code in GGEMS.

Introduction

GGEMS (GPU Geant4-based Monte Carlo Simulations) is an advanced Monte Carlo simulation platform using the OpenCL library managing CPU and GPU architecture. GGEMS is written in C++, and can be used using python commands. The reader is assumed to have some basic knowledge of object-oriented programming using C++.

Well-validated Geant4 physic models are used in GGEMS and implemented using OpenCL.

The aim of GGEMS is to provide a fast simulation platform for imaging application and particle therapy. To favor speed of computation, GGEMS is not a very generic platform as Geant4 or GATE. For very realistic simulation with lot of information results, Geant4 and GATE are still recommended.

GGEMS features:

- Photon particle tracking
- Multithreaded CPU
- GPU
- Multi devices (GPUs+CPU) approach
- Single or double float precision for dosimetry application
- External X-ray source
- Navigation in simple box volume or voxelized volume
- Flat or curved detector for CBCT/CT application

GGEMS medical applications:

- CT/CBCT imaging (standard, dual-energy)
- External radiotherapy (IMRT and VMAT)
- Portal imaging from LINAC system

In the next GGEMS releases, the aim is to implement the following applications and features:

- Visualization
- Positron particle tracking
- Electron particle tracking
- Mesh volume
- Voxelized source
- PET imaging
- SPECT imaging
- Intra-operative radiotherapy (brachytherapy and intrabeam)
- AMD architecture validation
- MacOS system validation

Requirements

GGEMS is a multiplatform application using OpenCL and could be used using OpenGL.

Operating Systems

Supported

- Linux (Any distribution, Debian, Ubuntu, …)
- Windows 10 & 11

Maybe Supported

GGEMS should work on a MacOS system, but this has not been tested

- MacOS X

Python Version

GGEMS supports the following version

- Python 3.6+

OpenCL Version

GGEMS validated using the following version

- OpenCL 1.2
- OpenCL 2.0 & 3.0 can be used, but GGEMS is based on OpenCL 1.2

Hardware

GGEMS can be used on lot of different hardwares such as CPU, GPU and graphic cards included in CPU (Intel HD Graphics)

Supported

- Intel (CPU + HD Graphics)
- NVIDIA

Maybe Supported

GGEMS should work on the following hardware, but not tested yet

- AMD

Building & Installing

Note

GGEMS is written in C++ and uses the OpenCL library. However, GGEMS can also be used by calling python v3 functions. Python is not mandatory, C++ is sufficient. Many examples in C++ and Python are written in this manual

Prerequisites

GGEMS is based on the OpenCL library. For each platform (NVIDIA, Intel and AMD), you must install the corresponding drivers.

NVIDIA

Linux & Windows

The NVIDIA driver as well as the CUDA library must be installed to use GGEMS on an NVIDIA platform. The recommended method for this installation is to download CUDA from the official NVIDIA website (https://developer.nvidia.com/cuda-downloads) and follow their instructions.

Warning

The CUDA library is not used by GGEMS. Only the OpenCL library provided with CUDA is used.

Warning

It is recommended to install CUDA yourself with the link provided by NVIDIA. On linux, the 'apt' program is very useful but can produce some inconvenience.

INTEL

Linux & Windows

To use GGEMS on an Intel platform (CPU or GPU) you must install the driver. For this it is recommended to install the driver provided by Intel oneAPI Base Toolkit (https://www.intel.com/content/www/us/en/developer/tools/oneapi/base-toolkit.html). The library will be installed with the Intel compiler and the other libraries.

AMD

Linux & Windows

The AMD platform has not yet been tested with GGEMS. The library should be compatible or only a few changes in the code should be made. The driver can be downloaded with the following link (https://www.amd.com/en/support). Do not hesitate to contact us if you need help compiling GGEMS on an AMD platform.

OpenGL visualization

Since version 1.2 of GGEMS, the OpenGL library can be used to visualize the simulation in 3D space. OpenGL can be used by both Windows and Linux. GGEMS is also based on 3 other libraries:

- GLFW that is an Open Source, multi-platform library for OpenGL, OpenGL ES and Vulkan development on the desktop. It provides a simple API for creating windows, contexts and surfaces, receiving input and events.
- GLEW that is a cross-platform open-source C/C++ extension loading library.
- GLM that is a header only C++ mathematics library for graphics software based on the OpenGL Shading Language (GLSL) specifications.

Important

For linux users, GLEW library must be installed from source (glew-XXX.zip) (http://glew.sourceforge.net/). When activating OpenGL, it’s mandatory to link GGEMS and libGLEW.a static library.

Warning

For linux users, GLFW and GLM libraries can be installed using the ‘apt’ program for example.

Warning

For Windows users the libraries should be downloaded from their respective website and installed if possible in the standard location C:\Program Files (x86)

GGEMS Installation

CMAKE is required to install GGEMS. The minimum version of CMAKE is 3.13.

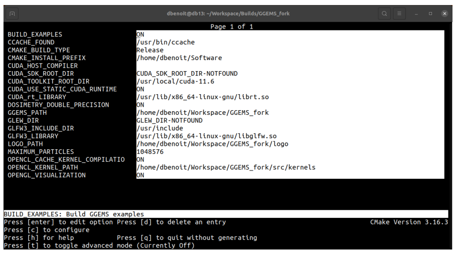
Linux

Cloning GGEMS using the following command:

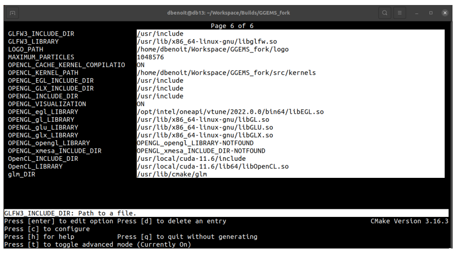
```
$ git clone https://github.com/GGEMS/ggems.git
```

Creating a folder named GGEMS_build (or another name), and launch the command ‘ccmake’:

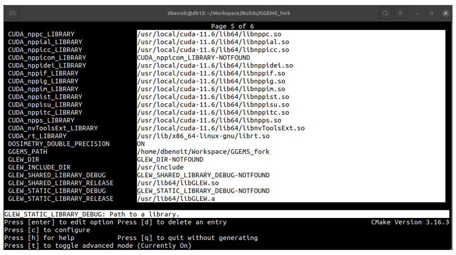
```
$ mkdir GGEMS_build
$ mkdir GGEMS_install
$ cd GGEMS_build
$ ccmake ../ggems
```



```
                                 Page 1 of 1
 BUILD_EXAMPLES                  ON
 CCACHE_FOUND                    /usr/bin/ccache
 CMAKE_BUILD_TYPE                Release
 CMAKE_INSTALL_PREFIX            /home/dbenoit/Software
 CUDA_HOST_COMPILER
 CUDA_SDK_ROOT_DIR               CUDA_SDK_ROOT_DIR-NOTFOUND
 CUDA_TOOLKIT_ROOT_DIR           /usr/local/cuda-11.6
 CUDA_USE_STATIC_CUDA_RUNTIME    ON
 CUDA_rt_LIBRARY                 /usr/lib/x86_64-linux-gnu/librt.so
 DOSIMETRY_DOUBLE_PRECISION      ON
 GGEMS_PATH                      /home/dbenoit/Workspace/GGEMS_fork
 GLEW_DIR                        GLEW_DIR-NOTFOUND
 GLFW3_INCLUDE_DIR               /usr/include
 GLFW3_LIBRARY                   /usr/lib/x86_64-linux-gnu/libglfw.so
 LOGO_PATH                       /home/dbenoit/Workspace/GGEMS_fork/logo
 MAXIMUM_PARTICLES               1048576
 OPENCL_CACHE_KERNEL_COMPILATIO  ON
 OPENCL_KERNEL_PATH              /home/dbenoit/Workspace/GGEMS_fork/src/kernels
 OPENGL_VISUALIZATION            ON



BUILD_EXAMPLES: Build GGEMS examples
Press [enter] to edit option Press [d] to delete an entry              CMake Version 3.16.3
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently Off)
```



```
                                 Page 6 of 6
 GLFW3_INCLUDE_DIR               /usr/include
 GLFW3_LIBRARY                   /usr/lib/x86_64-linux-gnu/libglfw.so
 LOGO_PATH                       /home/dbenoit/Workspace/GGEMS_fork/logo
 MAXIMUM_PARTICLES               1048576
 OPENCL_CACHE_KERNEL_COMPILATIO  ON
 OPENCL_KERNEL_PATH              /home/dbenoit/Workspace/GGEMS_fork/src/kernels
 OPENGL_EGL_INCLUDE_DIR          /usr/include
 OPENGL_GLX_INCLUDE_DIR          /usr/include
 OPENGL_INCLUDE_DIR              /usr/include
 OPENGL_VISUALIZATION            ON
 OPENGL_egl_LIBRARY              /opt/intel/oneapi/vtune/2022.0.0/bin64/libEGL.so
 OPENGL_gl_LIBRARY               /usr/lib/x86_64-linux-gnu/libGL.so
 OPENGL_glu_LIBRARY              /usr/lib/x86_64-linux-gnu/libGLU.so
 OPENGL_glx_LIBRARY              /usr/lib/x86_64-linux-gnu/libGLX.so
 OPENGL_opengl_LIBRARY           OPENGL_opengl_LIBRARY-NOTFOUND
 OPENGL_xmesa_INCLUDE_DIR        OPENGL_xmesa_INCLUDE_DIR-NOTFOUND
 OpenCL_INCLUDE_DIR              /usr/local/cuda-11.6/include
 OpenCL_LIBRARY                  /usr/local/cuda-11.6/lib64/libOpenCL.so
 glm_DIR                         /usr/lib/cmake/glm



GLFW3_INCLUDE_DIR: Path to a file.
Press [enter] to edit option Press [d] to delete an entry              CMake Version 3.16.3
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently On)
```



```
                                 Page 5 of 6
 CUDA_nppc_LIBRARY               /usr/local/cuda-11.6/lib64/libnppc.so
 CUDA_nppial_LIBRARY             /usr/local/cuda-11.6/lib64/libnppial.so
 CUDA_nppicc_LIBRARY             /usr/local/cuda-11.6/lib64/libnppicc.so
 CUDA_nppicom_LIBRARY            CUDA_nppicom_LIBRARY-NOTFOUND
 CUDA_nppidei_LIBRARY            /usr/local/cuda-11.6/lib64/libnppidei.so
 CUDA_nppif_LIBRARY              /usr/local/cuda-11.6/lib64/libnppif.so
 CUDA_nppig_LIBRARY              /usr/local/cuda-11.6/lib64/libnppig.so
 CUDA_nppim_LIBRARY              /usr/local/cuda-11.6/lib64/libnppim.so
 CUDA_nppist_LIBRARY             /usr/local/cuda-11.6/lib64/libnppist.so
 CUDA_nppisu_LIBRARY             /usr/local/cuda-11.6/lib64/libnppisu.so
 CUDA_nppitc_LIBRARY             /usr/local/cuda-11.6/lib64/libnppitc.so
 CUDA_npps_LIBRARY               /usr/local/cuda-11.6/lib64/libnpps.so
 CUDA_nvToolsExt_LIBRARY         /usr/local/cuda-11.6/lib64/libnvToolsExt.so
 CUDA_rt_LIBRARY                 /usr/lib/x86_64-linux-gnu/librt.so
 DOSIMETRY_DOUBLE_PRECISION      ON
 GGEMS_PATH                      /home/dbenoit/Workspace/GGEMS_fork
 GLEW_DIR                        GLEW_DIR-NOTFOUND
 GLEW_INCLUDE_DIR                /usr/include
 GLEW_SHARED_LIBRARY_DEBUG       GLEW_SHARED_LIBRARY_DEBUG-NOTFOUND
 GLEW_SHARED_LIBRARY_RELEASE     /usr/lib64/libGLEW.so
 GLEW_STATIC_LIBRARY_DEBUG       GLEW_STATIC_LIBRARY_DEBUG-NOTFOUND
 GLEW_STATIC_LIBRARY_RELEASE     /usr/lib64/libGLEW.a

GLEW_STATIC_LIBRARY_DEBUG: Path to a library.
Press [enter] to edit option Press [d] to delete an entry                CMake Version 3.16.3
Press [c] to configure
Press [h] for help           Press [q] to quit without generating
Press [t] to toggle advanced mode (Currently On)
```

Note

By default, the GNU C++ compiler is used on Linux. LLVM CLANG or Intel DPC++/C++ can also be used. The compiler can be modify using the CMakeLists.txt file.

After checking the installation parameters in the CMAKE window, you can run the installation commands:

```
$ make -jN
$ make install
```

To be able to use GGEMS correctly in a linux console, you must enter the right values for the environment variables (in the .bashrc file for example on Linux)

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:YOUR_PATH/GGEMS_install/ggems/lib
export PYTHONPATH=$PYTHONPATH:YOUR_PATH/GGEMS_install/ggems/python_module
export PYTHONPATH=$PYTHONPATH:YOUR_PATH/GGEMS_install/ggems/lib
```

GGEMS is now installed on your machine. To verify the installation, you can try the examples or call GGEMS in a Python console.

```
from ggems import *
opencl_manager = GGEMSOpenCLManager()
opencl_manager.print_infos()
opencl_manager.clean()
exit()
```

Windows

Note

The following guide assumes you are using the Windows console (cmd.exe), power shell can also be used. Visual C++ is the default compiler.

Important

Only Visual C++ (CL), LLVM CLANG and Intel DPC++/C++ are validated on Windows. GNU GCC is not validated.

Visual C++ is assumed to be properly installed and configured on your computer. A simple batch script (for instance named set_mvsc.bat) allows you to configure Visual C++. The following example is configured for Visual Studio 2022.

```
@echo OFF
if "%VCTOOLKIT_VARS_ARE_SET%" == "true" goto done
```

```
echo --- Setting Microsoft Visual C++ Toolkit 2022 environment variables... ---

call "C:\Program Files\Microsoft Visual Studio\2022\Community\VC\Auxiliary\Build\vcvarsall.b

set PATH="%VCToolkitInstallDir%"\bin;%PATH%
set INCLUDE="%VCToolkitInstallDir%"\include;%INCLUDE%
set LIB="%VCToolkitInstallDir%"\lib;%LIB%

set VCTOOLKIT_VARS_ARE_SET=true
echo Done.
:done
```

The previous batch script can be launched with the command:

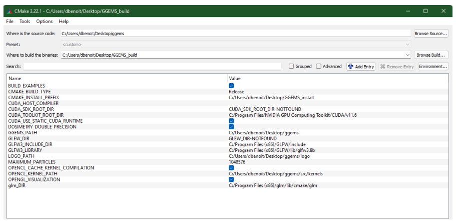
```
C:\Users\XXX> set_mvsc.bat
```

Cloning GGEMS using the following command:

```
$ git clone https://github.com/GGEMS/ggems.git
```

Creating a folder named GGEMS_build (or another name), and launch the command ‘cmake-gui’:

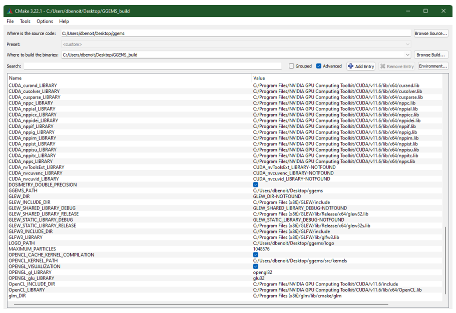
```
C:\Users\XXX> md GGEMS_build
C:\Users\XXX> md GGEMS_install
C:\Users\XXX> cd GGEMS_build
C:\Users\XXX\GGEMS_build> cmake-gui
```



CMake 3.22.1 - C:/Users/dbenoit/Desktop/GGEMS_build

Where is the source code: C:/Users/dbenoit/Desktop/ggems
Preset: <custom>
Where to build the binaries: C:/Users/dbenoit/Desktop/GGEMS_build
Search:
☐ Grouped ☐ Advanced

Name	Value
BUILD_EXAMPLES	✓
CMAKE_BUILD_TYPE	Release
CMAKE_INSTALL_PREFIX	C:/Users/dbenoit/Desktop/GGEMS_install
CUDA_HOST_COMPILER	
CUDA_SDK_ROOT_DIR	CUDA_SDK_ROOT_DIR-NOTFOUND
CUDA_TOOLKIT_ROOT_DIR	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6
CUDA_USE_STATIC_CUDA_RUNTIME	✓
DOSIMETRY_DOUBLE_PRECISION	✓
GGEMS_PATH	C:/Users/dbenoit/Desktop/ggems
GLEW_DIR	GLEW_DIR-NOTFOUND
GLFW3_INCLUDE_DIR	C:/Program Files (x86)/GLFW/include
GLFW3_LIBRARY	C:/Program Files (x86)/GLFW/lib/glfw3.lib
LOGO_PATH	C:/Users/dbenoit/Desktop/ggems/logo
MAXIMUM_PARTICLES	1048576
OPENCL_CACHE_KERNEL_COMPILATION	✓
OPENCL_KERNEL_PATH	C:/Users/dbenoit/Desktop/ggems/src/kernels
OPENGL_VISUALIZATION	✓
glm_DIR	C:/Program Files (x86)/glm/lib/cmake/glm



CMake 3.22.1 - C:/Users/dbenoit/Desktop/GGEMS_build

Where is the source code: C:/Users/dbenoit/Desktop/ggems
Preset: <custom>
Where to build the binaries: C:/Users/dbenoit/Desktop/GGEMS_build
Search:
☐ Grouped ✓ Advanced

Name	Value
CUDA_curand_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/curand.lib
CUDA_cusolver_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/cusolver.lib
CUDA_cusparse_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/cusparse.lib
CUDA_nppc_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppc.lib
CUDA_nppial_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppial.lib
CUDA_nppicc_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppicc.lib
CUDA_nppidei_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppidei.lib
CUDA_nppif_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppif.lib
CUDA_nppig_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppig.lib
CUDA_nppim_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppim.lib
CUDA_nppist_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppist.lib
CUDA_nppisu_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppisu.lib
CUDA_nppitc_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/nppitc.lib
CUDA_npps_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/npps.lib
CUDA_nvToolsExt_LIBRARY	CUDA_nvToolsExt_LIBRARY-NOTFOUND
CUDA_nvcuvenc_LIBRARY	CUDA_nvcuvenc_LIBRARY-NOTFOUND
CUDA_nvcuvid_LIBRARY	CUDA_nvcuvid_LIBRARY-NOTFOUND
DOSIMETRY_DOUBLE_PRECISION	✓
GGEMS_PATH	C:/Users/dbenoit/Desktop/ggems
GLEW_DIR	GLEW_DIR-NOTFOUND
GLEW_INCLUDE_DIR	C:/Program Files (x86)/GLEW/include
GLEW_SHARED_LIBRARY_DEBUG	GLEW_SHARED_LIBRARY_DEBUG-NOTFOUND
GLEW_SHARED_LIBRARY_RELEASE	C:/Program Files (x86)/GLEW/lib/Release/x64/glew32.lib
GLEW_STATIC_LIBRARY_DEBUG	GLEW_STATIC_LIBRARY_DEBUG-NOTFOUND
GLEW_STATIC_LIBRARY_RELEASE	C:/Program Files (x86)/GLEW/lib/Release/x64/glew32s.lib
GLFW3_INCLUDE_DIR	C:/Program Files (x86)/GLFW/include
GLFW3_LIBRARY	C:/Program Files (x86)/GLFW/lib/glfw3.lib
LOGO_PATH	C:/Users/dbenoit/Desktop/ggems/logo
MAXIMUM_PARTICLES	1048576
OPENCL_CACHE_KERNEL_COMPILATION	✓
OPENCL_KERNEL_PATH	C:/Users/dbenoit/Desktop/ggems/src/kernels
OPENGL_VISUALIZATION	✓
OPENGL_gl_LIBRARY	opengl32
OPENGL_glu_LIBRARY	glu32
OpenCL_INCLUDE_DIR	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/include
OpenCL_LIBRARY	C:/Program Files/NVIDIA GPU Computing Toolkit/CUDA/v11.6/lib/x64/OpenCL.lib
glm_DIR	C:/Program Files (x86)/glm/lib/cmake/glm

Final step, compilation and installation using nmake or ninja.

```
C:\Users\XXX\GGEMS_build> nmake install
```

or

```
C:\Users\XXX\GGEMS_build> ninja install
```

Note

For multi-processor compilation it is recommended to use Ninja on Windows, NMake only using 1 processor.

To work correctly the PATH and PYTHONPATH environment variables must be initialized according to your installation path. This can be done by writing a simple batch file for example, here named set_ggems.bat.

```
@echo OFF
if "%GGEMS_VARS_ARE_SET%" == "true" goto done

echo --- Setting GGEMS... ---
set PYTHONPATH=%PYTHONPATH%;C:\Users\XXX\GGEMS_install\ggems\python_module
set PYTHONPATH=%PYTHONPATH%;C:\Users\XXX\GGEMS_install\ggems\lib
set PATH=%PATH%;C:\Users\XXX\GGEMS_install\ggems\lib

set GGEMS_VARS_ARE_SET=true
echo Done.
:done
```

GGEMS is now installed on your machine. To verify the installation, you can try the examples or call GGEMS in a Python console.

```
from ggems import *
opencl_manager = GGEMSOpenCLManager()
opencl_manager.print_infos()
opencl_manager.clean()
exit()
```

CMAKE GGEMS Parameters

BUILD_EXAMPLES

By default this option is initialized to ON. During the compilation of GGEMS, the examples will also be compiled and installed.

CMAKE_INSTALL_PREFIX

Path to your installation folder.

DOSIMETRY_DOUBLE_PRECISION

By default this option is initialized to ON. This is an option that only impacts the dose calculation for dosimetry applications. Double precision calculations in dosimetry are highly recommended.

MAXIMUM_PARTICLES

In the GGEMS library, a certain number of particles are simulated at the same time. By default this number is 1048576. Depending on the performance of the graphics card this number may be larger for better performance.

Important

It is important to understand that this number is not the number of particles simulated, but the number of particles simulated at the same time by the OpenCL kernels.

OPENCL_CACHE_KERNEL_COMPILATION

By default this option is initialized to ON. When compiling OpenCL kernels while GGEMS is running, the kernels are cached. If you need to modify the code yourself in OpenCL kernels it is recommended to set this variable to OFF.

OPENGL_VISUALIZATION

By default this option is initialized to OFF. If you want to visualize your simulation, initialize this variable to ON, GGEMS will then use the OpenGL library.

Documentation

In the documentation dedicated to users, only the command lines written in Python are presented. For C++ commands, refer directly to the GGEMS examples provided when downloading the library.

Multi-Devices

The GGEMS library can use several devices during the simulation. There are two ways to do this, either give the device index or explicitly indicate the device(s).

- Using the device index:

```python
from ggems import *

opencl_manager = GGEMSOpenCLManager()
opencl_manager.set_device_index(0) # Activate device id 0
opencl_manager.set_device_index(2) # Activate device id 2, if it exists

opencl_manager.clean()
exit()
```

- Explicitly indicate the device(s):

```python
from ggems import *

opencl_manager = GGEMSOpenCLManager()
opencl_manager.set_device_to_activate('gpu', 'nvidia') # Activate all NVIDIA GPU only
opencl_manager.set_device_to_activate('gpu', 'intel') # Activate all Intel GPU only
opencl_manager.set_device_to_activate('gpu', 'amd') # Activate all AMD GPU only
opencl_manager.set_device_to_activate('cpu') # Activate cpu only
opencl_manager.set_device_to_activate('all') # Activate all found devices
opencl_manager.set_device_to_activate('0;2') # Activate devices 0 and 2

opencl_manager.clean()
exit()
```

OpenGL Visualization

Since GGEMS v1.2, it is possible to enable graphical visualization using the OpenGL library. In GGEMS several types of volumes can be drawn like spheres, cones, cylinders and voxelized volumes.

To be able to use OpenGL you must first call the singleton and then initialize some parameters.

Regarding the colors in GGEMS, it is possible to use defined colors such as:

- black
- blue
- lime
- cyan
- red
- magenta
- yellow
- white
- gray
- silver
- maroon

- olive
- green
- purple
- teal
- navy

Important

It is important to initialize the OpenGL settings at the start of the simulation. All volumes, and all sources must be declared after this initialization

```
from ggems import *

opengl_manager = GGEMSOpenGLManager()

# Multisample anti-aliasing, can be 1, 2, 4 or 8
opengl_manager.set_msaa(8)

# Background color for OpenGL window
opengl_manager.set_background_color('black')

# Draw axis X, Y and Z
opengl_manager.set_draw_axis(True)

# Output folder storing OpenGL images you want to save (*.png format)
opengl_manager.set_image_output('axis')

# Number of maximum particles drawn in the OpenGL viewport (max: 65536)
opengl_manager.set_displayed_particles(128)

# Change the color of photons, either using a color defined in GGEMS or using RGB indices
opengl_manager.set_particle_color('gamma', 152, 251, 152)
# opengl_manager.set_particle_color('gamma', color_name='red')

# Set the size of your world surrounding your simulation. It is important that this is large
opengl_manager.set_world_size(3.0, 3.0, 3.0, 'm')

# Size of OpenGL viewport
opengl_manager.set_window_dimensions(500, 500)

# Initialize OpenGL
opengl_manager.initialize()

# Show OpenGL window outside GGEMS simulation
opengl_manager.display()
```

If you have defined a navigator (phantom or detector) it is also possible to change the color of the material either by using a defined color or by using RGB indices. It is also possible to disable the color.

```
# We suppose you defined a phantom before composed by air and water
phantom.set_material_visible('Air', True) # or False if you do not want to draw the air voxe
phantom.set_material_color('Water', color_name='blue')

# We suppose you define a CBCT before composed by GOS
cbct_detector.set_material_color('GOS', 255, 0, 0) # Custom color using RGB
#cbct_detector.set_material_color('GOS', color_name='red') # Using registered color
```

Navigators

Particles are only tracked in browsers. In GGEMS there are two types of navigators:

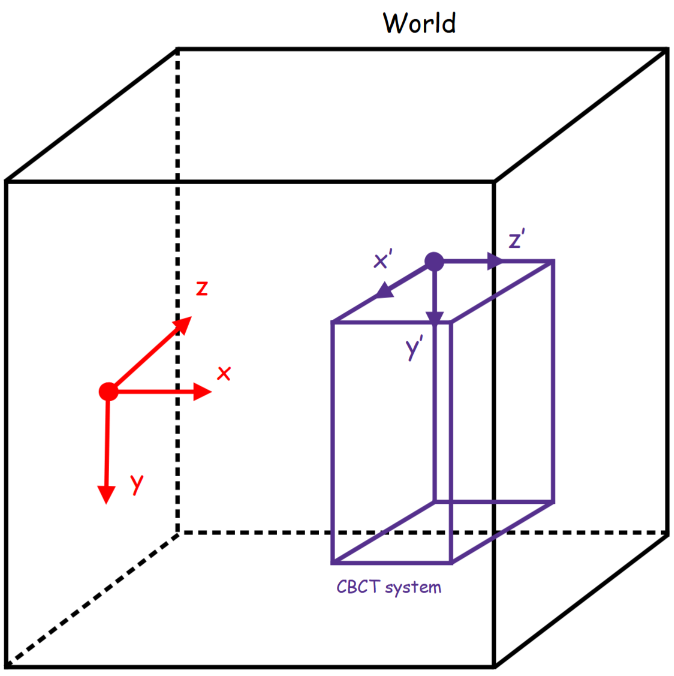
- System (detector)
- Phantom (object or patient)

For each navigator, three types of elements are associated:

- a solid: geometry of the navigator
- a list of materials
- physical processes

Systems

At the moment in GGEMS only CT and CBCT systems are available. The detector is made up of pixels assembled in a module. The figure below shows the reference axes of the world (in red) as well as a CT/CBCT system with its axes (in purple).



A CT/CBCT system is created using the following line:

```
cbct_system = GGEMSCTSystem('detector') # detector is a custom name for your system
```

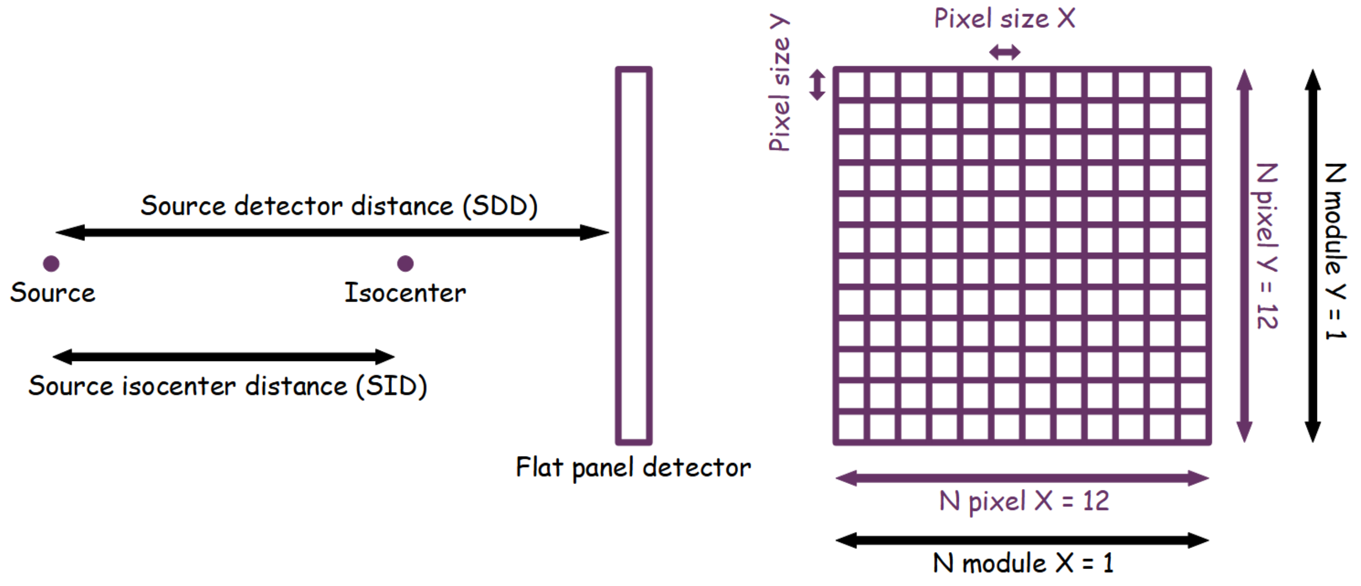
Types of CT/CBCT detector are:

- flat panel
- curved

Flat panel

This type of geometry is designed mainly for CBCT systems.

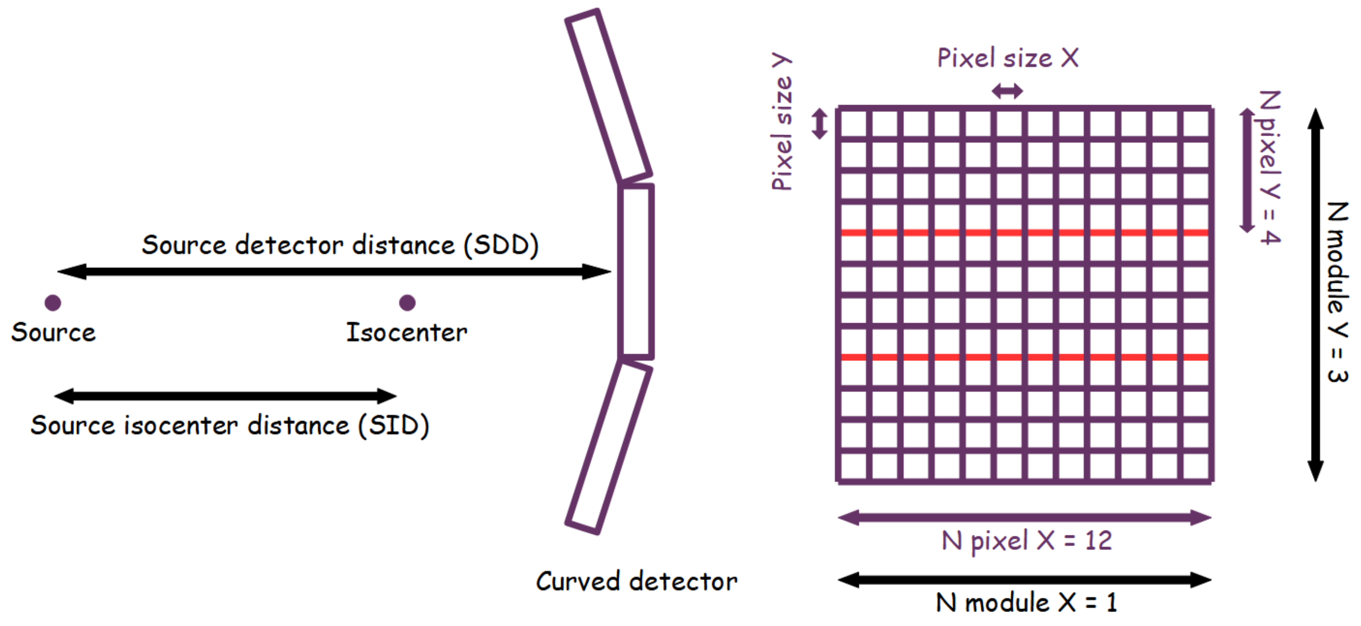
```
cbct_system.set_ct_type('flat')
```



Curved

This type of geometry is well suited for CT systems.

```
cbct_system.set_ct_type('curved')
```



Note

For curved geometry, the angle between modules is automatically calculated. There is no gap between the modules. The center of rotation of the system is the center of the world.

For each type of detector, the number of modules, the number of detection elements inside the module and their respective sizes are set as following:

```
cbct_system.set_number_of_modules(1, 3)
cbct_system.set_number_of_detection_elements(12, 4, 1)
cbct_system.set_size_of_detection_elements(1.0, 1.0, 1.0, 'mm')
```

A detector can be composed by only one type of material:

```
cbct_system.set_material('GOS')
```

An energy detection threshold can also be specified:

```
cbct_system.set_threshold(10.0, 'keV')
```

Source isocenter distance (SID) and source detector distance (SDD) is set with the following commands:

```
# Do not forget to add half size of detection element !!!
cbct_system.set_source_detector_distance(1500.5, 'mm')
cbct_system.set_source_isocenter_distance(900.0, 'mm')
```

Note

The position of the detector is calculated according to the values of the SID and SDD

A CT/CBCT system can be rotated around the world axis with the following command:

```
# 40 degree rotation around Z world axis
cbct_system.set_rotation(0.0, 0.0, 40.0, 'deg')
```

A CT/CBCT system can be translated along the world axis with the following command:

```
# 400 mm translation along Z world axis
cbct_system.set_global_system_position(0.0, 0.0, 400.0, 'mm');
```

The final projection including all photon interactions is saved in a file in MHD format, and scattered photons can be also save in another file, and we also give users the possibility to save the photons diffused for their needs:

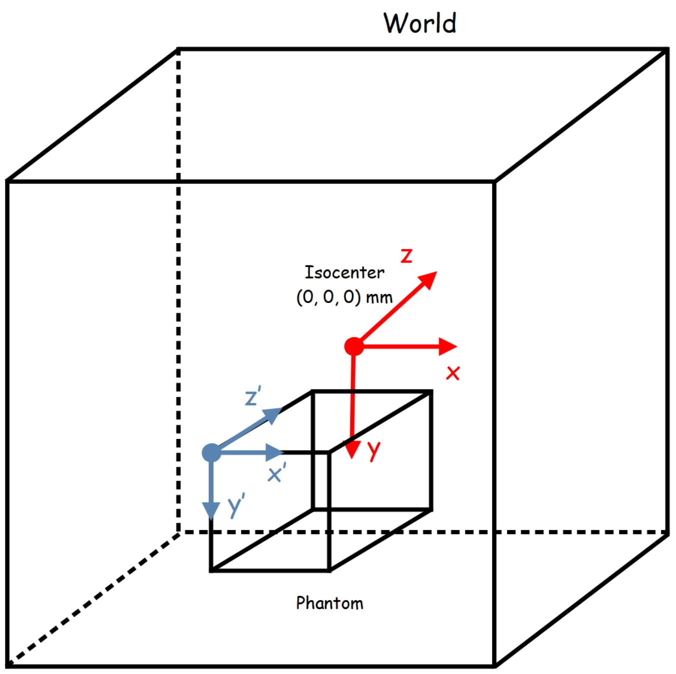
```
cbct_system.save('projection')
cbct_system.store_scatter(True)
```

To enable detector visualization and change the default color:

```
cbct_detector.set_visible(True)
cbct_detector.set_material_color('GOS', 255, 0, 0) # Custom color using RGB
cbct_detector.set_material_color('GOS', color_name='red') # Or using registered color
```

Phantoms

For the moment in GGEMS only voxelized volumes are available to define a phantom. The phantom must be in an MHD format associated with a TXT file storing the labels for the materials. The axes of the phantom correspond to the axes of the world.



To create a phantom you have to give it a name and load a file containing the phantom information and another file containing the material label:

```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('phantom.mhd', 'range_phantom.txt')
```

The phantom can be positioned and rotated in space using the following commands:

```
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')
```

If OpenGL is enabled the phantom can be viewed or disabled. The color of the materials composing the phantom can be modified using the RGB indices or the default colors. Regarding the air it is rather advisable not to display it in order to save memory:

```
phantom.set_visible(True)
phantom.set_material_visible('Air', False)
phantom.set_material_color('Water', color_name='blue')
```

World

Outside the navigator, particles are not tracked. However, a tool has been developed in GGEMS to record the photons leaving the navigator. Particles are projected into the world using a DDA algorithm.

The world module is ‘GGEMSWorld’:

```
world = GGEMSWorld()
```

After creating a GGEMSWorld, the dimension of the world and size of voxel can be set:

```
world.set_dimensions(200, 200, 200)
world.set_element_sizes(10.0, 10.0, 10.0, 'mm')
```

For world output, there are many informations that the user can save such as: energy and energy squared of photon crossing voxel, photon momentum and fluence (photon tracking):

```
world.set_output_basename('data/world')
world.energy_tracking(True)
world.energy_squared_tracking(True)
world.momentum(True)
world.photon_tracking(True)
```

Dosimetry

Dosimetry module can be activated to compute absorbed dose. For the moment only photons are simulated, electrons are ignored.

Dosimetry module is ‘GGEMSDosimetryCalculator’:

```
dosimetry = GGEMSDosimetryCalculator()
```

A navigator phantom must be attached to GGEMSDosimetryCalculator using:

```
dosimetry.attach_to_navigator('phantom')
```

Size of voxel (dosel) for dosimetry image could be set, if not the dosel size is the same than voxel phantom size:

```
dosimetry.set_dosel_size(0.5, 0.5, 0.5, 'mm')
```

Absorbed dose is computed in gray (Gy). By default dose in computed using materials in phantom. Otherwize the user can set water material everywhere in phantom.

```
dosimetry.water_reference(True)
```

A custom density threshold can be set. If density of phantom is below this threshold the dose value in 0 Gy.

```
dosimetry.minimum_density(0.1, 'g/cm3')
```

Since GGEMS v1.2, TLE method (Track Length Estimated) proposed in [Smekens2014] can be activated to improve statistics.

```
dosimetry.set_tle(is_tle)
```

Many informations can be registered such as: uncertainty values of the dose, the deposited energy in dosel, the squared of deposited energy in dosel and the number of interactions (hits) in dosel:

```
dosimetry.set_output('dosimetry')
dosimetry.uncertainty(True)
dosimetry.edep(True)
dosimetry.hit(True)
dosimetry.edep_squared(True)
```

There is a special output named ‘photon tracking’. This output registers the number of photons crossing a dosel. To use this option, the size of dosel has to be the same than the phantom voxel size, otherwize GGEMS will throw an error:

```
dosimetry.photon_tracking(True)
```

-

  F. Baldacci, A. Mittone, A. Bravin, P. Coan, F. Delaire, C. Ferrero, S. Gasilov, J. M. Létang, D. Sarrut, F. Smekens, et al., “A track length estimator method for dose calculations in low-energy x-ray irradiations: implementation, properties and performance”, Zeitschrift Fur Medizinische Physik, 2014.

Physical Processes & Range cuts

Physical Processes

The photon processes impletemented are:

- Compton scattering
- Photoelectric effect
- Rayleigh scattering

Each of these processes are extracted from Geant4 version 10.6. For more informations about physics, please read the documentation on the Geant4 website.

By using python, the singleton can be called with the following command:

```
processes_manager = GGEMSProcessesManager()
```

Important

Secondary particles (photon and electron) are not simulated yet. For Photoelectric effect, the photon is killed during the interaction and the energy is locally deposited, the fluorescence photon is not emitted.

Compton Scattering

The Geant4 model extracted is the 'G4KleinNishinaCompton' standard model. It is the fastest algorithm to simulate this process. Compton scattering is activated for all the navigators, or for a specific navigator.

```
# Activating Compton process for all navigators
processes_manager.add_process('Compton', 'gamma', 'all')

# Activating Compton process for a specific navigator name 'my_phantom'
processes_manager.add_process('Compton', 'gamma', 'my_phantom')
```

Photoelectric Effect

The Geant4 model extracted is the 'G4PhotoElectricEffect' standard model using Sandia tables. Photoelectric effect is activated for all the navigators, or for a specific navigator.

```
# Activating Photoelectric process for all navigators
processes_manager.add_process('Photoelectric', 'gamma', 'all')

# Activating Photoelectric process for a specific navigator name 'my_phantom'
processes_manager.add_process('Photoelectric', 'gamma', 'my_phantom')
```

Rayleigh Scattering

The Geant4 model extracted is the 'G4LivermoreRayleighModel' livermore model. Rayleigh scattering is activated for all the navigators, or for a specific navigator.

```
# Activating Rayleigh process for all navigators
processes_manager.add_process('Rayleigh', 'gamma', 'all')

# Activating Rayleigh process for a specific navigator name 'my_phantom'
processes_manager.add_process('Rayleigh', 'gamma', 'my_phantom')
```

Process Parameters Building

Cross sections are calculated during GGEMS initialization. The parameters for calculating the cross sections can be modified, but it is recommended to use the default ones. The parameters that can be modified are:

- Minimum energy of cross-section table
- Maximum energy of cross-section table
- Number of bins in cross-section table

Default parameters are defined as following:

```
processes_manager.set_cross_section_table_number_of_bins(220)
processes_manager.set_cross_section_table_energy_min(1.0, 'keV')
processes_manager.set_cross_section_table_energy_max(1.0, 'MeV')
```

Process Verbosity

Some informations about processes can be printed:

- Available processes
- Global informations about processes
- Cross-section values

The list of commands are:

```
processes_manager.print_available_processes()
processes_manager.print_infos()
processes_manager.print_tables(True)
```

Range Cuts

Cuts are defined for each particle in distance unit in a navigator or a specific navigator. During initialization cuts are converted in energy for each material in navigator. If the particle energy is below the cut, then this one is killed and the energy is locally deposited. By default cuts are 1 micron.

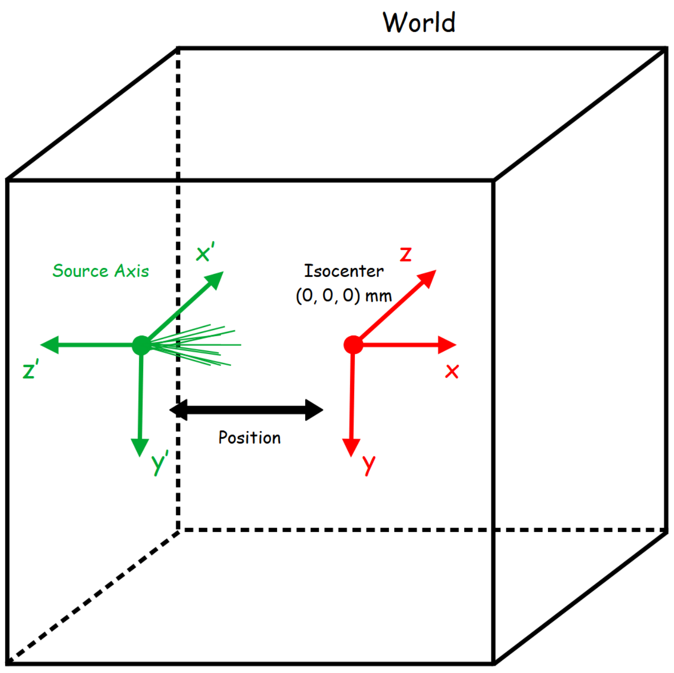
```
# For photon and all navigators
range_cuts_manager.set_cut('gamma', 0.1, 'mm', 'all')

# For photon and a specific navigator named 'my_phantom'
range_cuts_manager.set_cut('gamma', 0.1, 'mm', 'my_phantom')
```

Sources

In GGEMS, we develop a source for each application. For the moment only dosimetry and CT/CBCT applications are allowed. Only one source has been implemented: GGEMSXRaySource

X-ray Source

The X-Ray source is defined with a cone-beam geometry. The direction of the generated photons always point to the center of the world. The source has its own axes.



Create a source by choosing a name:

```
xray_source = GGEMSXRaySource('xray_source')
```

Particle type is only photon:

```
xray_source.set_source_particle_type('gamma')
```

Number of generated particles during a run:

```
xray_source.set_number_of_particles(1000000000)
```

Position and rotation of source are defined in global world axis. The X-Ray cone-beam source can be defined with an aperture angle.

```
xray_source.set_position(-595.0, 0.0, 0.0, 'mm')
xray_source.set_rotation(0.0, 0.0, 0.0, 'deg')
xray_source.set_beam_aperture(12.5, 'deg')
```

X-ray source can be defined with a focal spot size. If defined at (0, 0, 0) mm, it is similar to a point source.

```
xray_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
```

Important

The focal spot size is defined in source axis reference and not in global world reference.

The energy source can be defined using a single energy value or a spectrum included in a TXT file.

```
# Using a custom source
xray_source.set_polyenergy('data/spectrum_120kVp_2mmAl.dat')

# Monoenergetic source
xray_source.set_monoenergy(25.0, 'keV')
```

GGEMS Commands

Commands to manage GGEMS are listed here in detail:

```
# Create an GGEMS instance
ggems = GGEMS()
```

Lot of verboses are available using GGEMS:

```
# Infos about OpenCL environment
ggems.opencl_verbose(True)

# Infos about whole material database
ggems.material_database_verbose(True)

# Infos about navigator (system/phantom)
ggems.navigator_verbose(True)

# Infos about source
ggems.source_verbose(True)

# Infos about memory usage
ggems.memory_verbose(True)

# Infos about activated processe(s)
ggems.process_verbose(True)
```

```
# Infos about range cuts
ggems.range_cuts_verbose(True)

# Print infos about first random state and initial seed
ggems.random_verbose(True)

# Infos about elapsed time in kernels
ggems.profiling_verbose(True)

# Infos about a specific photon index (here photon index 12)
ggems.tracking_verbose(True, 12)
```

Important commands to initialize and run a GGEMS application:

```
# Initialization with a specific seed (here 9383) or let GGEMS get a random seed
ggems.initialize(9383)
# ggems.initialize() # random seed

# Running ggems
ggems.run()
```

Examples & Tools

Few examples and tools are provided for GGEMS users.

Note

Examples are compiled and installed when compilation option ‘BUILD_EXAMPLES’ is set to ON. C++ executables are installed in example folders.

Examples 0: Cross-Section Computation

The purpose of this example is to provide a tool computing cross-sections

```
$ python cross_sections.py [-h] [-d DEVICE] [-m MATERIAL] -p [PROCESS]-e [ENERGY] [-v VERBOS
-h/--help           Printing help into the screen
-d/--device         Setting OpenCL id
-m/--material       Setting one of material defined in GGEMS (Water, Air, ...)
-p/--process        Setting photon physical process (Compton, Rayleigh, Photoelectric)
-e/--energy         Setting photon energy in MeV
-v/--verbose        Setting level of verbosity
```

Verbosity level is defined in the range [0;3]. For a silent GGEMS execution, the level is set to 0, otherwise 3 for lot of informations.

```
GGEMSVerbosity(verbosity_level)

# Select a OpenCL device
opencl_manager.set_device_index(device_id)
```

Create a GGEMSMaterial instance then add a material. The initialization step is mandatory and compute all physical tables, and store them on an OpenCL device:

```
materials = GGEMSMaterials()
materials.add_material(material_name)
materials.initialize()
```

Create a GGEMSCrossSection instance and activate a process:

```
cross_sections = GGEMSCrossSections(materials)
cross_sections.add_process(process_name, 'gamma')
cross_sections.initialize()
```

For attenuation informations, create a GGEMSAttenuations:

```
attenuations = GGEMSAttenuations(materials, cross_sections)
attenuations.initialize();
```

Computing cross section value (in cm2.g-1) for a specific energy (in MeV):

```
cross_sections.get_cs(process_name, material_name, energy_MeV, 'MeV')

# Get attenuation value in cm-1
attenuations.get_mu(material_name, energy_MeV, 'MeV')
# Get energy attenuation value
attenuations.get_mu_en(material_name, energy_MeV, 'MeV')
```

Examples 1: Total Attenuation

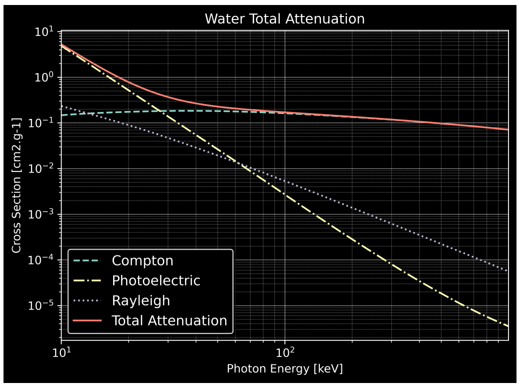
Warning

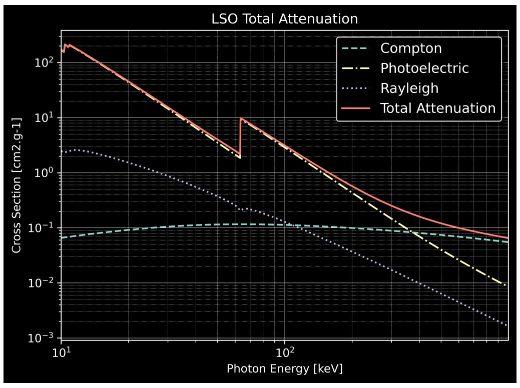
Example only available using python and matplotlib library

Example plotting the total attenuation of a material for energy between 0.01 MeV and 1 MeV. Commands are similar to example 0, and all physical processes are activated.

```
$ python total_attenuation.py [-h] [-d DEVICE] [-m MATERIAL] [-v VERBOSE]
-h/--help           Printing help into the screen
-d/--device         Setting OpenCL id
-m/--material       Setting one of material defined in GGEMS (Water, Air, ...)
-v/--verbose        Setting level of verbosity
```

Total attenuations for Water and LSO are shown below:





Examples 2: CT Scanner

In CT scanner example, a water box is simulated associated to a CT curved detector. One projection is computed simulating 1e9 particles.

```
$ python ct_scanner.py [-h] [-d DEVICE] [-b BALANCE] [-n N_PARTICLES] [-s SEED] [-v VERBOSE]
-h/--help              Printing help into the screen
-d/--device            OpenCL device (all, cpu, gpu, gpu_nvidia, gpu_intel, gpu_amd, "X;Y;Z"...
                       using all gpu: -d gpu
                       using device index 0 and 2: -d "0;2"
-b/--balance           Balance computation for device if many devices are selected "X;Y;Z"
                       60% computation on device 0 and 40% computatio on device 2: -b "0.6;0.4"
-n/--nparticles        Number of particles (default: 1000000)
-s/--seed              Seed of pseudo generator number (default: 777)
-v/--verbose           Setting level of verbosity
```

Load a water box phantom:

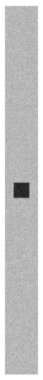
```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('data/phantom.mhd', 'data/range_phantom.txt')
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')
```

Create a CT curved detector:

```
ct_detector = GGEMSCTSystem('Stellar')
ct_detector.set_ct_type('curved')
ct_detector.set_number_of_modules(1, 46)
ct_detector.set_number_of_detection_elements(64, 16, 1)
ct_detector.set_size_of_detection_elements(0.6, 0.6, 0.6, 'mm')
ct_detector.set_material('GOS')
ct_detector.set_source_detector_distance(1085.6, 'mm')
ct_detector.set_source_isocenter_distance(595.0, 'mm')
ct_detector.set_rotation(0.0, 0.0, 0.0, 'deg')
ct_detector.set_threshold(10.0, 'keV')
ct_detector.save('data/projection')
ct_detector.store_scatter(True)
```

Create a cone-beam X-ray source:

```
point_source = GGEMSXRaySource('point_source')
point_source.set_source_particle_type('gamma')
point_source.set_number_of_particles(1000000000)
point_source.set_position(-595.0, 0.0, 0.0, 'mm')
point_source.set_rotation(0.0, 0.0, 0.0, 'deg')
point_source.set_beam_aperture(12.5, 'deg')
point_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
point_source.set_polyenergy('data/spectrum_120kVp_2mmAl.dat')
```



Performance on Windows 11 system and Visual C++ 2022:

Device	Computation Time [s]
GeForce GTX 1050 Ti	112
Quadro P400	385
Xeon X-2245 8 cores / 16 threads	421
GeForce GTX 1050 Ti (80%) Quadro P400 (20%)	91

Performance on Ubuntu 20.04 and GNU GCC 9.3:

Device	Computation Time [s]
GeForce GTX 1050 Ti	90
Quadro P400	360
Xeon X-2245 8 cores / 16 threads	395
GeForce GTX 1050 Ti (80%) Quadro P400 (20%)	70

Examples 3: Voxelized Phantom Generator

Voxelized phantom can be creating using GGEMS. Only basic shapes are available such as tube, box and sphere. Output format is MHD, and a range material data file is created in same time than a voxelized volume.

```
$ python generate_volume.py [-h] [-d DEVICE] [-v VERBOSE]
-h/--help            Printing help into the screen
-d/--device          Setting OpenCL id
-v/--verbose         Setting level of verbosity
```

Create a global volume storing all voxelized objets:

```
volume_creator_manager = GGEMSVolumeCreatorManager()

volume_creator_manager.set_dimensions(450, 450, 450)
volume_creator_manager.set_element_sizes(0.5, 0.5, 0.5, "mm")
volume_creator_manager.set_output('data/volume')
volume_creator_manager.set_range_output('data/range_volume')
volume_creator_manager.set_material('Air')
volume_creator_manager.set_data_type('MET_INT')
volume_creator_manager.initialize()
```

Draw an object in previous global volume:

```
# Creating a box
box = GGEMSBox(24.0, 36.0, 56.0, 'mm')
box.set_position(-70.0, -30.0, 10.0, 'mm')
box.set_label_value(1)
box.set_material('Water')
box.initialize()
box.draw()
box.delete()

# Creating a tube
tube = GGEMSTube(13.0, 8.0, 50.0, 'mm')
tube.set_position(20.0, 10.0, -2.0, 'mm')
tube.set_label_value(2)
tube.set_material('Calcium')
tube.initialize()
tube.draw()
tube.delete()

# Creating a sphere
sphere = GGEMSSphere(14.0, 'mm')
sphere.set_position(30.0, -30.0, 8.0, 'mm')
sphere.set_label_value(3)
sphere.set_material('Lung')
sphere.initialize()
sphere.draw()
sphere.delete()
```

Examples 4: Dosimetry

A cylinder is simulated computing absorbed dose inside it. Different results such as dose, energy deposited… are registered in MHD files. An external source, using GGEMS X-ray source is simulated generating 2e8 particles and TLE is activated to improve statistics.

```
$ python dosimetry_photon.py [-h] [-d DEVICE] [-b BALANCE] [-n N_PARTICLES] [-s SEED] [-v VE
-h/--help            Printing help into the screen
-d/--device          OpenCL device (all, cpu, gpu, gpu_nvidia, gpu_intel, gpu_amd, "X;Y;Z"...
                     using all gpu: -d gpu
                     using device index 0 and 2: -d "0;2"
-b/--balance         Balance computation for device if many devices are selected "X;Y;Z"
                     60% computation on device 0 and 40% computatio on device 2: -b "0.6;0.4"
-n/--nparticles      Number of particles (default: 1000000)
-t/--tle             Activating TLE method
-s/--seed            Seed of pseudo generator number (default: 777)
-v/--verbose         Setting level of verbosity
```

Cylinder phantom is loaded:

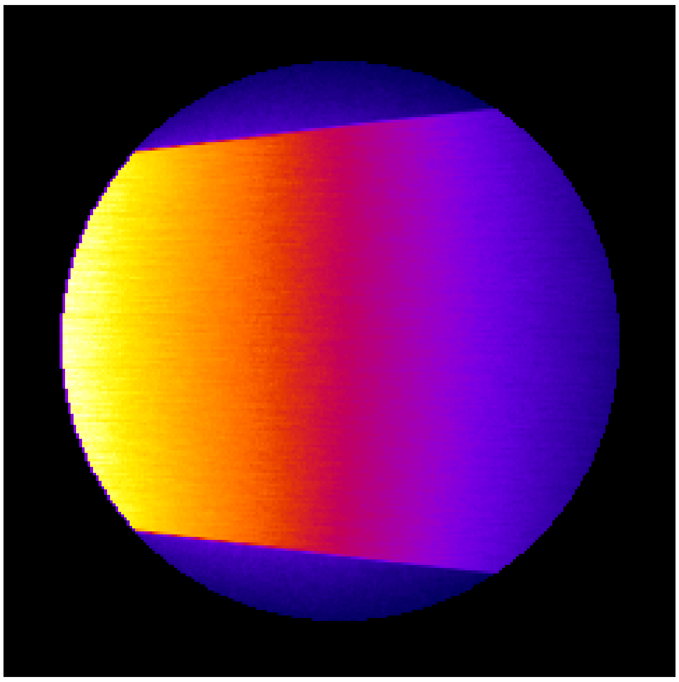
```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('data/phantom.mhd', 'data/range_phantom.txt')
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')
```

Dosimetry associated to the previous phantom:

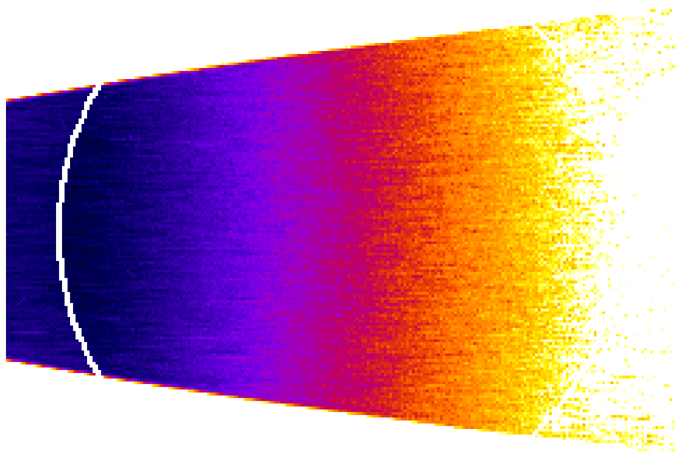
```
dosimetry = GGEMSDosimetryCalculator('phantom')
dosimetry.set_output('data/dosimetry')
dosimetry.set_dosel_size(0.5, 0.5, 0.5, 'mm')
dosimetry.water_reference(False)
dosimetry.minimum_density(0.1, 'g/cm3')
dosimetry.set_tle(is_tle)

dosimetry.uncertainty(True)
dosimetry.photon_tracking(True)
dosimetry.edep(True)
dosimetry.hit(True)
dosimetry.edep_squared(True)
```

External source using GGEMSXRaySource:

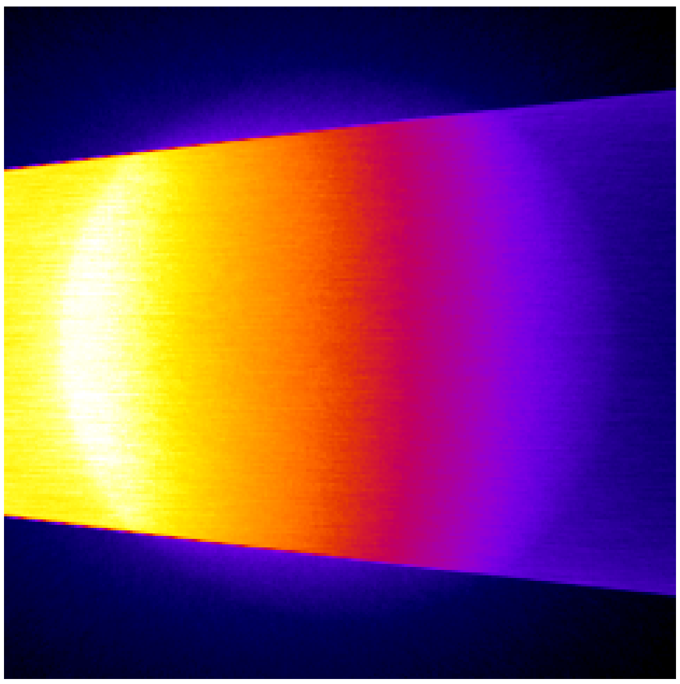
```
point_source = GGEMSXRaySource('point_source')
point_source.set_source_particle_type('gamma')
point_source.set_number_of_particles(200000000)
point_source.set_position(-595.0, 0.0, 0.0, 'mm')
point_source.set_rotation(0.0, 0.0, 0.0, 'deg')
point_source.set_beam_aperture(5.0, 'deg')
point_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
point_source.set_polyenergy('data/spectrum_120kVp_2mmAl.dat')
```



Dose absorbed by cylinder phantom



Uncertainty dose computation



Photon tracking in phantom

Performance on Windows 11 system and Visual C++ 2022:

Device	Computation Time [s]
GeForce GTX 1050 Ti	797
Quadro P400	2100
Xeon X-2245 8 cores / 16 threads	1027
GeForce GTX 1050 Ti (58%) Xeon X-2245 8 cores / 16 threads (42%)	512

Performance on Ubuntu 20.04 and GNU GCC 9.3:

Device	Computation Time [s]
GeForce GTX 1050 Ti	860
Quadro P400	2190
Xeon X-2245 8 cores / 16 threads	1010
GeForce GTX 1050 Ti (55%) Xeon X-2245 8 cores / 16 threads (45%)	510

Examples 5: World Tracking

A cylinder is simulated computing absorbed dose inside it, a CBCT flat panel detector is also defined storing photon counts. A GGEMSWorld volume is created in order to store the fluence outside cylinder phantom and detector. An external source, using GGEMS X-ray source is simulated generating 1e8 particles.

```
$ python world_tracking.py [-h] [-d DEVICE] [-b BALANCE] [-n N_PARTICLES] [-s SEED] [-v VERB
-h/--help              Printing help into the screen
-d/--device            OpenCL device (all, cpu, gpu, gpu_nvidia, gpu_intel, gpu_amd, "X;Y;Z"...
                       using all gpu: -d gpu
                       using device index 0 and 2: -d "0;2"
-b/--balance           Balance computation for device if many devices are selected "X;Y;Z"
                       60% computation on device 0 and 40% computatio on device 2: -b "0.6;0.4"
-n/--nparticles        Number of particles (default: 1000000)
-s/--seed              Seed of pseudo generator number (default: 777)
-v/--verbose           Setting level of verbosity
```

World definition:

```
world = GGEMSWorld()
world.set_dimensions(200, 200, 200)
world.set_element_sizes(10.0, 10.0, 10.0, 'mm')
world.set_output_basename('data/world')

world.energy_tracking(True)
world.energy_squared_tracking(True)
world.momentum(True)
world.photon_tracking(True)
```

Cylinder phantom is loaded and dosimetry module is associated to cylinder phantom, and all output are activated

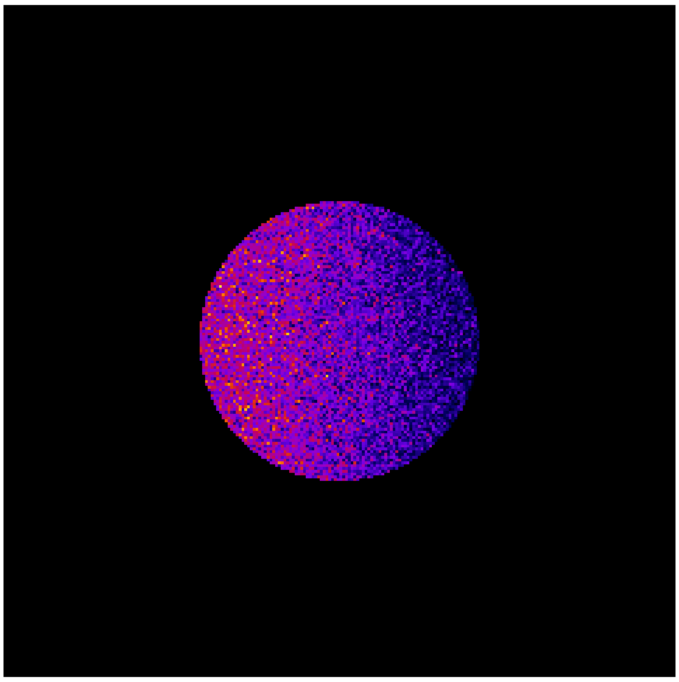
```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('data/phantom.mhd', 'data/range_phantom.txt')
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')

dosimetry = GGEMSDosimetryCalculator()
dosimetry.attach_to_navigator('phantom')
dosimetry.set_output_basename('data/dosimetry')
dosimetry.set_dosel_size(1.0, 1.0, 1.0, 'mm')
dosimetry.water_reference(False)
dosimetry.minimum_density(0.1, 'g/cm3')

dosimetry.uncertainty(True)
dosimetry.photon_tracking(True)
dosimetry.edep(True)
dosimetry.hit(True)
dosimetry.edep_squared(True)
```

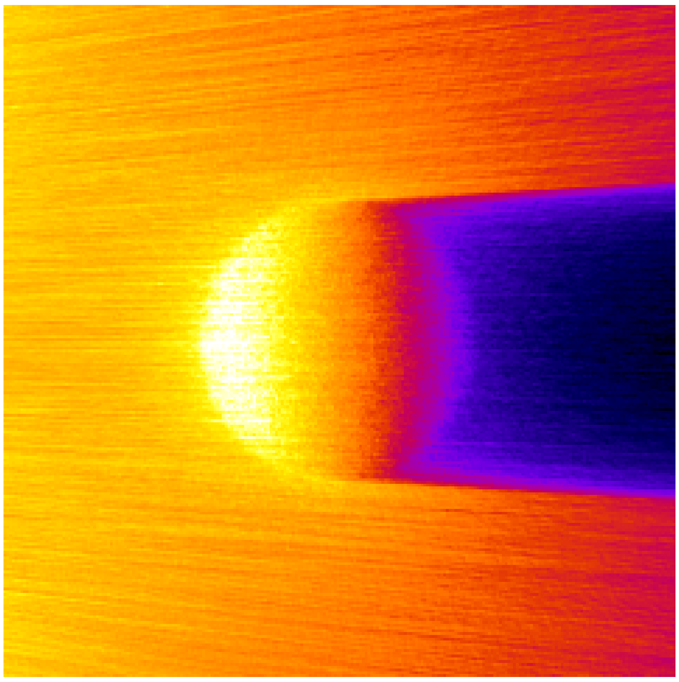
A CBCT flat panel detector definition:

```
cbct_detector = GGEMSCTSystem('custom')
cbct_detector.set_ct_type('flat')
cbct_detector.set_number_of_modules(1, 1)
cbct_detector.set_number_of_detection_elements(400, 400, 1)
cbct_detector.set_size_of_detection_elements(1.0, 1.0, 10.0, 'mm')
cbct_detector.set_material('Silicon')
cbct_detector.set_source_detector_distance(1500.0, 'mm')
cbct_detector.set_source_isocenter_distance(900.0, 'mm')
cbct_detector.set_rotation(0.0, 0.0, 0.0, 'deg')
cbct_detector.set_threshold(10.0, 'keV')
cbct_detector.save('data/projection.mhd')
```

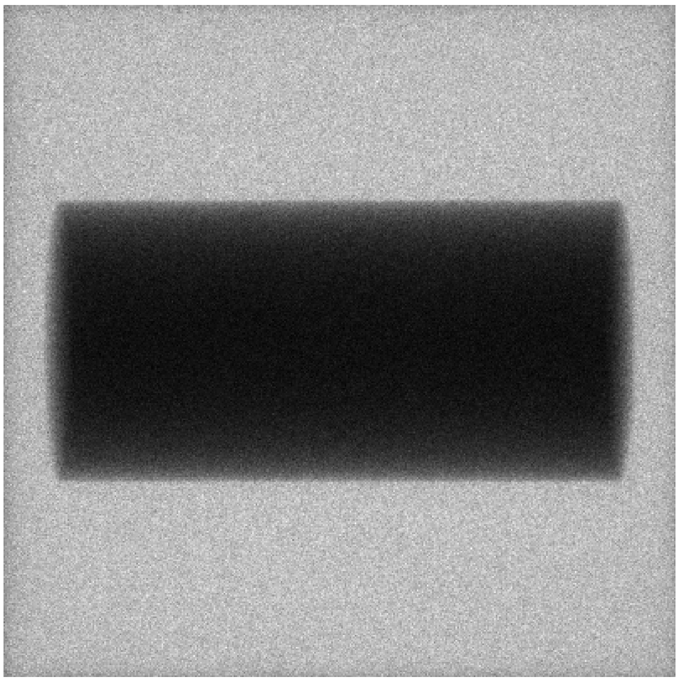
External source using GGEMSXRaySource:

```
point_source = GGEMSXRaySource('point_source')
point_source.set_source_particle_type('gamma')
point_source.set_number_of_particles(100000000)
point_source.set_position(-900.0, 0.0, 0.0, 'mm')
point_source.set_rotation(0.0, 0.0, 0.0, 'deg')
point_source.set_beam_aperture(12.0, 'deg')
point_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
point_source.set_monoenergy(60.0, 'keV')
```

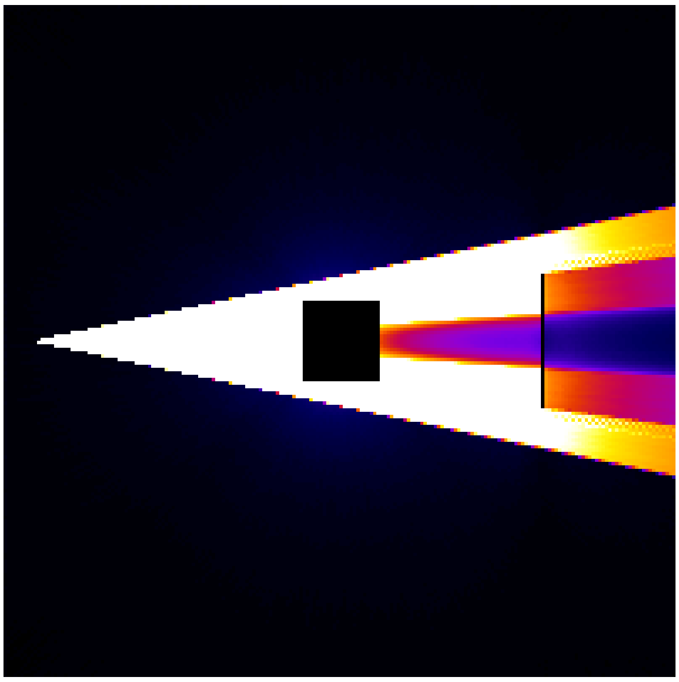

Dose absorbed by cylinder phantom



Photon tracking in phantom



Cylinder projection on flat panel detector



World photon tracking

Performance on Windows 11 system and Visual C++ 2022:

Device	Computation Time [s]
GeForce GTX 1050 Ti	353
Quadro P400	920
Xeon X-2245 8 cores / 16 threads	346
GeForce GTX 1050 Ti (50%) Quadro P400 (50%)	198

Performance on Ubuntu 20.04 and GNU GCC 9.3:

Device	Computation Time [s]
GeForce GTX 1050 Ti	400
Quadro P400	960
Xeon X-2245 8 cores / 16 threads	360
GeForce GTX 1050 Ti (40%) Quadro P400 (60%)	240

Examples 6: OpenGL Visualization

Since GGEMS v1.2, the OpenGL library can be activated.

```
python visualization.py [-h] [-v VERBOSE] [-e] [-w WDIMS WDIMS] [-m MSAA] [-a] [-p NPARTICLE
-h/--help              show this help message and exit
-v/--verbose           Set level of verbosity
-e/--nogl              Disable OpenGL
-w/--wdims             OpenGL window dimensions (default: [800, 800])
                       for a 400x400 window: -w 400 400
-m/--msaa              MSAA factor (1x, 2x, 4x or 8x) (default: 8)
-a/--axis              Drawing axis in OpenGL window
-p/--nparticlesgl      Number of displayed primary particles on OpenGL window (max: 65536) (de
-b/--drawgeom          Draw geometry only on OpenGL window (default: False)
-c/--wcolor            Background color of OpenGL window (default: black)
                       using blue color for background: -c "blue"
```

```
-d/--device          OpenCL device running visualization (default: 0)
                     only 1 device available for this example
                     using device index 0: -d 0
-n/--nparticles      Number of particles (default: 1000000)
-s/--seed            Seed of pseudo generator number (default: 777)
```

Create instance of GGEMSOpenGLManager:

```
opengl_manager = GGEMSOpenGLManager()
```

Many parameters can be set for OpenGL:

```
opengl_manager = GGEMSOpenGLManager()
opengl_manager.set_window_dimensions(window_dims[0], window_dims[1])
opengl_manager.set_msaa(msaa)
opengl_manager.set_background_color(window_color)
opengl_manager.set_draw_axis(is_axis)
opengl_manager.set_world_size(3.0, 3.0, 3.0, 'm')
opengl_manager.set_image_output('data/axis')
opengl_manager.set_displayed_particles(number_of_displayed_particles)
opengl_manager.set_particle_color('gamma', 152, 251, 152)
# opengl_manager.set_particle_color('gamma', color_name='red') # Using registered color
opengl_manager.initialize()
```

Display only geometry and system:

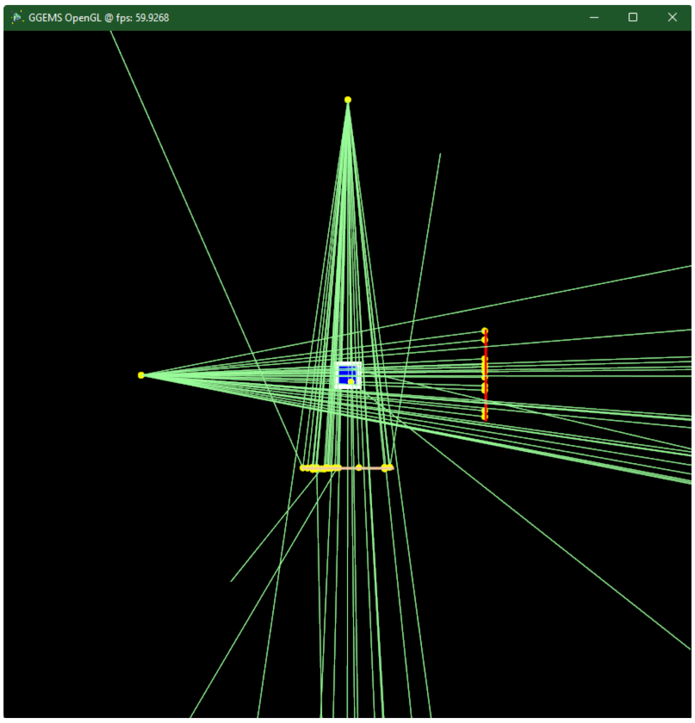
```
opengl_manager.display()
```

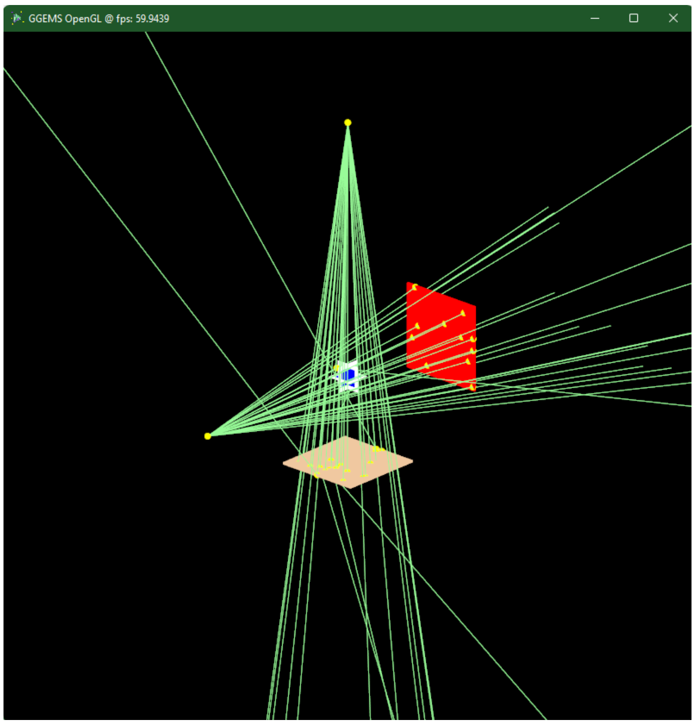
Using GGEMS for a complete visualization (geometry, system and particles):

```
ggems.run()
```

The OpenGL window is interactive, the scene can be moved using keyboard or mouse:

```
Keys:
    * [Esc/X]               Quit application
    * [C]                   Wireframe view
    * [V]                   Solid view
    * [R]                   Reset view
    * [K]                   Save current window to a PNG file
    * [+/-]                 Zoom in/out
    * [Up/Down]             Move forward/back
    * [W/S]
    * [Left/Right]          Move left/right
    * [A/D]

Mouse:
    * [Scroll Up/Down]      Zoom in/out
    * [Left button]         Rotation
    * [Middle button]       Translation
```





Multi-Devices

The GGEMS library can use several devices during the simulation. There are two ways to do this, either give the device index or explicitly indicate the device(s).

- Using the device index:

```python
from ggems import *

opencl_manager = GGEMSOpenCLManager()
```

```
opencl_manager.set_device_index(0) # Activate device id 0
opencl_manager.set_device_index(2) # Activate device id 2, if it exists

opencl_manager.clean()
exit()
```

- Explicitly indicate the device(s):

```
from ggems import *

opencl_manager = GGEMSOpenCLManager()
opencl_manager.set_device_to_activate('gpu', 'nvidia') # Activate all NVIDIA GPU only
opencl_manager.set_device_to_activate('gpu', 'intel') # Activate all Intel GPU only
opencl_manager.set_device_to_activate('gpu', 'amd') # Activate all AMD GPU only
opencl_manager.set_device_to_activate('cpu') # Activate cpu only
opencl_manager.set_device_to_activate('all') # Activate all found devices
opencl_manager.set_device_to_activate('0;2') # Activate devices 0 and 2

opencl_manager.clean()
exit()
```

OpenGL Visualization

Since GGEMS v1.2, it is possible to enable graphical visualization using the OpenGL library. In GGEMS several types of volumes can be drawn like spheres, cones, cylinders and voxelized volumes.

To be able to use OpenGL you must first call the singleton and then initialize some parameters.

Regarding the colors in GGEMS, it is possible to use defined colors such as:

- black
- blue
- lime
- cyan
- red
- magenta
- yellow
- white
- gray
- silver
- maroon
- olive
- green
- purple
- teal
- navy

Important

It is important to initialize the OpenGL settings at the start of the simulation. All volumes, and all sources must be declared after this initialization

```
from ggems import *

opengl_manager = GGEMSOpenGLManager()

# Multisample anti-aliasing, can be 1, 2, 4 or 8
opengl_manager.set_msaa(8)

# Background color for OpenGL window
opengl_manager.set_background_color('black')

# Draw axis X, Y and Z
opengl_manager.set_draw_axis(True)

# Output folder storing OpenGL images you want to save (*.png format)
opengl_manager.set_image_output('axis')

# Number of maximum particles drawn in the OpenGL viewport (max: 65536)
opengl_manager.set_displayed_particles(128)

# Change the color of photons, either using a color defined in GGEMS or using RGB indices
opengl_manager.set_particle_color('gamma', 152, 251, 152)
# opengl_manager.set_particle_color('gamma', color_name='red')

# Set the size of your world surrounding your simulation. It is important that this is large
opengl_manager.set_world_size(3.0, 3.0, 3.0, 'm')

# Size of OpenGL viewport
opengl_manager.set_window_dimensions(500, 500)

# Initialize OpenGL
opengl_manager.initialize()

# Show OpenGL window outside GGEMS simulation
opengl_manager.display()
```

If you have defined a navigator (phantom or detector) it is also possible to change the color of the material either by using a defined color or by using RGB indices. It is also possible to disable the color.

```
# We suppose you defined a phantom before composed by air and water
phantom.set_material_visible('Air', True) # or False if you do not want to draw the air voxe
phantom.set_material_color('Water', color_name='blue')

# We suppose you define a CBCT before composed by GOS
cbct_detector.set_material_color('GOS', 255, 0, 0) # Custom color using RGB
#cbct_detector.set_material_color('GOS', color_name='red') # Using registered color
```

Navigators

Particles are only tracked in browsers. In GGEMS there are two types of navigators:

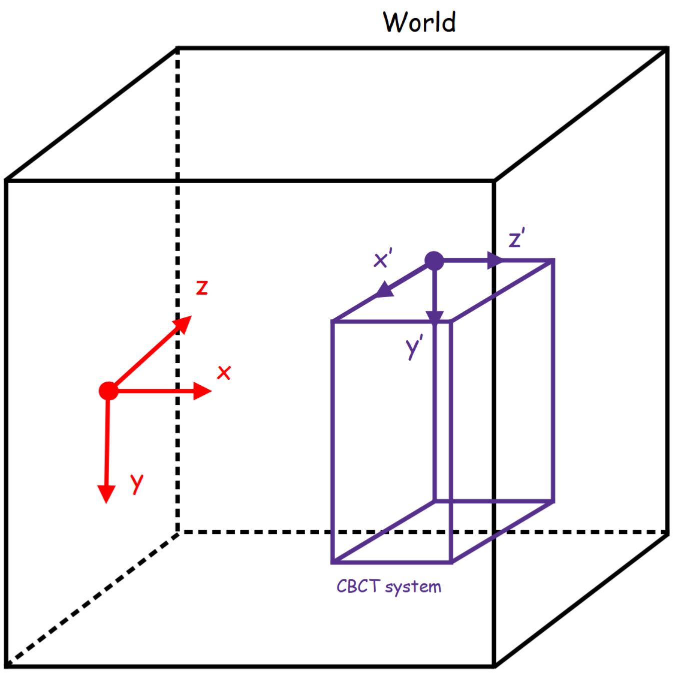
- System (detector)
- Phantom (object or patient)

For each navigator, three types of elements are associated:

- a solid: geometry of the navigator
- a list of materials
- physical processes

Systems

At the moment in GGEMS only CT and CBCT systems are available. The detector is made up of pixels assembled in a module. The figure below shows the reference axes of the world (in red) as well as a CT/CBCT system with its axes (in purple).



A CT/CBCT system is created using the following line:

```
cbct_system = GGEMSCTSystem('detector') # detector is a custom name for your system
```

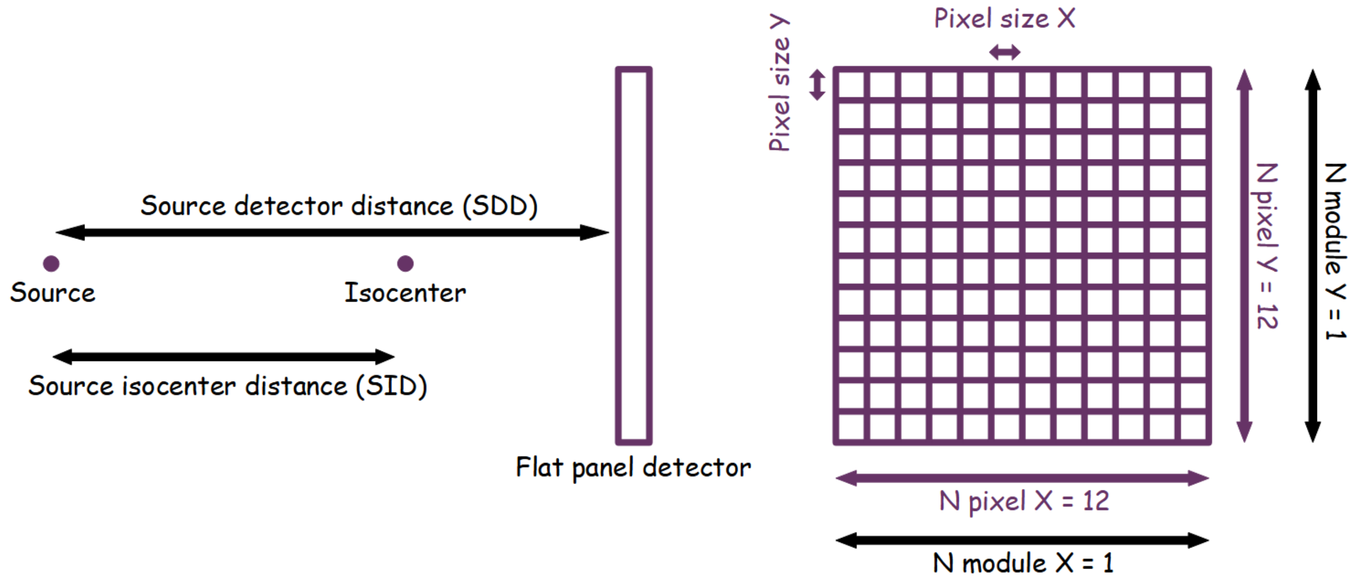
Types of CT/CBCT detector are:

- flat panel
- curved

Flat panel

This type of geometry is designed mainly for CBCT systems.

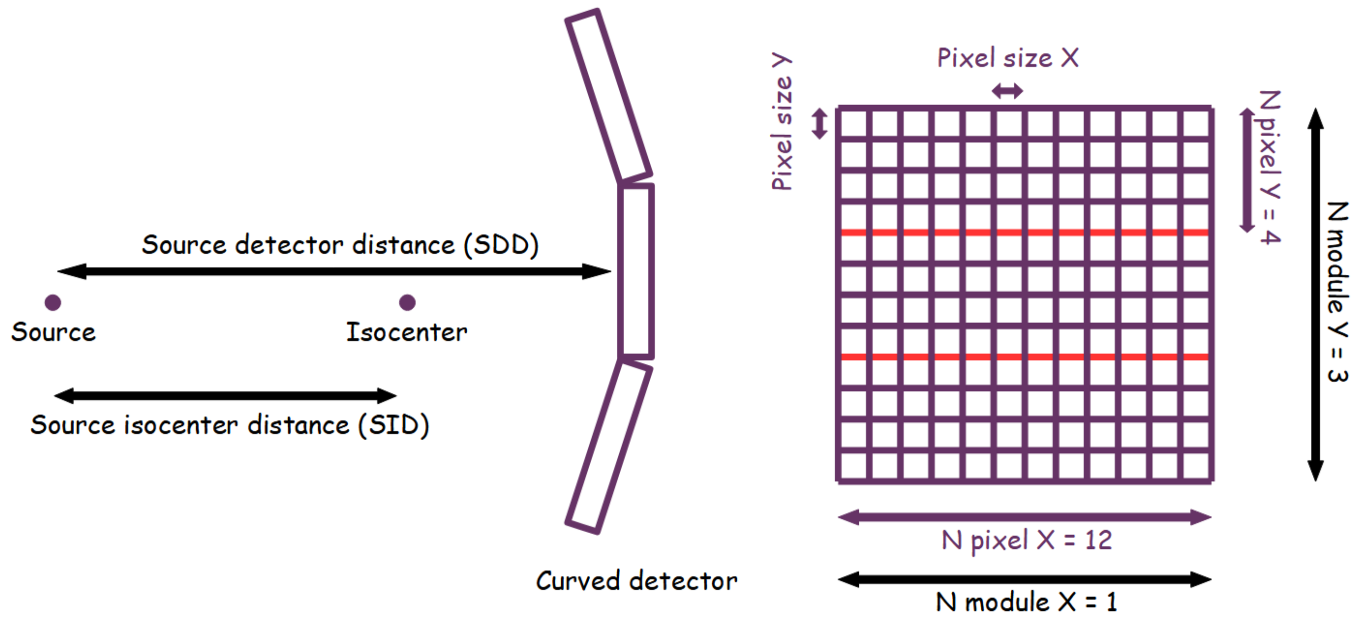
```
cbct_system.set_ct_type('flat')
```



Curved

This type of geometry is well suited for CT systems.

```
cbct_system.set_ct_type('curved')
```



Note

For curved geometry, the angle between modules is automatically calculated. There is no gap between the modules. The center of rotation of the system is the center of the world.

For each type of detector, the number of modules, the number of detection elements inside the module and their respective sizes are set as following:

```
cbct_system.set_number_of_modules(1, 3)
cbct_system.set_number_of_detection_elements(12, 4, 1)
cbct_system.set_size_of_detection_elements(1.0, 1.0, 1.0, 'mm')
```

A detector can be composed by only one type of material:

```
cbct_system.set_material('GOS')
```

An energy detection threshold can also be specified:

```
cbct_system.set_threshold(10.0, 'keV')
```

Source isocenter distance (SID) and source detector distance (SDD) is set with the following commands:

```
# Do not forget to add half size of detection element !!!
cbct_system.set_source_detector_distance(1500.5, 'mm')
cbct_system.set_source_isocenter_distance(900.0, 'mm')
```

Note

The position of the detector is calculated according to the values of the SID and SDD

A CT/CBCT system can be rotated around the world axis with the following command:

```
# 40 degree rotation around Z world axis
cbct_system.set_rotation(0.0, 0.0, 40.0, 'deg')
```

A CT/CBCT system can be translated along the world axis with the following command:

```
# 400 mm translation along Z world axis
cbct_system.set_global_system_position(0.0, 0.0, 400.0, 'mm');
```

The final projection including all photon interactions is saved in a file in MHD format, and scattered photons can be also save in another file, and we also give users the possibility to save the photons diffused for their needs:

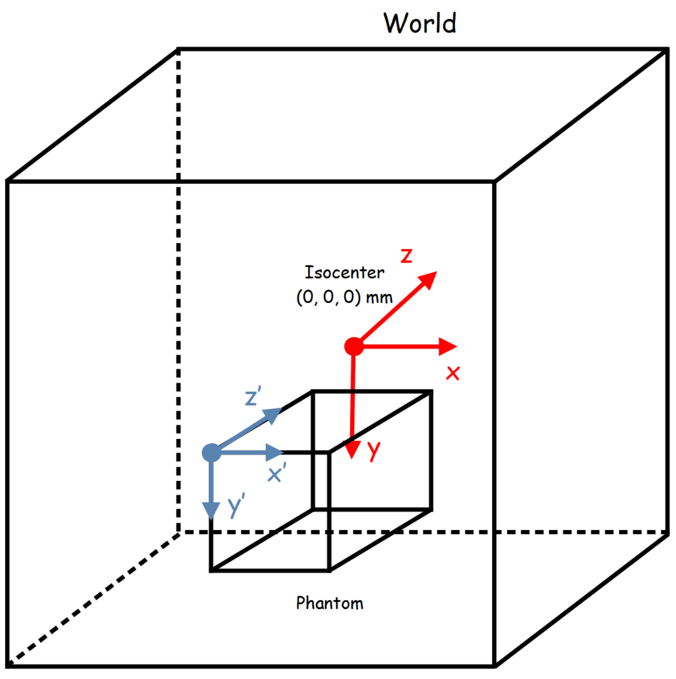
```
cbct_system.save('projection')
cbct_system.store_scatter(True)
```

To enable detector visualization and change the default color:

```
cbct_detector.set_visible(True)
cbct_detector.set_material_color('GOS', 255, 0, 0) # Custom color using RGB
cbct_detector.set_material_color('GOS', color_name='red') # Or using registered color
```

Phantoms

For the moment in GGEMS only voxelized volumes are available to define a phantom. The phantom must be in an MHD format associated with a TXT file storing the labels for the materials. The axes of the phantom correspond to the axes of the world.



To create a phantom you have to give it a name and load a file containing the phantom information and another file containing the material label:

```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('phantom.mhd', 'range_phantom.txt')
```

The phantom can be positioned and rotated in space using the following commands:

```
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')
```

If OpenGL is enabled the phantom can be viewed or disabled. The color of the materials composing the phantom can be modified using the RGB indices or the default colors. Regarding the air it is rather advisable not to display it in order to save memory:

```
phantom.set_visible(True)
phantom.set_material_visible('Air', False)
phantom.set_material_color('Water', color_name='blue')
```

World

Outside the navigator, particles are not tracked. However, a tool has been developed in GGEMS to record the photons leaving the navigator. Particles are projected into the world using a DDA algorithm.

The world module is ‘GGEMSWorld’:

```
world = GGEMSWorld()
```

After creating a GGEMSWorld, the dimension of the world and size of voxel can be set:

```
world.set_dimensions(200, 200, 200)
world.set_element_sizes(10.0, 10.0, 10.0, 'mm')
```

For world output, there are many informations that the user can save such as: energy and energy squared of photon crossing voxel, photon momentum and fluence (photon tracking):

```
world.set_output_basename('data/world')
world.energy_tracking(True)
world.energy_squared_tracking(True)
world.momentum(True)
world.photon_tracking(True)
```

Dosimetry

Dosimetry module can be activated to compute absorbed dose. For the moment only photons are simulated, electrons are ignored.

Dosimetry module is ‘GGEMSDosimetryCalculator’:

```
dosimetry = GGEMSDosimetryCalculator()
```

A navigator phantom must be attached to GGEMSDosimetryCalculator using:

```
dosimetry.attach_to_navigator('phantom')
```

Size of voxel (dosel) for dosimetry image could be set, if not the dosel size is the same than voxel phantom size:

```
dosimetry.set_dosel_size(0.5, 0.5, 0.5, 'mm')
```

Absorbed dose is computed in gray (Gy). By default dose in computed using materials in phantom. Otherwize the user can set water material everywhere in phantom.

```
dosimetry.water_reference(True)
```

A custom density threshold can be set. If density of phantom is below this threshold the dose value in 0 Gy.

```
dosimetry.minimum_density(0.1, 'g/cm3')
```

Since GGEMS v1.2, TLE method (Track Length Estimated) proposed in [Smekens2014] can be activated to improve statistics.

```
dosimetry.set_tle(is_tle)
```

Many informations can be registered such as: uncertainty values of the dose, the deposited energy in dosel, the squared of deposited energy in dosel and the number of interactions (hits) in dosel:

```
dosimetry.set_output('dosimetry')
dosimetry.uncertainty(True)
dosimetry.edep(True)
dosimetry.hit(True)
dosimetry.edep_squared(True)
```

There is a special output named ‘photon tracking’. This output registers the number of photons crossing a dosel. To use this option, the size of dosel has to be the same than the phantom voxel size, otherwize GGEMS will throw an error:

```
dosimetry.photon_tracking(True)
```

- F. Baldacci, A. Mittone, A. Bravin, P. Coan, F. Delaire, C. Ferrero, S. Gasilov, J. M. Létang, D. Sarrut, F. Smekens, et al., “A track length estimator method for dose calculations in low-energy x-ray irradiations: implementation, properties and performance”, Zeitschrift Fur Medizinische Physik, 2014.

Physical Processes & Range cuts

Physical Processes

The photon processes impletemented are:

- Compton scattering
- Photoelectric effect
- Rayleigh scattering

Each of these processes are extracted from Geant4 version 10.6. For more informations about physics, please read the documentation on the Geant4 website.

By using python, the singleton can be called with the following command:

```
processes_manager = GGEMSProcessesManager()
```

Important

Secondary particles (photon and electron) are not simulated yet. For Photoelectric effect, the photon is killed during the interaction and the energy is locally deposited, the fluorescence photon is not emitted.

Compton Scattering

The Geant4 model extracted is the ‘G4KleinNishinaCompton’ standard model. It is the fastest algorithm to simulate this process. Compton scattering is activated for all the navigators, or for a specific navigator.

```
# Activating Compton process for all navigators
processes_manager.add_process('Compton', 'gamma', 'all')

# Activating Compton process for a specific navigator name 'my_phantom'
processes_manager.add_process('Compton', 'gamma', 'my_phantom')
```

Photoelectric Effect

The Geant4 model extracted is the ‘G4PhotoElectricEffect’ standard model using Sandia tables. Photoelectric effect is activated for all the navigators, or for a specific navigator.

```
# Activating Photoelectric process for all navigators
processes_manager.add_process('Photoelectric', 'gamma', 'all')

# Activating Photoelectric process for a specific navigator name 'my_phantom'
processes_manager.add_process('Photoelectric', 'gamma', 'my_phantom')
```

Rayleigh Scattering

The Geant4 model extracted is the ‘G4LivermoreRayleighModel’ livermore model. Rayleigh scattering is activated for all the navigators, or for a specific navigator.

```
# Activating Rayleigh process for all navigators
processes_manager.add_process('Rayleigh', 'gamma', 'all')

# Activating Rayleigh process for a specific navigator name 'my_phantom'
processes_manager.add_process('Rayleigh', 'gamma', 'my_phantom')
```

Process Parameters Building

Cross sections are calculated during GGEMS initialization. The parameters for calculating the cross sections can be modified, but it is recommended to use the default ones. The parameters that can be modified are:

- Minimum energy of cross-section table
- Maximum energy of cross-section table
- Number of bins in cross-section table

Default parameters are defined as following:

```
processes_manager.set_cross_section_table_number_of_bins(220)
processes_manager.set_cross_section_table_energy_min(1.0, 'keV')
processes_manager.set_cross_section_table_energy_max(1.0, 'MeV')
```

Process Verbosity

Some informations about processes can be printed:

- Available processes
- Global informations about processes
- Cross-section values

The list of commands are:

```
processes_manager.print_available_processes()
processes_manager.print_infos()
processes_manager.print_tables(True)
```

Range Cuts

Cuts are defined for each particle in distance unit in a navigator or a specific navigator. During initialization cuts are converted in energy for each material in navigator. If the particle energy is below the cut, then this one is killed and the energy is locally deposited. By default cuts are 1 micron.

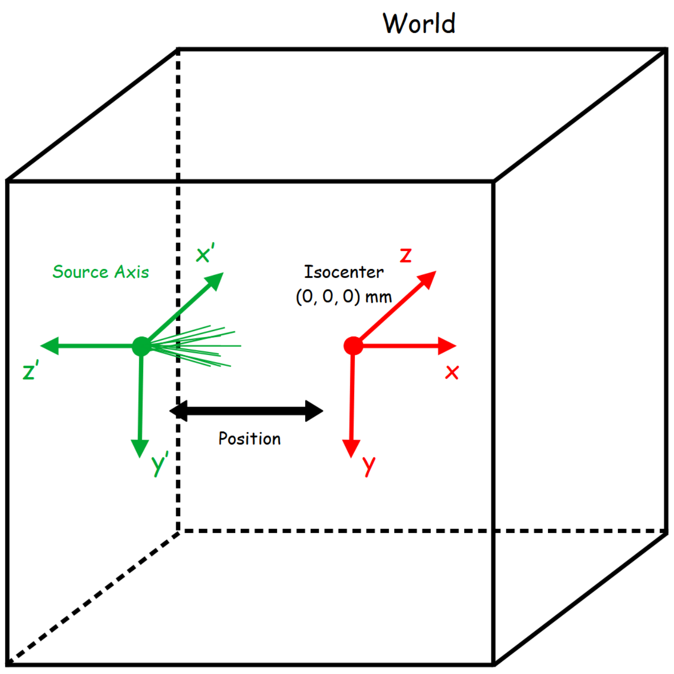
```
# For photon and all navigators
range_cuts_manager.set_cut('gamma', 0.1, 'mm', 'all')

# For photon and a specific navigator named 'my_phantom'
range_cuts_manager.set_cut('gamma', 0.1, 'mm', 'my_phantom')
```

Sources

In GGEMS, we develop a source for each application. For the moment only dosimetry and CT/CBCT applications are allowed. Only one source has been implemented: GGEMSXRaySource

X-ray Source

The X-Ray source is defined with a cone-beam geometry. The direction of the generated photons always point to the center of the world. The source has its own axes.



Create a source by choosing a name:

```
xray_source = GGEMSXRaySource('xray_source')
```

Particle type is only photon:

```
xray_source.set_source_particle_type('gamma')
```

Number of generated particles during a run:

```
xray_source.set_number_of_particles(1000000000)
```

Position and rotation of source are defined in global world axis. The X-Ray cone-beam source can be defined with an aperture angle.

```
xray_source.set_position(-595.0, 0.0, 0.0, 'mm')
xray_source.set_rotation(0.0, 0.0, 0.0, 'deg')
xray_source.set_beam_aperture(12.5, 'deg')
```

X-ray source can be defined with a focal spot size. If defined at (0, 0, 0) mm, it is similar to a point source.

```
xray_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
```

Important

The focal spot size is defined in source axis reference and not in global world reference.

The energy source can be defined using a single energy value or a spectrum included in a TXT file.

```
# Using a custom source
xray_source.set_polyenergy('data/spectrum_120kVp_2mmAl.dat')

# Monoenergetic source
xray_source.set_monoenergy(25.0, 'keV')
```

GGEMS Commands

Commands to manage GGEMS are listed here in detail:

```
# Create an GGEMS instance
ggems = GGEMS()
```

Lot of verboses are available using GGEMS:

```
# Infos about OpenCL environment
ggems.opencl_verbose(True)

# Infos about whole material database
ggems.material_database_verbose(True)

# Infos about navigator (system/phantom)
ggems.navigator_verbose(True)

# Infos about source
ggems.source_verbose(True)

# Infos about memory usage
ggems.memory_verbose(True)

# Infos about activated processe(s)
ggems.process_verbose(True)
```

```
# Infos about range cuts
ggems.range_cuts_verbose(True)

# Print infos about first random state and initial seed
ggems.random_verbose(True)

# Infos about elapsed time in kernels
ggems.profiling_verbose(True)

# Infos about a specific photon index (here photon index 12)
ggems.tracking_verbose(True, 12)
```

Important commands to initialize and run a GGEMS application:

```
# Initialization with a specific seed (here 9383) or let GGEMS get a random seed
ggems.initialize(9383)
# ggems.initialize() # random seed

# Running ggems
ggems.run()
```

Examples & Tools

Few examples and tools are provided for GGEMS users.

Note

Examples are compiled and installed when compilation option ‘BUILD_EXAMPLES’ is set to ON. C++ executables are installed in example folders.

Examples 0: Cross-Section Computation

The purpose of this example is to provide a tool computing cross-sections

```
$ python cross_sections.py [-h] [-d DEVICE] [-m MATERIAL] -p [PROCESS]-e [ENERGY] [-v VERBOS
-h/--help           Printing help into the screen
-d/--device         Setting OpenCL id
-m/--material       Setting one of material defined in GGEMS (Water, Air, ...)
-p/--process        Setting photon physical process (Compton, Rayleigh, Photoelectric)
-e/--energy         Setting photon energy in MeV
-v/--verbose        Setting level of verbosity
```

Verbosity level is defined in the range [0;3]. For a silent GGEMS execution, the level is set to 0, otherwise 3 for lot of informations.

```
GGEMSVerbosity(verbosity_level)

# Select a OpenCL device
opencl_manager.set_device_index(device_id)
```

Create a GGEMSMaterial instance then add a material. The initialization step is mandatory and compute all physical tables, and store them on an OpenCL device:

```
materials = GGEMSMaterials()
materials.add_material(material_name)
materials.initialize()
```

Create a GGEMSCrossSection instance and activate a process:

```
cross_sections = GGEMSCrossSections(materials)
cross_sections.add_process(process_name, 'gamma')
cross_sections.initialize()
```

For attenuation informations, create a GGEMSAttenuations:

```
attenuations = GGEMSAttenuations(materials, cross_sections)
attenuations.initialize();
```

Computing cross section value (in cm2.g-1) for a specific energy (in MeV):

```
cross_sections.get_cs(process_name, material_name, energy_MeV, 'MeV')

# Get attenuation value in cm-1
attenuations.get_mu(material_name, energy_MeV, 'MeV')
# Get energy attenuation value
attenuations.get_mu_en(material_name, energy_MeV, 'MeV')
```

Examples 1: Total Attenuation

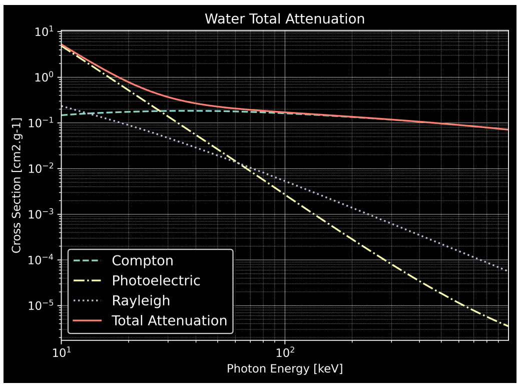
Warning

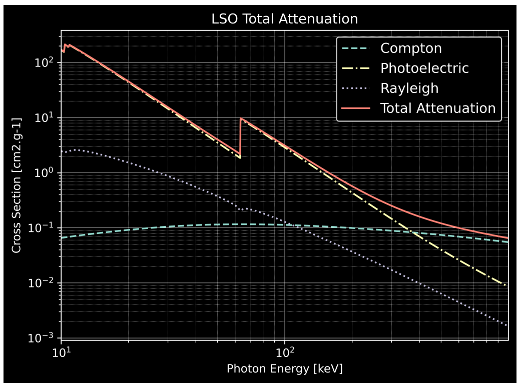
Example only available using python and matplotlib library

Example plotting the total attenuation of a material for energy between 0.01 MeV and 1 MeV. Commands are similar to example 0, and all physical processes are activated.

```
$ python total_attenuation.py [-h] [-d DEVICE] [-m MATERIAL] [-v VERBOSE]
-h/--help           Printing help into the screen
-d/--device         Setting OpenCL id
-m/--material       Setting one of material defined in GGEMS (Water, Air, ...)
-v/--verbose        Setting level of verbosity
```

Total attenuations for Water and LSO are shown below:





Examples 2: CT Scanner

In CT scanner example, a water box is simulated associated to a CT curved detector. One projection is computed simulating 1e9 particles.

```
$ python ct_scanner.py [-h] [-d DEVICE] [-b BALANCE] [-n N_PARTICLES] [-s SEED] [-v VERBOSE]
-h/--help              Printing help into the screen
-d/--device            OpenCL device (all, cpu, gpu, gpu_nvidia, gpu_intel, gpu_amd, "X;Y;Z"...
                       using all gpu: -d gpu
                       using device index 0 and 2: -d "0;2"
-b/--balance           Balance computation for device if many devices are selected "X;Y;Z"
                       60% computation on device 0 and 40% computatio on device 2: -b "0.6;0.4"
-n/--nparticles        Number of particles (default: 1000000)
-s/--seed              Seed of pseudo generator number (default: 777)
-v/--verbose           Setting level of verbosity
```

Load a water box phantom:

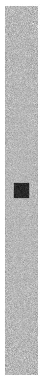
```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('data/phantom.mhd', 'data/range_phantom.txt')
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')
```

Create a CT curved detector:

```
ct_detector = GGEMSCTSystem('Stellar')
ct_detector.set_ct_type('curved')
ct_detector.set_number_of_modules(1, 46)
ct_detector.set_number_of_detection_elements(64, 16, 1)
ct_detector.set_size_of_detection_elements(0.6, 0.6, 0.6, 'mm')
ct_detector.set_material('GOS')
ct_detector.set_source_detector_distance(1085.6, 'mm')
ct_detector.set_source_isocenter_distance(595.0, 'mm')
ct_detector.set_rotation(0.0, 0.0, 0.0, 'deg')
ct_detector.set_threshold(10.0, 'keV')
ct_detector.save('data/projection')
ct_detector.store_scatter(True)
```

Create a cone-beam X-ray source:

```
point_source = GGEMSXRaySource('point_source')
point_source.set_source_particle_type('gamma')
point_source.set_number_of_particles(1000000000)
point_source.set_position(-595.0, 0.0, 0.0, 'mm')
point_source.set_rotation(0.0, 0.0, 0.0, 'deg')
point_source.set_beam_aperture(12.5, 'deg')
point_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
point_source.set_polyenergy('data/spectrum_120kVp_2mmAl.dat')
```



Performance on Windows 11 system and Visual C++ 2022:

Device	Computation Time [s]
GeForce GTX 1050 Ti	112
Quadro P400	385
Xeon X-2245 8 cores / 16 threads	421
GeForce GTX 1050 Ti (80%) Quadro P400 (20%)	91

Performance on Ubuntu 20.04 and GNU GCC 9.3:

Device	Computation Time [s]
GeForce GTX 1050 Ti	90
Quadro P400	360
Xeon X-2245 8 cores / 16 threads	395
GeForce GTX 1050 Ti (80%) Quadro P400 (20%)	70

Examples 3: Voxelized Phantom Generator

Voxelized phantom can be creating using GGEMS. Only basic shapes are available such as tube, box and sphere. Output format is MHD, and a range material data file is created in same time than a voxelized volume.

```
$ python generate_volume.py [-h] [-d DEVICE] [-v VERBOSE]
-h/--help           Printing help into the screen
-d/--device         Setting OpenCL id
-v/--verbose        Setting level of verbosity
```

Create a global volume storing all voxelized objets:

```
volume_creator_manager = GGEMSVolumeCreatorManager()

volume_creator_manager.set_dimensions(450, 450, 450)
volume_creator_manager.set_element_sizes(0.5, 0.5, 0.5, "mm")
volume_creator_manager.set_output('data/volume')
volume_creator_manager.set_range_output('data/range_volume')
volume_creator_manager.set_material('Air')
volume_creator_manager.set_data_type('MET_INT')
volume_creator_manager.initialize()
```

Draw an object in previous global volume:

```
# Creating a box
box = GGEMSBox(24.0, 36.0, 56.0, 'mm')
box.set_position(-70.0, -30.0, 10.0, 'mm')
box.set_label_value(1)
box.set_material('Water')
box.initialize()
box.draw()
box.delete()

# Creating a tube
tube = GGEMSTube(13.0, 8.0, 50.0, 'mm')
tube.set_position(20.0, 10.0, -2.0, 'mm')
tube.set_label_value(2)
tube.set_material('Calcium')
tube.initialize()
tube.draw()
tube.delete()

# Creating a sphere
sphere = GGEMSSphere(14.0, 'mm')
sphere.set_position(30.0, -30.0, 8.0, 'mm')
sphere.set_label_value(3)
sphere.set_material('Lung')
sphere.initialize()
sphere.draw()
sphere.delete()
```

Examples 4: Dosimetry

A cylinder is simulated computing absorbed dose inside it. Different results such as dose, energy deposited… are registered in MHD files. An external source, using GGEMS X-ray source is simulated generating 2e8 particles and TLE is activated to improve statistics.

```
$ python dosimetry_photon.py [-h] [-d DEVICE] [-b BALANCE] [-n N_PARTICLES] [-s SEED] [-v VE
-h/--help             Printing help into the screen
-d/--device           OpenCL device (all, cpu, gpu, gpu_nvidia, gpu_intel, gpu_amd, "X;Y;Z"...
                      using all gpu: -d gpu
                      using device index 0 and 2: -d "0;2"
-b/--balance          Balance computation for device if many devices are selected "X;Y;Z"
                      60% computation on device 0 and 40% computatio on device 2: -b "0.6;0.4"
-n/--nparticles       Number of particles (default: 1000000)
-t/--tle              Activating TLE method
-s/--seed             Seed of pseudo generator number (default: 777)
-v/--verbose          Setting level of verbosity
```

Cylinder phantom is loaded:

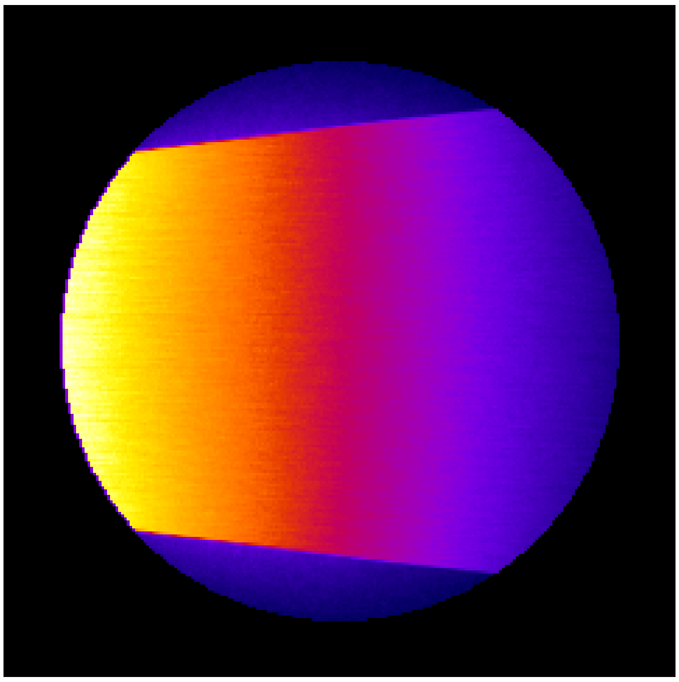
```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('data/phantom.mhd', 'data/range_phantom.txt')
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')
```

Dosimetry associated to the previous phantom:

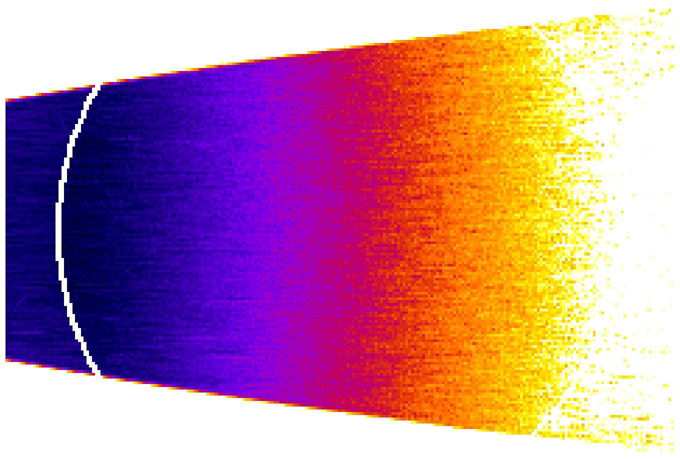
```
dosimetry = GGEMSDosimetryCalculator('phantom')
dosimetry.set_output('data/dosimetry')
dosimetry.set_dosel_size(0.5, 0.5, 0.5, 'mm')
dosimetry.water_reference(False)
dosimetry.minimum_density(0.1, 'g/cm3')
dosimetry.set_tle(is_tle)

dosimetry.uncertainty(True)
dosimetry.photon_tracking(True)
dosimetry.edep(True)
dosimetry.hit(True)
dosimetry.edep_squared(True)
```

External source using GGEMSXRaySource:

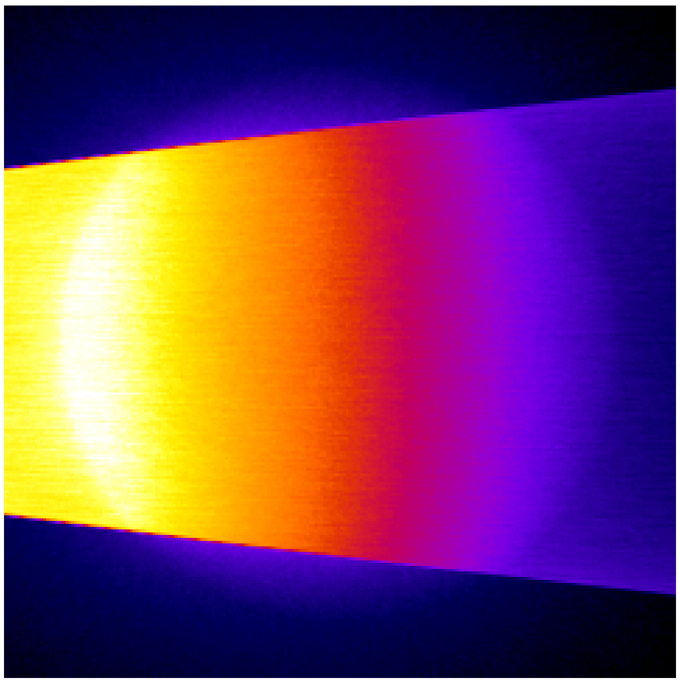
```
point_source = GGEMSXRaySource('point_source')
point_source.set_source_particle_type('gamma')
point_source.set_number_of_particles(200000000)
point_source.set_position(-595.0, 0.0, 0.0, 'mm')
point_source.set_rotation(0.0, 0.0, 0.0, 'deg')
point_source.set_beam_aperture(5.0, 'deg')
point_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
point_source.set_polyenergy('data/spectrum_120kVp_2mmAl.dat')
```



Dose absorbed by cylinder phantom



Uncertainty dose computation



Photon tracking in phantom

Performance on Windows 11 system and Visual C++ 2022:

Device	Computation Time [s]
GeForce GTX 1050 Ti	797
Quadro P400	2100
Xeon X-2245 8 cores / 16 threads	1027
GeForce GTX 1050 Ti (58%) Xeon X-2245 8 cores / 16 threads (42%)	512

Performance on Ubuntu 20.04 and GNU GCC 9.3:

Device	Computation Time [s]
GeForce GTX 1050 Ti	860
Quadro P400	2190
Xeon X-2245 8 cores / 16 threads	1010
GeForce GTX 1050 Ti (55%) Xeon X-2245 8 cores / 16 threads (45%)	510

Examples 5: World Tracking

A cylinder is simulated computing absorbed dose inside it, a CBCT flat panel detector is also defined storing photon counts. A GGEMSWorld volume is created in order to store the fluence outside cylinder phantom and detector. An external source, using GGEMS X-ray source is simulated generating 1e8 particles.

```
$ python world_tracking.py [-h] [-d DEVICE] [-b BALANCE] [-n N_PARTICLES] [-s SEED] [-v VERB
-h/--help             Printing help into the screen
-d/--device           OpenCL device (all, cpu, gpu, gpu_nvidia, gpu_intel, gpu_amd, "X;Y;Z"...
                      using all gpu: -d gpu
                      using device index 0 and 2: -d "0;2"
-b/--balance          Balance computation for device if many devices are selected "X;Y;Z"
                      60% computation on device 0 and 40% computatio on device 2: -b "0.6;0.4"
-n/--nparticles       Number of particles (default: 1000000)
-s/--seed             Seed of pseudo generator number (default: 777)
-v/--verbose          Setting level of verbosity
```

World definition:

```
world = GGEMSWorld()
world.set_dimensions(200, 200, 200)
world.set_element_sizes(10.0, 10.0, 10.0, 'mm')
world.set_output_basename('data/world')

world.energy_tracking(True)
world.energy_squared_tracking(True)
world.momentum(True)
world.photon_tracking(True)
```

Cylinder phantom is loaded and dosimetry module is associated to cylinder phantom, and all output are activated

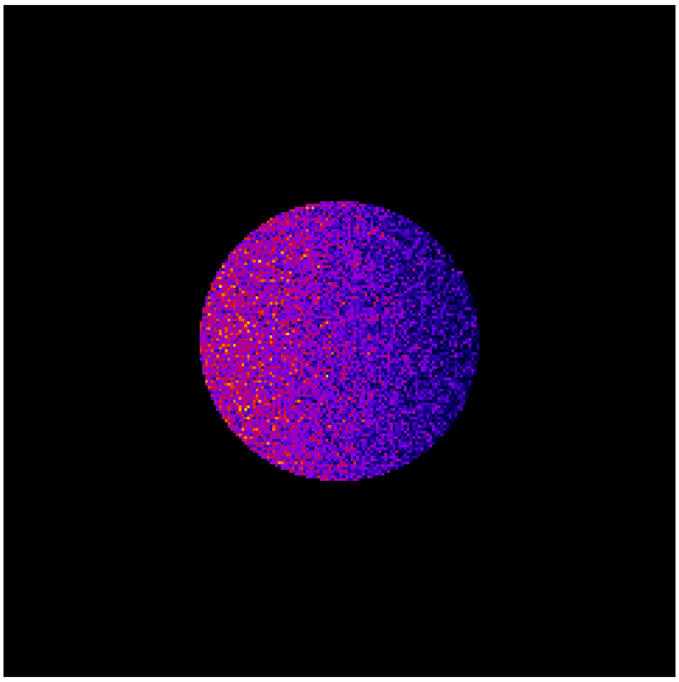
```
phantom = GGEMSVoxelizedPhantom('phantom')
phantom.set_phantom('data/phantom.mhd', 'data/range_phantom.txt')
phantom.set_rotation(0.0, 0.0, 0.0, 'deg')
phantom.set_position(0.0, 0.0, 0.0, 'mm')

dosimetry = GGEMSDosimetryCalculator()
dosimetry.attach_to_navigator('phantom')
dosimetry.set_output_basename('data/dosimetry')
dosimetry.set_dosel_size(1.0, 1.0, 1.0, 'mm')
dosimetry.water_reference(False)
dosimetry.minimum_density(0.1, 'g/cm3')

dosimetry.uncertainty(True)
dosimetry.photon_tracking(True)
dosimetry.edep(True)
dosimetry.hit(True)
dosimetry.edep_squared(True)
```

A CBCT flat panel detector definition:

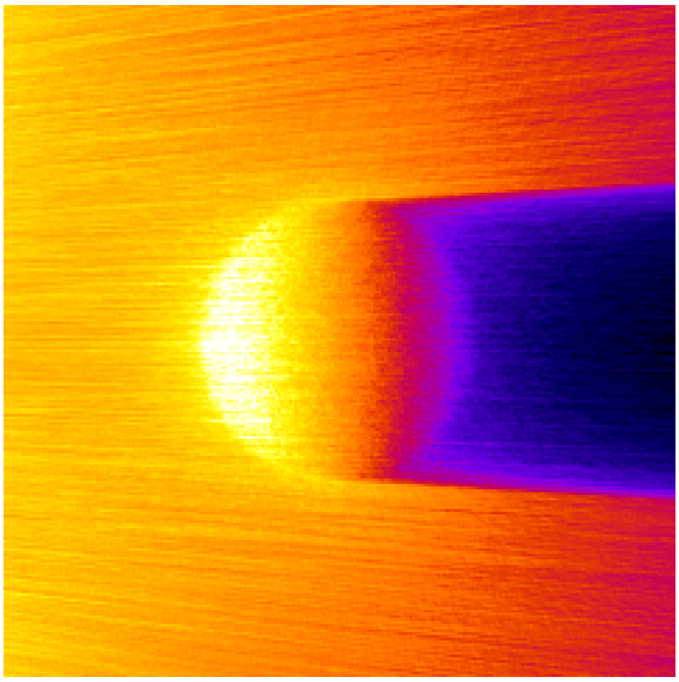
```
cbct_detector = GGEMSCTSystem('custom')
cbct_detector.set_ct_type('flat')
cbct_detector.set_number_of_modules(1, 1)
cbct_detector.set_number_of_detection_elements(400, 400, 1)
cbct_detector.set_size_of_detection_elements(1.0, 1.0, 10.0, 'mm')
cbct_detector.set_material('Silicon')
cbct_detector.set_source_detector_distance(1500.0, 'mm')
cbct_detector.set_source_isocenter_distance(900.0, 'mm')
cbct_detector.set_rotation(0.0, 0.0, 0.0, 'deg')
cbct_detector.set_threshold(10.0, 'keV')
cbct_detector.save('data/projection.mhd')
```

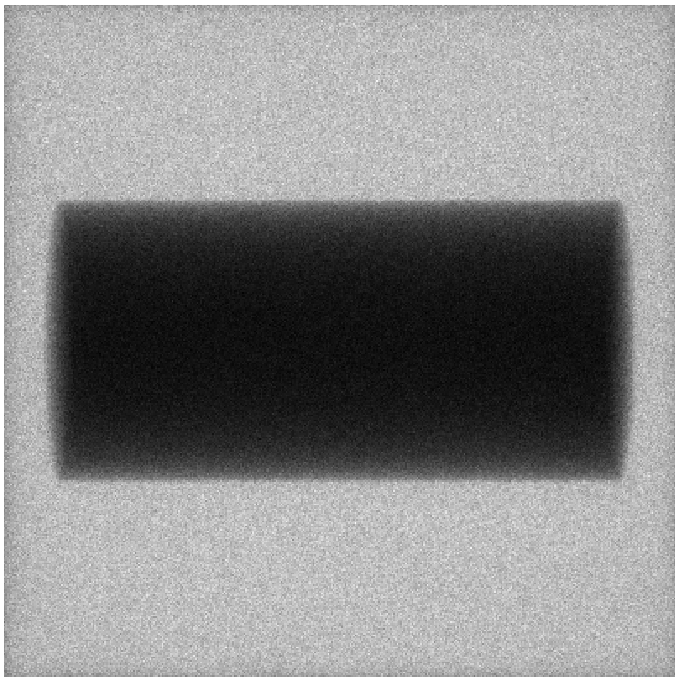
External source using GGEMSXRaySource:

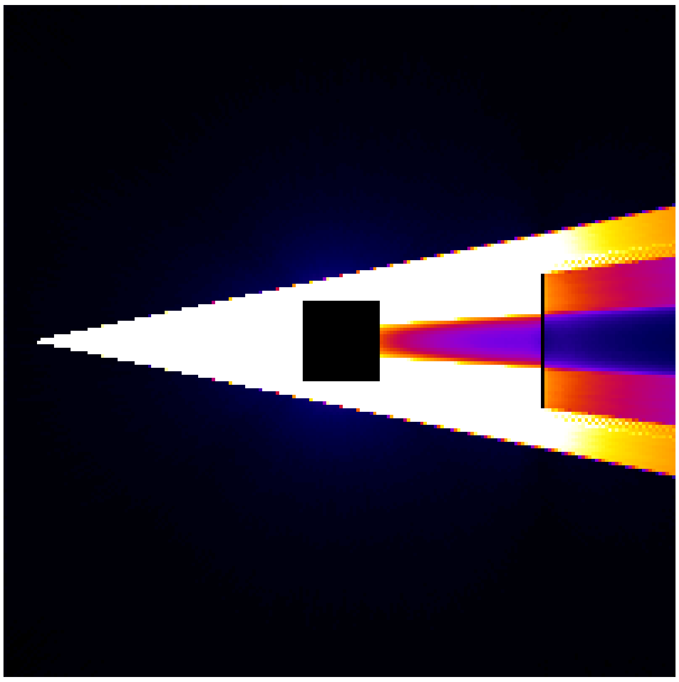
```
point_source = GGEMSXRaySource('point_source')
point_source.set_source_particle_type('gamma')
point_source.set_number_of_particles(100000000)
point_source.set_position(-900.0, 0.0, 0.0, 'mm')
point_source.set_rotation(0.0, 0.0, 0.0, 'deg')
point_source.set_beam_aperture(12.0, 'deg')
point_source.set_focal_spot_size(0.0, 0.0, 0.0, 'mm')
point_source.set_monoenergy(60.0, 'keV')
```



Dose absorbed by cylinder phantom


Photon tracking in phantom


Cylinder projection on flat panel detector


World photon tracking

Performance on Windows 11 system and Visual C++ 2022:

Device	Computation Time [s]
GeForce GTX 1050 Ti	353
Quadro P400	920
Xeon X-2245 8 cores / 16 threads	346
GeForce GTX 1050 Ti (50%) Quadro P400 (50%)	198

Performance on Ubuntu 20.04 and GNU GCC 9.3:

Device	Computation Time [s]
GeForce GTX 1050 Ti	400
Quadro P400	960
Xeon X-2245 8 cores / 16 threads	360
GeForce GTX 1050 Ti (40%) Quadro P400 (60%)	240

Examples 6: OpenGL Visualization

Since GGEMS v1.2, the OpenGL library can be activated.

```
python visualization.py [-h] [-v VERBOSE] [-e] [-w WDIMS WDIMS] [-m MSAA] [-a] [-p NPARTICLE
-h/--help              show this help message and exit
-v/--verbose           Set level of verbosity
-e/--nogl              Disable OpenGL
-w/--wdims             OpenGL window dimensions (default: [800, 800])
                       for a 400x400 window: -w 400 400
-m/--msaa              MSAA factor (1x, 2x, 4x or 8x) (default: 8)
-a/--axis              Drawing axis in OpenGL window
-p/--nparticlesgl      Number of displayed primary particles on OpenGL window (max: 65536) (de
-b/--drawgeom          Draw geometry only on OpenGL window (default: False)
-c/--wcolor            Background color of OpenGL window (default: black)
                       using blue color for background: -c "blue"
```

```
-d/--device          OpenCL device running visualization (default: 0)
                     only 1 device available for this example
                     using device index 0: -d 0
-n/--nparticles      Number of particles (default: 1000000)
-s/--seed            Seed of pseudo generator number (default: 777)
```

Create instance of GGEMSOpenGLManager:

```
opengl_manager = GGEMSOpenGLManager()
```

Many parameters can be set for OpenGL:

```
opengl_manager = GGEMSOpenGLManager()
opengl_manager.set_window_dimensions(window_dims[0], window_dims[1])
opengl_manager.set_msaa(msaa)
opengl_manager.set_background_color(window_color)
opengl_manager.set_draw_axis(is_axis)
opengl_manager.set_world_size(3.0, 3.0, 3.0, 'm')
opengl_manager.set_image_output('data/axis')
opengl_manager.set_displayed_particles(number_of_displayed_particles)
opengl_manager.set_particle_color('gamma', 152, 251, 152)
# opengl_manager.set_particle_color('gamma', color_name='red') # Using registered color
opengl_manager.initialize()
```

Display only geometry and system:

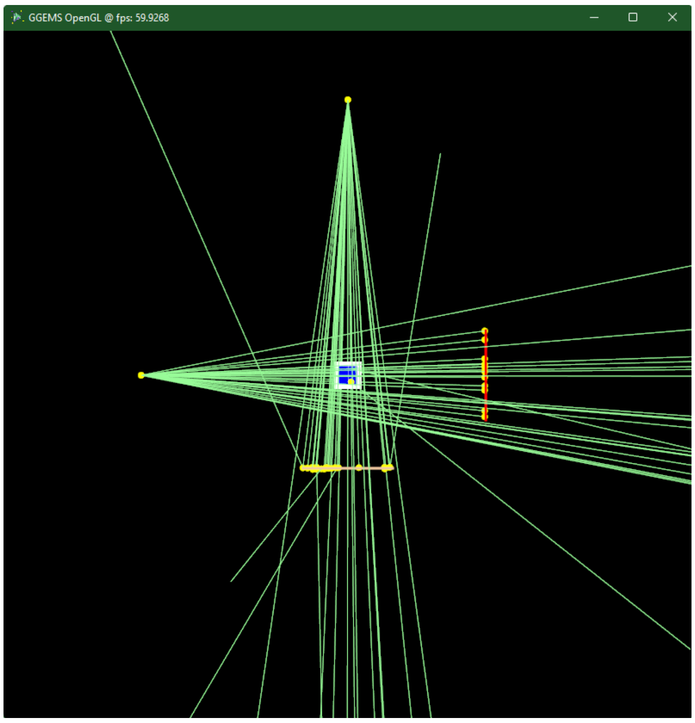
```
opengl_manager.display()
```

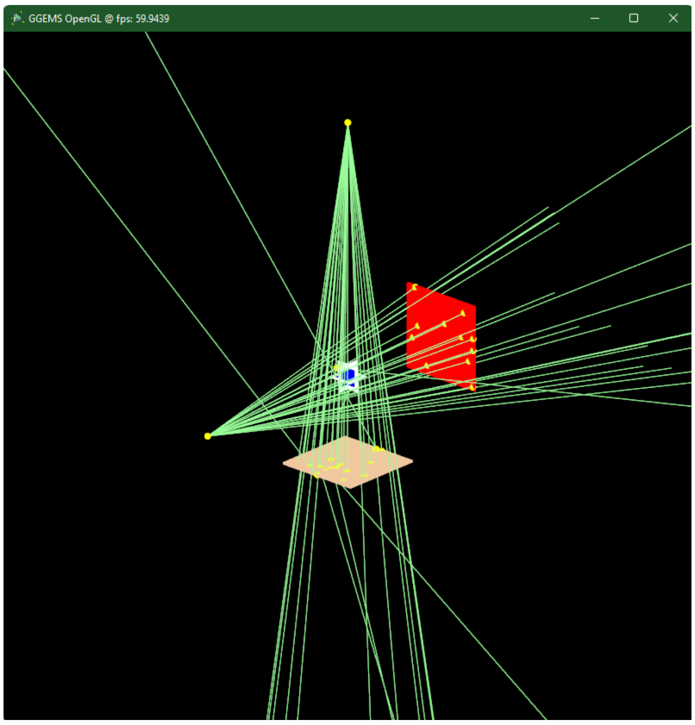
Using GGEMS for a complete visualization (geometry, system and particles):

```
ggems.run()
```

The OpenGL window is interactive, the scene can be moved using keyboard or mouse:

```
Keys:
    * [Esc/X]             Quit application
    * [C]                 Wireframe view
    * [V]                 Solid view
    * [R]                 Reset view
    * [K]                 Save current window to a PNG file
    * [+/-]               Zoom in/out
    * [Up/Down]           Move forward/back
    * [W/S]
    * [Left/Right]        Move left/right
    * [A/D]

Mouse:
    * [Scroll Up/Down]    Zoom in/out
    * [Left button]       Rotation
    * [Middle button]     Translation
```





Release Notes

Supported and Tested Platforms

GGEMS has been tested only on 64 bits architecture.

Platforms:

Linux: Ubuntu 20.04 LTS

Windows: Windows 11

Compilers:

Linux: gnu g++ 9.3, clang 10, Intel oneAPI DC++/C++ 2022.0.0

Windows: Visual C++ 19.30, clang 13, Intel oneAPI DC++/C++ 2022.0.0

OpenCL devices:

Intel

Xeon E5-2680, Xeon W-2245

HD Graphics 530

NVIDIA

Quadro P400, P2000

GeForce GTX 980 Ti, 1050 Ti, 1080 Ti

OpenGL using:

GLFW v3.3.6

GLEW v2.1.0

GLM v0.9.9.9

What You Can Do in GGEMS

Applications:

CT and CBCT systems

Photon dosimetry

Sources:

Analytical X-ray source using spectrum

Point source and rectangular source derived for X-ray source

Volume:

Voxelized phantom

Physical processes:

Compton scattering

Rayleigh scattering

Photoelectric effect

Particles:

Photon

Output:

Raw file

MHD file

Compilation Warnings

There may be a few compilation warnings on some platforms, particularly on MacOS, where GGEMS has not been tested.

GGEMS Software License

A Software License applies to the GGEMS code. Users must accept this license in order to use it. The details and the list of copyright holders is available at https://ggems.fr/about and also in the text file LICENSE distributed with the source code.

Change Log

CMAKE

- Intel oneAPI DPC++/C++ can be selected for Unix and Windows users
- CUDA package is required to find OpenCL library properly using GPU architecture
- OpenGL visualization in option. OpenGL requires GLFW3, GLEW and GLM libraries
- The cmake-config directory is added in CMAKE_MODULE_PATH

GGEMS

- New GGEMSRGBColor structure storing color
- In GGEMSMaterialsDatabaseManager, default colors are set to materials
- Files ‘std_image.h’ and ‘std_image_write.h’ added to manage files
- OpenGL library can be activated in GGEMS for visualization
- New GGEMSOpenGLManager singleton C++ class managing OpenGL
- New GGEMSOpenGLAxis class drawing X, Y and Z axis in GGEMS windows
- New GGEMSOpenGLVolume class storing informations and pointers of OpenGL volume
- New GGEMSOpenGLPrism class drawing cone section
- New GGEMSOpenGLParaGrid class drawing matrix of voxels
- New GGEMSOpenGLSphere class drawing sphere
- New GGEMSOpenGLParticles class drawing particles
- New GGEMSAttenuations class computing attenuation and energy-absorption coefficient values and loading these values to OpenCL device
- New GGEMSMuDataConstants namespace storing attenation values for materials from Hydrogen to Uranium in an energy interval 1 keV to 1 MeV
- New GGEMSMuMuEnData structure storing OpenCL buffer to attenuation and energy-absorption coefficient values

Fixed Bugs

- Bug in deallocation for GGEMSVolumeCreatorManager class
- Bug in text parser

Features

- TLE (Track Length Estimator) method for dosimetry application
- OpenGL visualization in GGEMS

Examples

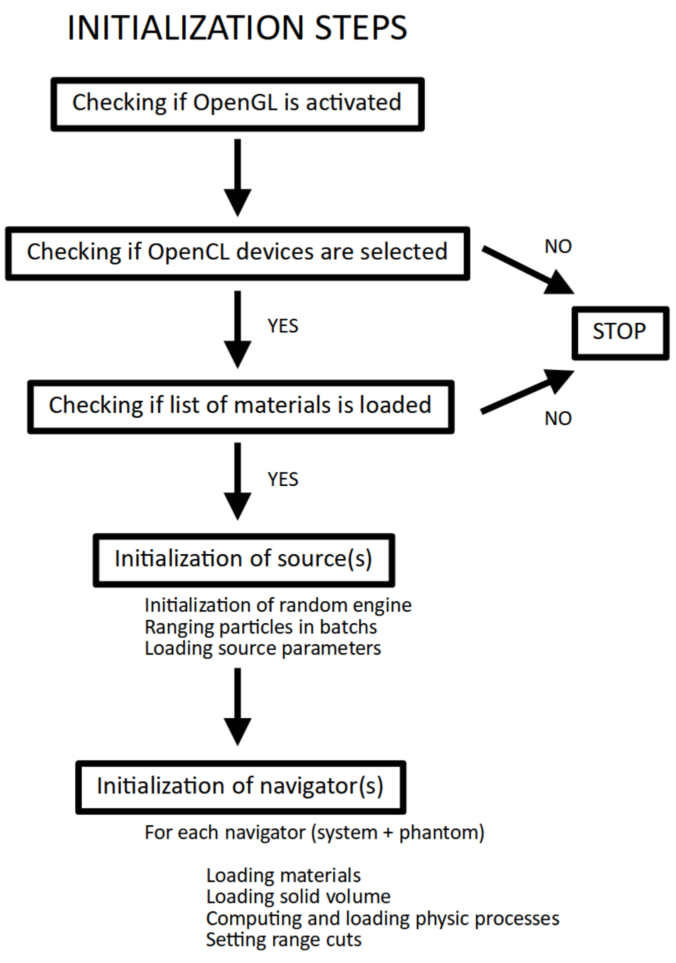
- New example 6_Visualization illustrating OpenGL in GGEMS

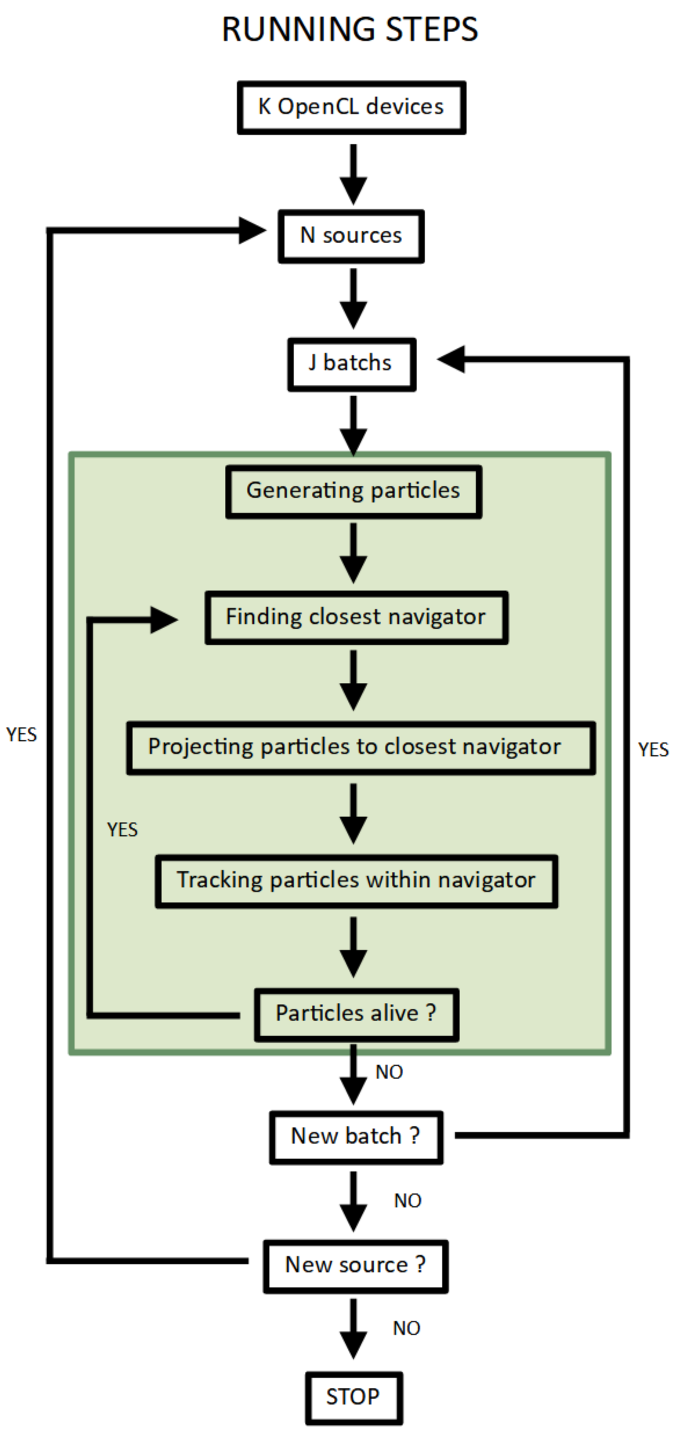
GGEMS Design

The GGEMS class has two important steps:

- Initialization
- Running

The following schemes summarize these steps.


INITIALIZATION STEPS
Checking if OpenGL is activated
Checking if OpenCL devices are selected
NO
STOP
YES
Checking if list of materials is loaded
NO
YES
Initialization of source(s)
Initialization of random engine
Ranging particles in batchs
Loading source parameters
Initialization of navigator(s)
For each navigator (system + phantom)
Loading materials
Loading solid volume
Computing and loading physic processes
Setting range cuts

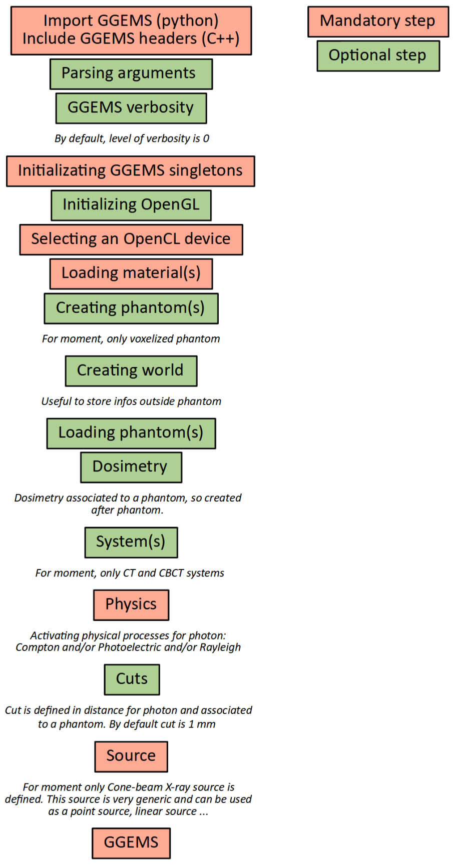


Make a GGEMS Project

GGEMS is designed as a library, and can be called using either python or C++. The performance are the same.

Template

GGEMS (C++ or python) macros are writting following this template:


Import GGEMS (python)
Include GGEMS headers (C++)
Parsing arguments
GGEMS verbosity
By default, level of verbosity is 0
Initializating GGEMS singletons
Initializing OpenGL
Selecting an OpenCL device
Loading material(s)
Creating phantom(s)
For moment, only voxelized phantom
Creating world
Useful to store infos outside phantom
Loading phantom(s)
Dosimetry
Dosimetry associated to a phantom, so created after phantom.
System(s)
For moment, only CT and CBCT systems
Physics
Activating physical processes for photon: Compton and/or Photoelectric and/or Rayleigh
Cuts
Cut is defined in distance for photon and associated to a phantom. By default cut is 1 mm
Source
For moment only Cone-beam X-ray source is defined. This source is very generic and can be used as a point source, linear source ...
GGEMS
Mandatory step
Optional step

Python

Warning

Check your PYTHONPATH environment variable is set correctly.

Using GGEMS with python is very simple. In a folder storing your project write a python file importing GGEMS.

```
from ggems import *
```

Verbosity level is defined in the range [0;3]. For a silent GGEMS execution, the level is set to 0, otherwise 3 for lot of informations.

```
GGEMSVerbosity(0)
```

Necessary singletons should be called.

```
opencl_manager = GGEMSOpenCLManager()
opengl_manager = GGEMSOpenGLManager()
materials_database_manager = GGEMSMaterialsDatabaseManager()
processes_manager = GGEMSProcessesManager()
range_cuts_manager = GGEMSRangeCutsManager()
volume_creator_manager = GGEMSVolumeCreatorManager()
```

An OpenCL device could be selected.

```
opencl_manager.set_device_index(0)
```

A material database has to be loaded in GGEMS. A material file is provided in GGEMS in ‘data’ folder. This file can be copy and paste in your project, and a new material can be added inside it.

```
materials_database_manager.set_materials('materials.txt')
```

Photon physical processes are activated using the process name, the particle name and the associated phantom name (or ‘all’ for all defined phantoms).

```
processes_manager.add_process('Compton', 'gamma', 'all')
processes_manager.add_process('Photoelectric', 'gamma', 'all')
processes_manager.add_process('Rayleigh', 'gamma', 'all')
```

Physical tables can be customized by changing the number of bins and the energy range. The following values are the default values.

```
processes_manager.set_cross_section_table_number_of_bins(220)
processes_manager.set_cross_section_table_energy_min(1.0, 'keV')
processes_manager.set_cross_section_table_energy_max(10.0, 'MeV')
```

Range cuts are defined in distance, particle type has to be specified and cuts are associated to a phantom (or ‘all’ for all defined phantoms). The distance is converted in energy during the initialization step. During the simulation, if energy particle is below to a cut, the particle is killed and the energy is locally deposited.

```
range_cuts_manager.set_cut('gamma', 0.1, 'mm', 'all')
```

GGEMS can be launched using the GGEMS python class. All verboses can be set to ‘True’ or ‘False’. In ‘tracking_verbose’, the second parameters is the index of particle to track. The method ‘initialize’ intializes all objects in GGEMS, and the simulation starts with the method ‘run’.

```
ggems = GGEMS()
ggems.opencl_verbose(True)
ggems.material_database_verbose(True)
ggems.navigator_verbose(True)
ggems.source_verbose(True)
ggems.memory_verbose(True)
ggems.process_verbose(True)
ggems.range_cuts_verbose(True)
ggems.random_verbose(True)
ggems.profiling_verbose(True)
ggems.tracking_verbose(True, 0)

ggems.initialize(seed) #using a custom seed
# ggems.initialize()
ggems.run()
```

The last step, exit GGEMS properly by cleaning OpenCL:

```
ggems.delete()
opencl_manager.clean()
exit()
```

C++

Warning

Check your PATH (on Windows) and LD_LIBRARY_PATH (on Linux) environment variables are set correctly.

Warning

For better compatibility we recommend to compile your GGEMS applications in C++ with the same compiler as the one used to compile the GGEMS library.

First method (recommended method)

In C++, the easiest way to compile your own code is to create your application in the ‘example’ directory of GGEMS, and declare your application in the CMAKE of GGEMS. To do this, edit the ‘GGEMS/examples/CMakeLists.txt’ file:

```
# ...
ADD_SUBDIRECTORY(5_World_Tracking)
# ...
ADD_SUBDIRECTORY(MY_PROJECT)
# ...
```

In the previous example, we considered that you created a folder named ‘MY_PROJECT’ in GGEMS example directory.

From this step, you can take inspiration from other GGEMS examples by adding (optional) ‘src’ and ‘include’ directories if you need to implement new classes. Here is a complete sample CMakeLists.txt file:

```
PROJECT(MY_PROJECT)

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include)
FILE(GLOB my_source ${PROJECT_SOURCE_DIR}/src/*.cc)

ADD_EXECUTABLE(my_project main.cc ${my_source})
TARGET_LINK_LIBRARIES(my_project ggems)

INSTALL(DIRECTORY ${CMAKE_CURRENT_SOURCE_DIR} DESTINATION ggems/examples)
INSTALL(TARGETS my_project DESTINATION ggems/examples/MY_PROJECT)
```

Your application will be compiled together with the GGEMS library.

Second method

Create a project folder (named ‘my_project’ for instance), then create two other folders ‘include’ and ‘src’ if custom C++ classes are written. Create two files: ‘main.cc’ and ‘CMakeLists.txt’. At this stage, the folder structure is:

```
<my_project>
|-- include\
|-- src\
|-- main.cc
|-- CMakeLists.txt
```

A CMakeLists.txt file is required to build a C++ executable for GGEMS. An example is given on Windows using Visual C++.

```
CMAKE_MINIMUM_REQUIRED(VERSION 3.13 FATAL_ERROR)

SET(ENV{CC} "cl")
SET(ENV{CXX} "cl")

PROJECT(MYPROJECT LANGUAGES CXX)

# Finding CUDA and OpenCL
FIND_PACKAGE(CUDA REQUIRED)
SET(OpenCL_ROOT "${CUDA_TOOLKIT_ROOT_DIR}")
FIND_PACKAGE(OpenCL REQUIRED)
IF(${OpenCL_VERSION_STRING} VERSION_GREATER "1.2")
  ADD_DEFINITIONS(-DCL_USE_DEPRECATED_OPENCL_1_2_APIS)
ENDIF()

# If OpenGL visualization if necessary
OPTION(OPENGL_VISUALIZATION "Using OpenGL for visualization" OFF)
IF(OPENGL_VISUALIZATION)
  ADD_DEFINITIONS(-DOPENGL_VISUALIZATION)
  FIND_PACKAGE(GLFW3 REQUIRED)
  FIND_PACKAGE(OpenGL REQUIRED)
  SET(GLEW_USE_STATIC_LIBS TRUE) # Using GLEW in static, important for Python!!!
  FIND_PACKAGE(GLEW REQUIRED)
  FIND_PACKAGE(glm REQUIRED)
  GET_FILENAME_COMPONENT(GLM_INCLUDE_DIR ${glm_DIR}/../../../include ABSOLUTE)
ENDIF()

# Giving path to GGEMS headers and to GGEMS librarie
SET(GGEMS_INCLUDE_DIRS "" CACHE PATH "Path to the GGEMS include directory")
SET(GGEMS_LIBRARY "" CACHE FILEPATH "Path to GGEMS library (.lib or .so)")

# Forcing release mode
SET(CMAKE_BUILD_TYPE "Release" CACHE STRING "Choose the type of build, options are: Debug Re
SET_PROPERTY(CACHE CMAKE_BUILD_TYPE PROPERTY STRINGS "Debug" "Release")

# Compiling using c++17
set(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} /Ox /std:c++17")

INCLUDE_DIRECTORIES(${PROJECT_SOURCE_DIR}/include ${GGEMS_INCLUDE_DIRS})

FILE(GLOB source ${PROJECT_SOURCE_DIR}/src/*.cc)

ADD_EXECUTABLE(my_project main.cc ${source})
IF(OPENGL_VISUALIZATION)
  TARGET_LINK_LIBRARIES(my_project OpenCL::OpenCL ${GGEMS_LIBRARY} ${GLFW3_LIBRARY} OpenGL::
ELSE()
  TARGET_LINK_LIBRARIES(my_project OpenCL::OpenCL ${GGEMS_LIBRARY})
ENDIF()
```

In main.cc file, GGEMS files are included:

```
#include <GGEMS/global/GGEMS.hh>
#include <GGEMS/global/GGEMSOpenCLManager.hh>
#include <GGEMS/graphics/GGEMSOpenGLManager.hh>
// ...
```

For silent GGEMS execution, the level is set to 0, otherwize 3 for maximum informations.

```
GGcout.SetVerbosity(0);
GGcerr.SetVerbosity(0);
GGwarn.SetVerbosity(0);
```

Necessary singletons should be called.

```
GGEMSOpenCLManager& opencl_manager = GGEMSOpenCLManager::GetInstance();
GGEMSMaterialsDatabaseManager& material_manager = GGEMSMaterialsDatabaseManager::GetInstance
GGEMSVolumeCreatorManager& volume_creator_manager = GGEMSVolumeCreatorManager::GetInstance()
GGEMSProcessesManager& processes_manager = GGEMSProcessesManager::GetInstance();
GGEMSRangeCutsManager& range_cuts_manager = GGEMSRangeCutsManager::GetInstance();
```

An OpenCL device could be selected:

```
opencl_manager.DeviceToActivate(0);
```

A material database has to be loaded in GGEMS. A material file is provided in GGEMS in ‘data’ folder. This file can be copy and paste in your project, and a new material can be added inside it.

```
material_manager.SetMaterialsDatabase("materials.txt");
```

Photon physical processes are activated using the process name, the particle name and the associated phantom name (or ‘all’ for all defined phantoms).

```
processes_manager.AddProcess("Compton", "gamma", "all");
processes_manager.AddProcess("Photoelectric", "gamma", "all");
processes_manager.AddProcess("Rayleigh", "gamma", "all");
```

Physical tables can be customized by changing the number of bins and the energy range. The following values are the default values.

```
processes_manager.SetCrossSectionTableNumberOfBins(220);
processes_manager.SetCrossSectionTableMinimumEnergy(1.0f, "keV");
processes_manager.SetCrossSectionTableMaximumEnergy(1.0f, "MeV");
```

Range cuts are defined in distance, particle type has to be specified and cuts are associated to a phantom (or 'all' for all defined phantoms). The distance is converted in energy during the initialization step. During the simulation, if energy particle is below to a cut, the particle is killed and the energy is locally deposited.

```
range_cuts_manager.SetLengthCut("all", "gamma", 0.1f, "mm");
```

GGEMS can be launched using the GGEMS python class. All verboses can be set to 'True' or 'False'. In 'tracking_verbose', the second parameters is the index of particle to track. The method 'initialize' intializes all objects in GGEMS, and the simulation starts with the method 'run'.

```
GGEMS ggems;
ggems.SetOpenCLVerbose(true);
ggems.SetMaterialDatabaseVerbose(true);
ggems.SetNavigatorVerbose(true);
ggems.SetSourceVerbose(true);
ggems.SetMemoryRAMVerbose(true);
ggems.SetProcessVerbose(true);
ggems.SetRangeCutsVerbose(true);
ggems.SetRandomVerbose(true);
ggems.SetProfilingVerbose(true);
ggems.SetTrackingVerbose(true, 0);

ggems.Initialize(seed); // using a custom seed
// ggems.Initialize();
ggems.Run();
```

The last step, exit GGEMS properly by cleaning OpenCL:

```
GGEMSOpenCLManager::GetInstance().Clean();
```

Portable Document

GGEMS documentation can be downloaded in PDF format from the following link:

GGEMS pdf