PenRed: User Manual

V. Giménez-Alventosa[1], V. Giménez Gómez[2], and S. Oliver[3]

[1]Departament de Física Atòmica Molecular i Nuclear, Universitat de València , Dr. Moliner, 50, 46100, Burjassot, València, Spain, gimenez.alventosa.vicent@gmail.com
[2]Departament de Física Teòrica and IFIC, Universitat de València-CSIC , Dr. Moliner, 50, 46100, Burjassot, València, Spain,
[3]Instituto de Seguridad Industrial, Radiofísica y Medioambiental (ISIRYM), Universitat Politècnica de València, Camí de Vera s/n, 46022, València, Spain, sanolgi@upvnet.upv.es

May 27, 2025

Abstract

The present document is a manual of the PenRed code system which content is focused on the description of the framework usage. PenRed is a general-purpose, stand-alone and extensible framework code based on PENELOPE for parallel Monte Carlo simulations of radiation transport through matter. It has been implemented in C++ programming language and takes advantage of modern object-oriented technologies. In addition, PenRed offers, among other features, the capability to read and process DICOM images to perform simulations. These feature facilitate its usage in medical applications. Our framework has been successfully tested against the original PENELOPE Fortran code.

1 Introduction

The present document contains all the necessary information to use the PenRed [1] framework. That is, all the available modules already included in the package and the provided main program. This document is not intended to describe how to implement new components or describe in detail the code structure. That information can be included in a future manual.

1.1 What is PenRed?

PenRed is a fully parallel, modular and customizable framework for Monte Carlo simulations of the passage of radiation through matter. It is based on the PENELOPE [2] code system, from which inherits its physics models and tracking algorithms for charged particles. For further details on the interactions models, the reader is referred to the PENELOPE manual [3]. PenRed has been coded in C++ following an object-oriented programming paradigm

V. Giménez-Alventosa https://orcid.org/0000-0003-1646-6094
V. Giménez Gómez https://orcid.org/0000-0003-3855-2567
S. Oliver https://orcid.org/0000-0001-8258-3972

restricted to the C++14 standard. Our engine implements parallelism via a double approach: on the one hand, by using standard C++ threads for shared memory, improving the access and usage of the memory, and, on the other hand, via the MPI standard for distributed memory infrastructures. Notice that both kinds of parallelism can be combined together in the same simulation. In addition, PenRed provides a modular structure with methods designed to easily extend its functionality. Thus, users can create their own independent modules to adapt our engine to their needs without changing the existing ones. Furthermore, user extensions will take advantage of the built-in parallelism without any extra effort or knowledge of parallel programming.

1.2 Document structure

The manual is organized as follows. The section 2 shows a useful guide of installation and execution of the framework to run simulations. Section 3 provides some information of the internal data structure format used to configure the simulations. In section 4 a briefly description of important definitions, constants, parameters and units is shown. Following, section 5, shows how to execute the main program provided within the PenRed package, and describes all the implemented tallies, sources, geometries, variance reduction techniques and other parameters to configure the simulation. Then, in the section 6 describes *pyPenred* a python binding developed to use PenRed through python programming. Finally, the section 7 describe the examples located in the `examples` folder, which are ready to be simulated. Finally, the offered utilities in the PenRed package and how to use them are explained in the section 8.

Notice that the section 4 summarise some useful definitions necessary to understand the code and its operation.

Contents

2 Installation and execution

Although PenRed can be compiled with no external dependencies, some features depend on external libraries that should be installed to activate the corresponding functionalities (see below). It is the user's responsibility to ensure these external dependencies are installed before enabling a feature. Users should refer to the installation guides of the respective optional packages.

PenRed can be compiled with a compiler supporting the C++14 standard and CMake[1] version 3.11 or greater. Once a suitable C++ compiler is available, to compile PenRed, first download the source code from our GitHub repository[2]. This can be done manually from the repository page or using *git* with the following command line instruction:

```
git clone https://github.com/PenRed/PenRed.git
```

With the code downloaded, navigate to the `src` folder to compile it. This folder includes a CMakeList file and two scripts to compile it automatically. Depending on the operating system:

- **Linux/Unix**: Run the bash script *compile.sh* with

  ```
  bash compile.sh
  ```

- **Windows**: Run the batch script *compile.bat*

The compilation scripts will create a *build* folder where all the temporary files required by the compilation will be stored. Finally, a *compilation* folder will be created in the *src* directory with all generated executables. Among all the executables, the PenRed main program, the one used to run the simulation examples, is located at:

src/compiled/mains/pen_main

Editing the corresponding script, allows the user to enable/disable the following main optional features:

- **DICOMs**: If enabled, PenRed's capabilities to read and simulate DICOM images will be activated. This option requires the DICOM toolkit (dcmtk)[4]. The build system will use the version installed on the system, if available. Otherwise, the **dcmtk** library and its required dependencies will be automatically downloaded and compiled. Please note that this process may take several minutes, depending on the performance of the computer.

- **Multi-threading**: This option enables multi-threading capabilities. PenRed implements multi-threading via the standard thread library specified in the C++ standard. Thus, no extra library is required to enable this option.

- **MPI**: This option enables MPI simulations. It requires a library with an implementation of the MPI standard, such as OpenMPI[5] or MPICH[6].

- **Load Balance**: Enables load balancing between threads and MPI processes. This option requires at least multi-threading capabilities, as it uses threads to handle communications.

[1]`https://cmake.org/download/`
[2]`https://github.com/PenRed/PenRed`

These external libraries can be found in most Linux and Unix package repositories or be downloaded for Windows. For example, to install the *dcmtk* packages on Fedora, the user can use the following `dnf` command:

```
sudo dnf install dcmtk dcmtk-devel
```

and also its dependencies,

```
sudo dnf install libjpeg libjpeg-devel libxml2 libxml2-devel openssl openssl-devel
```

2.1 Manual installation with CMake

To access all available compilation options, users can either modify the compilation script by adding definitions manually or use *CMake* to select the desired options.

In this case, we will use the *CMake* application directly. First, download *CMake* from repositories or its webpage[3] and follow the installation instructions. Note that a C++ compiler is still required.

Once installed, the *CMake* utility can be used with its user interface or directly via the command line.

2.1.1 User interface

To configure PenRed using the *CMake* user interface, open it and set the source folder to the PenRed *src* directory, and the build folder to the desired path where the intermediate files will be stored (Figure 1). It is recommended to create a new directory for the build path, such as a folder named *build* in the *src* directory.

[3] `https://cmake.org/download/`

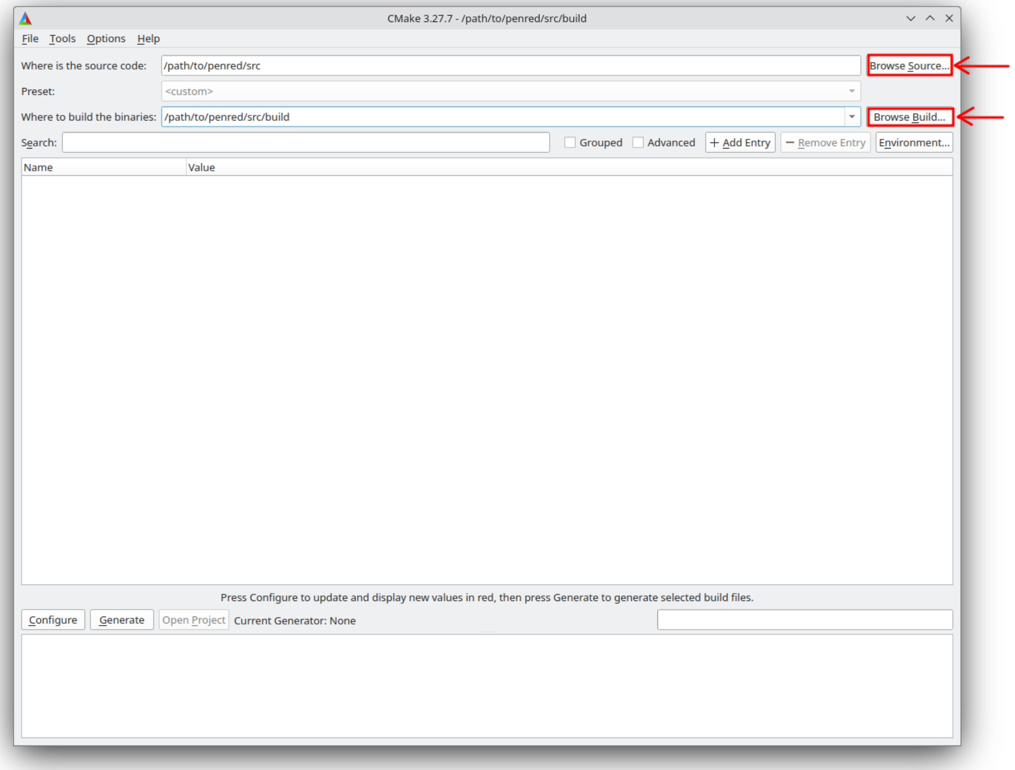
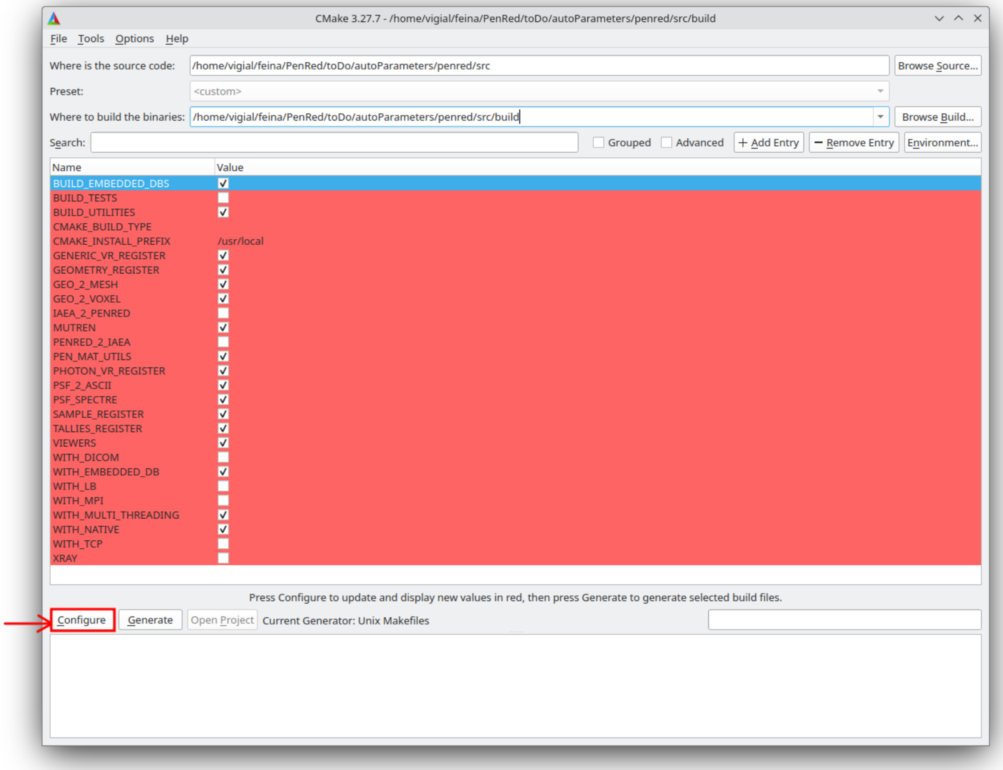

CMake 3.27.7 - /path/to/penred/src/build
File Tools Options Help
Where is the source code: /path/to/penred/src
Browse Source...
Preset: <custom>
Where to build the binaries: /path/to/penred/src/build
Browse Build...
Search:
Grouped
Advanced
+ Add Entry
− Remove Entry
Environment...
Name
Value
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: None

Figure 1: Set source and build paths with cmake user interface.

Then, push the *configure* button at the bottom left of the window. This step will generate the required files in the **build** folder to show a list of all compilation options (Figure 2).



Where is the source code: /home/vigial/feina/PenRed/toDo/autoParameters/penred/src

Preset: <custom>

Where to build the binaries: /home/vigial/feina/PenRed/toDo/autoParameters/penred/src/build

Search:

Name	Value
BUILD_EMBEDDED_DBS	✓
BUILD_TESTS	
BUILD_UTILITIES	✓
CMAKE_BUILD_TYPE	
CMAKE_INSTALL_PREFIX	/usr/local
GENERIC_VR_REGISTER	✓
GEOMETRY_REGISTER	✓
GEO_2_MESH	✓
GEO_2_VOXEL	✓
IAEA_2_PENRED	
MUTREN	✓
PENRED_2_IAEA	
PEN_MAT_UTILS	✓
PHOTON_VR_REGISTER	✓
PSF_2_ASCII	✓
PSF_SPECTRE	✓
SAMPLE_REGISTER	✓
TALLIES_REGISTER	✓
VIEWERS	✓
WITH_DICOM	
WITH_EMBEDDED_DB	✓
WITH_LB	
WITH_MPI	
WITH_MULTI_THREADING	✓
WITH_NATIVE	✓
WITH_TCP	
XRAY	

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Unix Makefiles

Figure 2: Configure project with cmake user interface, generating a list of compilation options.

Once the options have been selected, press the *generate* button to create all the files in the *build* folder. Finally, if using a compiler with an IDE, like MSVC, to compile the code, the *Open Project* button will open the project with the corresponding IDE, where the user can build and install the package.

If not, and the *Open Project* button remains disabled, the build must be done by command line in two steps. First, navigate to the build folder with the instruction:

```
cd path/to/folder/build
```

then, run the *CMake* build and install instruction:

```
cmake --build .  --config Release --target install
```

Once completed, the corresponding executables will be stored in the *compiled* folder inside the penRed *src* directory. The main program is stored in:

src/compiled/mains/pen_main

2.1.2 Command line

To configure the build options via command line, first, navigate to the *src* directory, create a **build** folder and go inside:

```
cd /path/to/PenRed/repository/src

mkdir build

cd build
```

Then, run the *ccmake* command

```
ccmake ../
```

This will open a terminal application to configure the PenRed build. First, configure the project by pressing *c* on the keyboard. If no error occurs, a list of compilation options will be displayed. The user can navigate and edit the options with the keyboard. Once the configuration is set, generate the build system by pressing *g*. If the *g* option does not appear, press the configure button again. The build system is now ready to be compiled. Exit the *ccmake* application with *q* and, in the same folder, run the following instruction:

```
cmake --build .  --config Release --target install
```

Once the code has been compiled, the penRed main program executable, ready to simulate, can be found at,

src/compiled/mains/pen_main

2.2 MSVS installation

To compile the PenRed code using MSVS, first, select "Clone a repository" from the Visual Studio start window (Figure 3).

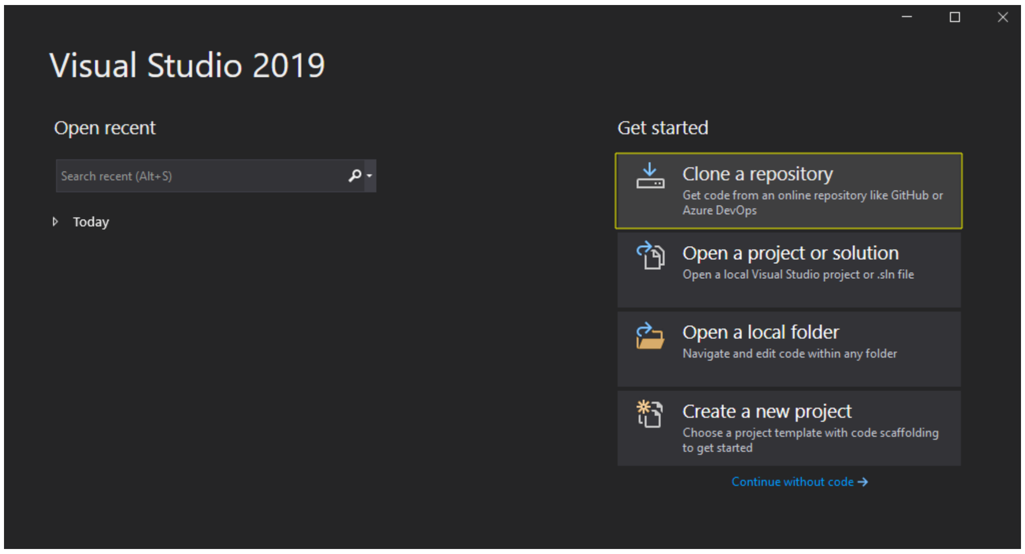
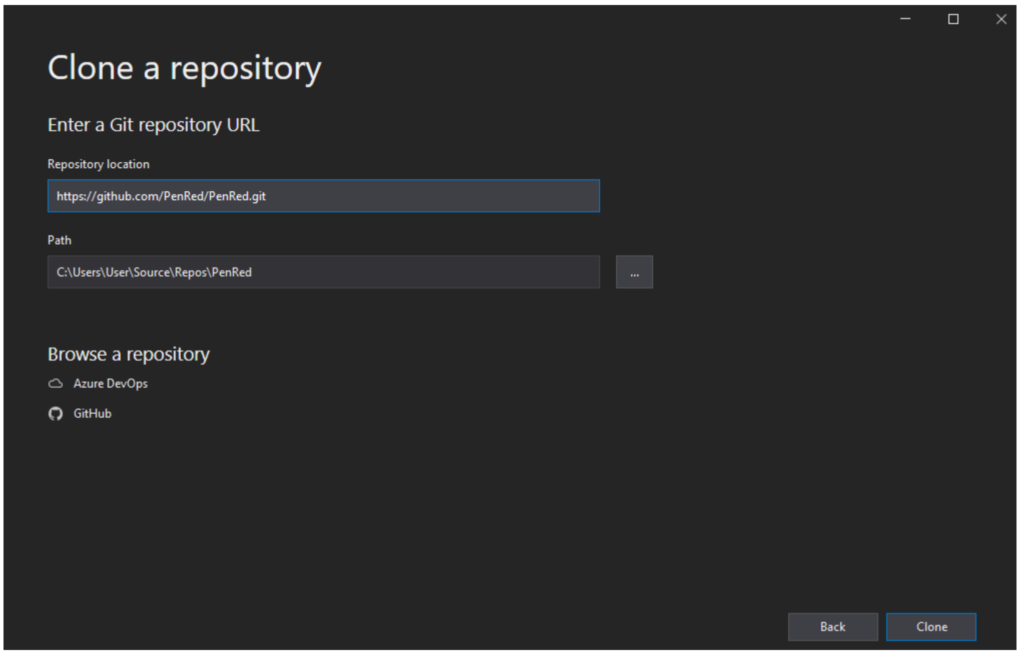

Visual Studio 2019
Open recent
Search recent (Alt+S)
Today
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

Figure 3: Clone a repository in MSVS

Then, set the PenRed repository url and push on the clone button (Figure 4).

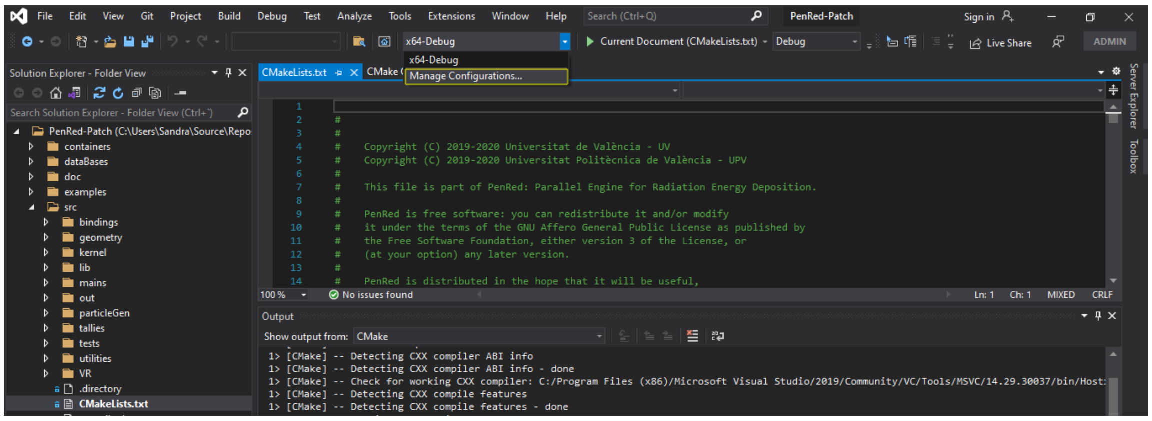


Repository location: https://github.com/PenRed/PenRed.git

Path: C:\Users\User\Source\Repos\PenRed

Figure 4: Clone PenRed repository with MSVS

The download will start automatically. Once the Cmake configuration ends, to avoid compiling with a debug profile, add a new configuration with the configuration manager (Figure 5).



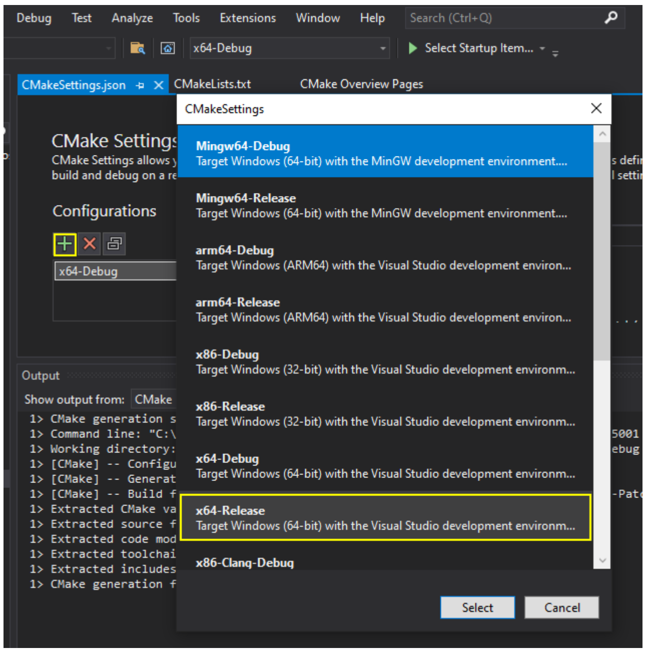
```

#
#
#   Copyright (C) 2019-2020 Universitat de València - UV
#   Copyright (C) 2019-2020 Universitat Politècnica de València - UPV
#
#   This file is part of PenRed: Parallel Engine for Radiation Energy Deposition.
#
#   PenRed is free software: you can redistribute it and/or modify
#   it under the terms of the GNU Affero General Public License as published by
#   the Free Software Foundation, either version 3 of the License, or
#   (at your option) any later version.
#
#   PenRed is distributed in the hope that it will be useful,
```

```
1> [CMake] -- Detecting CXX compiler ABI info
1> [CMake] -- Detecting CXX compiler ABI info - done
1> [CMake] -- Check for working CXX compiler: C:/Program Files (x86)/Microsoft Visual Studio/2019/Community/VC/Tools/MSVC/14.29.30037/bin/Host:
1> [CMake] -- Detecting CXX compile features
1> [CMake] -- Detecting CXX compile features - done
```

Figure 5: Manage MSVS configurations

In the configuration panel to the left, click on the button with the green *plus* sign to add a new configuration and select the release depending on your system. In the following image, we selected a release for 64-bit system (Figure 6).



CMake Settings allows y
build and debug on a re

```
1> CMake generation s
1> Command line: "C:\
1> Working directory:
1> [CMake] -- Configu
1> [CMake] -- Generat
1> [CMake] -- Build f
1> Extracted CMake va
1> Extracted source f
1> Extracted code mod
1> Extracted toolchai
1> Extracted includes
1> CMake generation f
```

Mingw64-Debug
Target Windows (64-bit) with the MinGW development environment....

Mingw64-Release
Target Windows (64-bit) with the MinGW development environment....

arm64-Debug
Target Windows (ARM64) with the Visual Studio development environ...

arm64-Release
Target Windows (ARM64) with the Visual Studio development environ...

x86-Debug
Target Windows (32-bit) with the Visual Studio development environm...

x86-Release
Target Windows (32-bit) with the Visual Studio development environm...

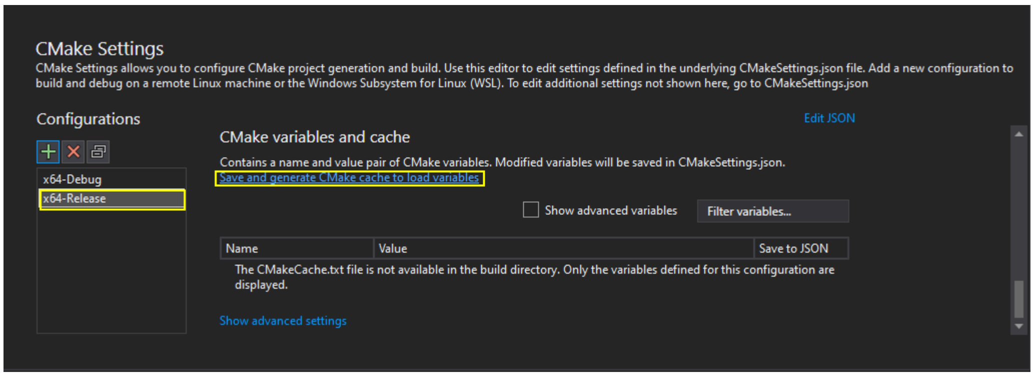
x64-Debug
Target Windows (64-bit) with the Visual Studio development environm...

x64-Release
Target Windows (64-bit) with the Visual Studio development environm...

x86-Clang-Debug

Figure 6: Add configuration in MSVS

Once you have selected the new configuration, and push on "Save and generate CMake cache to load variables" to be able to change the CMake variables for this configuration and compile it (Figure 7).



CMake Settings

CMake Settings allows you to configure CMake project generation and build. Use this editor to edit settings defined in the underlying CMakeSettings.json file. Add a new configuration to build and debug on a remote Linux machine or the Windows Subsystem for Linux (WSL). To edit additional settings not shown here, go to CMakeSettings.json

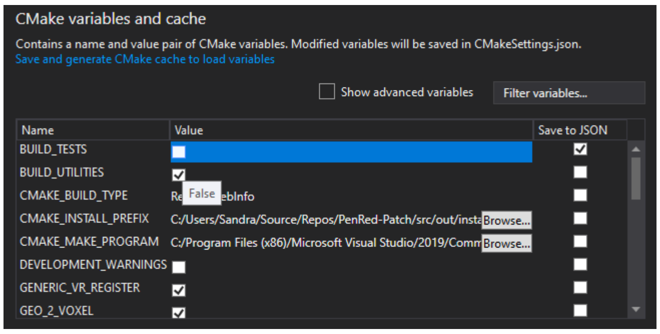
CMake variables and cache

Contains a name and value pair of CMake variables. Modified variables will be saved in CMakeSettings.json.
Save and generate CMake cache to load variables

The CMakeCache.txt file is not available in the build directory. Only the variables defined for this configuration are displayed.

Figure 7: Select configuration in MSVS

Now, we can change all the Cmake configuration variables to enable or disable MPI support and other features (Figure 8).


CMake variables and cache

Contains a name and value pair of CMake variables. Modified variables will be saved in CMakeSettings.json.
Save and generate CMake cache to load variables

☐ Show advanced variables   Filter variables...

Name	Value	Save to JSON
BUILD_TESTS	☐	✓
BUILD_UTILITIES	✓	☐
CMAKE_BUILD_TYPE	Re False ebInfo	☐
CMAKE_INSTALL_PREFIX	C:/Users/Sandra/Source/Repos/PenRed-Patch/src/out/insta Browse...	☐
CMAKE_MAKE_PROGRAM	C:/Program Files (x86)/Microsoft Visual Studio/2019/Comm Browse...	☐
DEVELOPMENT_WARNINGS	☐	☐
GENERIC_VR_REGISTER	✓	☐
GEO_2_VOXEL	✓	☐

Figure 8: Set compilation flags in MSVS

Finally, build and install PenRed (Figure 9).

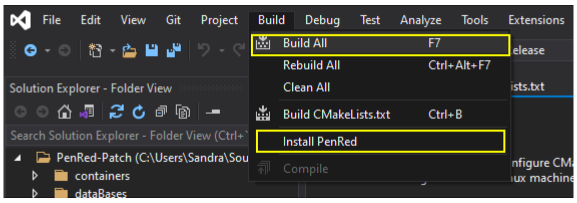

File Edit View Git Project Build Debug Test Analyze Tools Extensions
Build All F7
Rebuild All Ctrl+Alt+F7
Clean All
Build CMakeLists.txt Ctrl+B
Install PenRed
Compile
Solution Explorer - Folder View
Search Solution Explorer - Folder View (Ctrl+
PenRed-Patch (C:\Users\Sandra\Sou
containers
dataBases

Figure 9: Build and install PenRed in MSVS

If the compilation finishes successfully, a new folder named *compiled* will be created inside the *src* folder containing the PenRed's main program and the executable for all other enabled utilities (Figure 10).

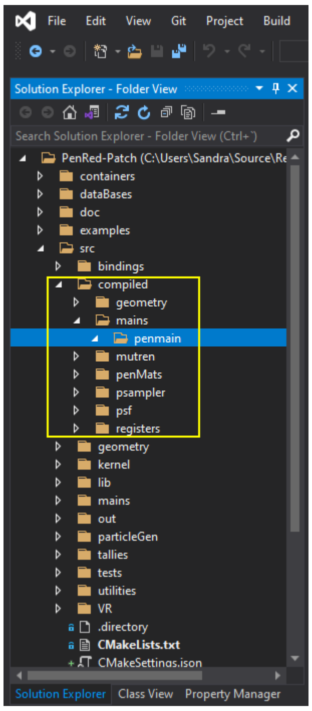


Figure 10: PenRed compiled programs

2.3 Basic usage

To execute the main program, the user needs a configuration file and, likely, the required database files, such as material and geometry files. Their paths should be specified in the configuration file, leaving the configuration file path as the only program argument. All details regarding the simulation configuration can be found in section 5. Additionally, the *examples* folder contains several configuration file examples with the corresponding material and geometry files ready to be executed (section 7). To execute the program, the user must use the command line instruction:

```
./pen_main path/to/configuration/file
```

Therefore, to run the provided examples, copy the main program executable in the corresponding example folder and run the instruction:

```
./pen_main config.in
```

If MPI capabilities have been enabled during compilation, the code should be executed as any MPI program. For example:

```
mpirun -np Nprocesses ./pen_main path/to/configuration/file
```

where *Nprocesses* specify the number of MPI processes to use. Of course, the user can use any other options of the *mpirun* command, such us specify the hosts where execute the code via the *hostfile* option.

2.4 Optional features

The following list summarise some of the optional compilation flags available during the PenRed compilation procedure.

- **BUILD_UTILITIES**: Enables or disable the compilation of utilities, such as `geo2vox`, sampler, geometry and tally registers or load balance server. If enabled, the compilation of each utility can be enabled or disabled individually.
- **WITH_MULTI_THREADING**: Handles multithreading capabilities.
- **WITH_MPI**: Handles MPI capabilities. Require an MPI library installed.
- **WITH_NATIVE**: Enable or disable the compilation with native architecture optimizations.
- **WITH_DICOM**: Handles DICOM capabilities. Requires the `dcmtk` library. If *dcmtk* is not installed, it will be downloaded and compiled automatically.
- **FORCE_DICOM_DOWNLOAD**: Forces the build system to download and compile the *dcmtk* library, even if it is already installed. This option must be used with **WITH_DICOM** enabled.
- **WITH_LB**: Enables or disables load balance capabilities. If enabled, some optional features related with the balance system can be configured:
  - **WITH_SSL**: Enables or disables SSL secure connections for TCP or HTTPS communications. Requires OpenSSL libraries and its dependences.

  - **WITH_HTTP**: Enables or disables connections to a HTTP/HTTPS rest api as balance server (Experimental feature). Requires the libCurl library.

- **BUILD_TESTS**: Enables or disable the compilation of test codes.

3 Internal data format

src/kernel/parsers

To provide a unified format for input and configuration, PenRed implements a set of data structures. This implementation can be found in the following files,

src/kernel/parsers/includes/pen_data.hh

and

src/kernel/parsers/source/pen_data.cpp

The format used in PenRed is basically a *key/value* pair where the key format is based on unix folder system, i.e. a generic key is a string with the following structure:

$/folder1/folder2/.../element$

On the other hand, the value can be a number, bool (True or False), character, string or an array of numbers, bools or characters. Notice that these types can be combined in a single array. Also, strings must be made with double quotes ("text") to be parsed correctly. For example, code 1 shows a configuration for a cylindrical dose distribution tally.

```
tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
tallies/cylDoseDistrib/print-xyz true
tallies/cylDoseDistrib/rmin 0.0
tallies/cylDoseDistrib/rmax 30.0
tallies/cylDoseDistrib/nbinsr 60
tallies/cylDoseDistrib/zmin 0
tallies/cylDoseDistrib/zmax 30.0
tallies/cylDoseDistrib/nbinsz 60
```

Code 1: Internal data example

3.1 Implementation

This section provides a brief explanation of internal data implementation. PenRed internal data structure consists of four classes where each one uses the previous class to create a more complex structure. First, *pen_parserData* is basically a union with four allowed types: *char, int, double* and *bool*. Notice that *pen_parserData* can't be a string or pointer. Next, *pen_parserArray* is a standard C++ vector of *pen_parserData* variables. The third class is *pen_parserElement*, which can store a C++ string or a *pen_parserArray* variable. Finally, *pen_parserSection* stores a C++ map where each key-value pair uses the types,

$< std :: string|pen\_parserArray >$

where the left element is the pair key and the right element is the pair value. As we mentioned at previous section, key format follows the structure of Unix paths, i.e. ,

/folder1/folder2/element.

In our schema, each "folder" will be considered as a "section". On the other hand, a key with no sub-folders is considered an "element". Notice that keys whose include an element path as a section are not allowed and the program will return an error or overwrite this element according to the called function. Thus, the following configuration can't exist in a input format map structure,

/folder1/folder2/element
/folder1/folder2/element/element2

since *element* is not a section and therefore cannot contain any other section nor element.

For further information and usage examples, consult the provided examples in *src/tests/internalData* folder.

4 Constants, parameters and definitions

This section summarises some important constants and values of the PenRed engine.

4.1 Important definitions

- **Void region/volume**: Geometry region with material index 0.
- **Primary particle**: Particle produced directly by a source and not as a consequence of any interaction of some other particle.
- **Secondary particle**: Particle produced by some interaction of other particle.
- **History**: A *history* consists of a primary particle and all its derived secondary particles.
- **Particle simulation**: Encompasses since the particle enters into a non void region of the geometry system, until it is absorbed or escaped from the system.
- **History simulation**: Includes all the particle simulations of the particles conforming the history.
- **Source simulation**: Includes all the history simulations due primary particles produced by the source.
- **Global simulation**: Consists of all source simulations.
- **Particle state**: All variables stored at the corresponding state structure, which base type should be the provided *pen_particleState*.
- **Object state**: We refer as a object state to the values of its internal variables, regardless if they are private, public or protected. Also, inherited variables are considered as part of the object state. For example, lets see the objects defined at the code 2. The state of an object of the first type *A*, consists of the variables *a1*, *a2* and *a3*. On the other hand, the state of an object *B* consists of *a1*, *a2*, *a3*, *b1* and *b2*, due the inheritance from the class *A*.

```

class A{
    protected:
        int a1,a2;
    public:
        double a3;

    A() : a1(0),a2(2),a3(5.3){}

    void do1();
    void do2();
};

class B : public A{
    private:
        int b1;
    public:
        double b2;

    B() : b1(0),{}

    void doB1();
    void doB2();
};

```

Code 2: Source configuration parameters

- **Verbose levels**: Verbose levels consists of a set of non negative indexes that indicates how verbose will be our execution. The meaning of that levels are summarised following,

  – 0: Any message should be printed.

  – 1: Only error messages should be printed.

  – 2: Errors, warnings and important information should be printed.

  – > 2: Prints all relevant information.

  Optionally, greater verbose levels could be used to filter very verbose information.

- **ILB values**: ILB labels follows the definition of the PENELOPE package, which are summarised at the table extracted from the PENELOPE manual [3]. Notice that our index begins from 0 because C++ syntax but at the FORTRAN code the first index is 1.

ILB	description
ILB[0]	Generation of the particle; 1 for primary particles, 2 for their direct descendants and so on. Primary particles are assumed to be labelled with $ILB[0] = 1$.
ILB[1]	Parent particle *kpar* index (see table 3), only if $ILB[0] > 1$.
ILB[2]	Interaction mechanism $ICOL$ (see tables at section 4.4) that originated the particle, only when $ILB[0] > 1$.
ILB[3]	A non-zero value identifies particles emitted from atomic relaxation events and describes the atomic transition where the particle was released. The numerical value is, $$ILB[3] = Z \times 10^6 + IS1 \times 10^4 + IS2 \times 100 + IS3 \qquad (1)$$ where Z is the atomic number of the emitting atom and $IS1$, $IS2$ and $IS3$ are the labels of the active atomic electron shells (see table 7). For instance, $ILB[3] = 29010300$ designates a $K - L2$ x ray from copper ($Z = 29$), and $ILB[3] = 29010304$ indicates a $K - L2 - L3$ Auger electron from the same element. When $ILB[3] \neq 0$, the value of $ILB[2]$ indicates the itneraction mechanism that caused the initial vacancy in the decaying atom.
ILB[4]	This label can be defined by the user and must be transferred to all particle descendants.

Table 1: Description of the ILB components

4.2 Units

PenRed supposes the use of specific units internally. These units are summarised at the table 2.

Magnitude	Unit
Length	cm
Energy	eV
Time	s
Material mass	g
Density	g/cm^3

Table 2: PenRed internal units.

4.3 Particle indexes

In penRed, each particle has a text identifier, known as particle name, and a numeric identifier set by an internal enumeration. Both values for each particle are summarised in the Table 3.

particle name	Enumeration identifier	Numerical index
electron	PEN_ELECTRON	0
gamma	PEN_PHOTON	1
positron	PEN_POSITRON	2

Table 3: Particle indexes and names.

4.4 Particle interaction indexes

PenRed uses enumerations to indexing the interactions of each particle. This section show the corresponding index for each particle interaction.

Interaction	**Enumeration identifier**	**Numerical index**
Elastic collision	BETAe_HARD_ELASTIC	0
Inelastic collision	BETAe_HARD_INELASTIC	1
Bremsstrahlung	BETAe_HARD_BREMSSTRAHLUNG	2
Inner shell interaction	BETAe_HARD_INNER_SHELL	3
Delta interaction	BETAe_DELTA	4
Soft interaction	BETAe_SOFT_INTERACTION	5

Table 4: Electron interaction indexes.

Interaction	**Enumeration identifier**	**Numerical index**
Rayleigh	GAMMA_RAYLEIGH	0
Compton	GAMMA_COMPTON	1
Photoelectric	GAMMA_PHOTOELECTRIC	2
Pair production	GAMMA_PAIR_PRODUCTION	3
Delta interaction	GAMMA_DELTA	4

Table 5: Photon interaction indexes.

Interaction	**Enumeration identifier**	**Numerical index**
Elastic collision	BETAp_HARD_ELASTIC	0
Inelastic collision	BETAp_HARD_INELASTIC	1
Bremsstrahlung	BETAp_HARD_BREMSSTRAHLUNG	2
Inner shell interaction	BETAp_HARD_INNER_SHELL	3
Annihilation	BETAp_ANNIHILATION	4
Delta interaction	BETAp_DELTA	5
Soft interaction	BETAp_SOFT_INTERACTION	6

Table 6: Positron interaction indexes.

4.5 Atomic electron shells

The atomic electron shells labels are designed following the indexes used at PENELOPE code. These indexes are summarised at the table 7, which has been extracted from the PENELOPE manual [3].

Label	Shell	Label	Shell	Label	Shell
1	K ($1s_{1/2}$)	11	N2 ($4p_{1/2}$)	21	O5 ($5d_{5/2}$)
2	L1 ($2s_{1/2}$)	12	N3 ($4p_{3/2}$)	22	O6 ($5f_{5/2}$)
3	L2 ($2p_{1/2}$)	13	N4 ($4d_{3/2}$)	23	O7 ($5f_{7/2}$)
4	L3 ($2p_{3/2}$)	14	N5 ($4d_{5/2}$)	24	P1 ($6s_{1/2}$)
5	M1 ($3s_{1/2}$)	15	N6 ($4f_{5/2}$)	25	P2 ($6p_{1/2}$)
6	M2 ($3p_{1/2}$)	16	N7 ($4f_{7/2}$)	26	P3 ($6p_{3/2}$)
7	M3 ($3p_{3/2}$)	17	O1 ($5s_{1/2}$)	27	P4 ($6d_{3/2}$)
8	M4 ($3d_{3/2}$)	18	O2 ($5p_{1/2}$)	28	P5 ($6d_{5/2}$)
9	M5 ($3d_{5/2}$)	19	O3 ($5p_{3/2}$)	29	Q1 ($7s_{1/2}$)
10	N1 ($4s_{1/2}$)	20	O4 ($5d_{3/2}$)	30	outer shells

Table 7: Atomic electron shells indexes.

5 Framework usage

This section, address how to use the main program provided within the PenRed package. This one allows the user to make simulations without programming. Instead, the user will use a configuration file which structure is explained following.

First, all configuration files must follow the PenRed's internal data library format, which has been explained in section 3. Secondly, all configuration files must include some mandatory sections to specify the simulation characteristics. These sections are, sources, geometry, materials, tallies and global parameters, and all of them are described in the following sections.

5.1 Materials

The material properties, such as cross sections, density, components, etc., are defined in material files. These files can be generated at runtime by specifying the composition in the configuration, as will be discussed, or by using an external program. Additionally, since the files follow the same structure as those used in the original PENELOPE FORTRAN code, material files can also be generated using the tools and databases provided in the original PENELOPE package, which are included in the PenRed package. The material database is located in the folder

```
dataBases/penmaterials
```

where both, the program code to build materials and the database are stored. That code is a literal translation to C++ of the original FORTRAN code used by PENELOPE to create the materials. Thus, outputs of both C++ and FORTRAN versions must be perfectly equivalent and usable as input for PenRed and PENELOPE. To compile the material build program, the user must compile the "material.cpp" file. For instance, the following line can be used to compile this code using the C++ GNU GCC compiler,

```
g++ -o createMat material.cpp
```

where *createMat* is the executable name. Once the code has been compiled, execute it in the same folder where the database is (folder *pdfiles*) and follow the program instructions to create the material. As in the FORTRAN code, a set of predefined materials can be build via numerical codes, which are specified in the appendix A, in the tables 10 and 11.

5.1.1 Configuration

To specify a material in the configuration file, the user must adhere to the pattern shown in code 3. Here, *materials* denotes a constant text, *material-name* serves as a user-selected text identifier or name, and *parameter/path* and *value* represent the parameters and values to be specified for that material.

```
materials/material-name/parameter/path value
```

Code 3: Material configuration pattern

For instance, the code 4 shows the copper material configuration used in the first example provided in the package.

```
materials/cu/number 1

materials/cu/eabs/electron 1.0e3
materials/cu/eabs/positron 1.0e3
materials/cu/eabs/gamma 1.0e3

materials/cu/C1 0.05
materials/cu/C2 0.05

materials/cu/WCC 1.0e3
materials/cu/WCR 1.0e3

materials/cu/filename "Cu.mat"
```

Code 4: Complete material configuration example

As shown, the parameters required for each material specification are analogous to those needed by the PENELOPE FORTRAN version. Initially, the material *number* is employed to identify the material in the geometry system. This number must be greater than 0, with material 0 representing void, and lesser or equal to the total number of materials.

Other parameters are optional and can be set automatically by penRed default values, with only the material number, and the filename if the material composition is not provided, being mandatory parameters. However, it is advisable to select these parameters based on the simulation characteristics. In the following sections, all available configuration parameters for materials will be described.

5.1.1.1 Composition

A material can be automatically created using the *elements* section. In this section, each path defines an element's atomic mass, and the corresponding value specifies the fraction by mass of this element in the material composition. This pattern is illustrated in Code 5.

```
materials/material-name/elements/Z fraction-by-mass-weight
```

Code 5: Material composition configuration pattern

For example, to define the PMMA material in the *8-fake-chamber* examples, the following configuration can be used:

```
materials/chamberTop/number 3

materials/chamberTop/elements/1 0.080538
materials/chamberTop/elements/6 0.599848
materials/chamberTop/elements/8 0.319614
materials/chamberTop/density 1.19500000E+00
materials/chamberTop/force-creation true

materials/chamberTop/eabs/electron 1.0e3
```

```
materials/chamberTop/eabs/positron 1.0e3
materials/chamberTop/eabs/gamma 1.0e3

materials/chamberTop/C1 0.00
materials/chamberTop/C2 0.00

materials/chamberTop/WCC 0.0
materials/chamberTop/WCR 0.0

materials/chamberTop/filename "pmma.mat"
```

Code 6: Material composition configuration pattern

In Code 6, the parameters for material file creation, in addition to the already discussed *elements* subsection, are:

- **density**: Defines the material density in g/cm^3.
- **force-creation**: This is an optional parameter that enables/disables forcing the creation of the material file. If the file already exists, the program will skip its creation and process the existing one instead, as default behaviour.
- **filename**: This parameter is mandatory if the **elements** section is not defined. It specifies the material file where the material data will be read, or written if it is created during execution.

5.1.1.2 Composition databases

Instead of manually specifying the composition of a material, users can load a material composition from the available databases. To view the available databases and included materials, use the *compositionsDB* utility (Section 8.10).

To specify a material in a database, the following values must be specified:

- **DB**: Name of the database.
- **DB-material**: Name of the DB material.

For example, the following code generate a material from the *ICRP_AF* database:

```
materials/humeri/number 3

materials/humeri/DB "ICRP_AF"
materials/humeri/DB-material "Humeri_upper_cortical"
materials/humeri/filename "humeri.mat"
```

Code 7: Material composition DB configuration pattern

5.1.1.3 Absorption energy

Absorption energy can be provided on both a global and local scale for each particle. When a particle reaches its absorption energy threshold, it will be absorbed within its respective material. Similarly, if secondary particles are generated with a energy below this threshold, they will be considered locally absorbed and will not be simulated. To set a default absorption energy value for all materials, use the following path:

```
material-eabs/particle-name value
```

Alternatively, to apply the absorption energy to a specific material, use the following path:

```
materials/material-name/eabs/particle-name value
```

Here, *material-name* represents the user-defined name of the material, *particle-name* (Table 3) refers to the particle to which the material-specific absorption energy will be applied, and *value* denotes the absorption energy in electronvolts (eV). If no global or local values are specified, a default value of 1 keV will be used.

5.1.1.4 Limiting range

Instead of absorption energy, the minimum range to simulate each particle type can be defined. Thus, a particle whose energy corresponds to a smaller *range* than the one specified will be absorbed. The interpretation of the range depends on the particle:

- **gamma**: The range is defined as the particle mean free path (inverse attenuation coefficient), in cm.
- **electrons and positrons**: The range is interpreted as the Continuous Slowing Down Approximation (CSDA) range, in cm.

To specify this parameter, the following pattern must be used in the corresponding material configuration section,

```
materials/material-name/range/particle-name value
```

where *range* is a constant key and *particle-name* must be replaced by the particle name to which the range constraint is to be applied. Finally, the *value* specify the limiting range in cm.

In addition, global particle limiting ranges can be defined for all materials. These can be specified following the pattern,

```
material-ranges/particle-name value
```

outside the material section. There, *material-ranges* is a constant key, *particle-name* is the particle name to apply the limiting range and *value* specify the limiting range in cm.

Notice that default range values can be overwritten by local material ranges. For example, the configuration shown in the Code 8 sets the the default range for electron and gamma to 0.1 cm for all materials. Then, a *infinite* absorption energy is set to positrons to avoid their transport. Finally, in the *water* material, the default range value is overwritten for gammas to 0.05 cm. Because the electron transport parameters are not specified, all materials use the default ones.

```

material-ranges/electrons 0.1
material-ranges/gamma 0.1

materials/cu/number 1
materials/cu/eabs/positron 1.0e35
materials/cu/filename "Cu.mat"

materials/Fe/number 2
materials/Fe/eabs/positron 1.0e35
```

```
materials/Fe/filename "Fe.mat"

materials/water/number 3
materials/water/range/gamma 0.05
materials/water/eabs/positron 1.0e35
materials/water/filename "H2O.mat"
```
Code 8: Configuration example with different absorption limits

5.1.1.5 Electron and positron transport parameters

Next, the $C1$, $C2$, WCC and WCR parameters are used to control the class II transport of electrons and positrons. To be able to select the appropriate parameters, the corresponding description follows, which has been extracted from the PENELOPE manual [3], where are further explained,

- **C1**: Average angular deflection, $C1 \approx 1- < cos\theta >$, produced by multiple elastic scattering along a path length equal to the mean free path between consecutive hard elastic events. The maximum allowed value is 0.2, but a value of 0.05 is usually adequate.
- **C2**: Maximum average fractional energy loss between consecutive hard elastic events. As $C1$, the maximum allowed value is 0.2 and a value of about 0.05 is, usually, adequate.
- **WCC**: Cutoff energy loss, in eV, for hard inelastic collisions.
- **WCR**: Cutoff energy loss, in eV, for hard bremsstrahlung emission.

The election of these parameters determine the simulation trade off between speed and accuracy. For better accuracy, $C1$ and $C2$ should have small values (0.01). Larger values makes the simulation faster but less accurate. On the other hand, the cutoff energies WCC and WCR speed up the simulation when using larger values, but if these are too large, the tallied energy distributions could be distorted. To avoid this effect, the values of WCC and WCR should be lesser than the bin width used to tally the energy distributions. Finally, to ensure the reliability of the simulation, the user must ensure that the number of steps, or random hinges, per primary track is "statistically sufficient", which should be achieved with more than 10 steps.

If the mentioned parameters are not specified by the user, PenRed will set the parameters automatically.

5.1.1.6 Filename

Finally, the $filename$ parameter specify the relative path to the material file. The path to the material file must be enclosed by quotes to be identified as text.

Notice that the configuration shown in the code 4 must be repeated for each used material, changing the corresponding *name* and parameter values.

5.2 Particle sources

Particle sources in penRed are responsible for generating particles to be simulated. These particles can be either primary (originating directly from the source) or secondary (produced by interactions during the simulation). Multiple particle sources can be defined within the same configuration file, and they will be processed sequentially.

To configure a particle source parameter, users must adhere to the syntax structure shown in Code 9:

```
sources/type/name/parameter/path value
```

Code 9: Source configuration parameters

Here, the field *type* must be set to either:

- *generic*: For sampling generic particle states
- *polarized*: For sampling gamma rays with explicit polarization states

The *name* field assigns a unique identifier to the source, while *parameter/path* specifies the target configuration parameter, and *value* defines its setting.

The initial state of simulated particles is determined through two distinct approaches to state sampling. First, the *generic* samplers focus on a single aspect of the particle state:

- *spatial*: Defines the particle's origin.
- *direction*: Sets the initial direction.
- *energy*: Samples the initial particle kinetic energy.
- *time*: Samples the initial particle age.

The second category consists of *specific* samplers, which provide greater flexibility in particle state initialization. Unlike generic samplers that handle individual state components, specific samplers can:

- Define the complete particle state (overriding generic samplers)
- Combine with generic samplers to control specific state parameters

Important Note: The distinction between *generic/specific* samplers (which govern state initialization methods) should not be confused with *generic/polarized* sources (which determine fundamental particle properties).

5.2.1 Generic Sampling Sources

The Code 10 illustrates common mandatory and optional parameters for a generic samplers-based source named *source1*:

```
sources/generic/source1/nhist 1.0e6          (mandatory)
sources/generic/source1/kpar "gamma"         (mandatory)
sources/generic/source1/record-time true     (optional)
sources/generic/source1/source-body 1        (optional)
sources/generic/source1/source-material 1    (optional)
```

Code 10: Generic sampling based source configuration parameters

This configuration generates a million primary gamma particles and enables time recording. The parameters are defined as follows:

- *nhist*: Number of particles to simulate.
- *kpar*: Particle type ("gamma", "electron", or "positron").
- *record-time*: Boolean flag to enable/disable time tracking.

Additionally, users may restrict particle generation to a specific geometry body or material using the optional parameters *source-body* or *source-material*, respectively. Both require an integer index referencing the desired body or material. Note: Only one restriction (body or material) can be active per source. If a sampled particle originates outside the specified region, the spatial sampling repeats until the condition is satisfied.

Previous parameters alone do not define the initial particle state (position, direction, energy, and time). For this, a sampler for each state property must be defined using the keywords:

- *spatial*: Sets particle origin.
- *direction*: Sets initial direction sampling.
- *energy*: Sample the initial particle kinetic energy.
- *time*: Samples the initial particle age (Optional).

All samplers require the *type* parameter to select the sampling method or class. For example, a mono-energetic energy sampler is specified as:

```
sources/generic/source1/energy/type "MONOENERGETIC"
```

Code 11: Mono-energetic sampler configuration

In addition, each sampling type could requires additional specific parameters. For example, to set 30 keV for a *"MONOENERGETIC"* energy source, the energy (in eV) is set via the *energy* parameter:

```
sources/generic/source1/energy/type "MONOENERGETIC"
sources/generic/source1/energy/energy 3.0e7
```

Code 12: Energy value specification

The other samplers are configured analogously but using their own parameters. A source without specific sampler (a pure generic sampler-based source) requires **spatial**, **direction**, and **energy** samplers (**time** is optional). Below is a working example (comments start with #):

```
#-------------------------
#  Source 1
##########################

sources/generic/source1/nhist 1.0e6
sources/generic/source1/kpar "gamma"
sources/generic/source1/record-time true

# Directional sampling
##########################

sources/generic/source1/direction/type "CONE"


# Set theta
sources/generic/source1/direction/theta 0.0

# Set phi
sources/generic/source1/direction/phi 0.0

# Set aperture
sources/generic/source1/direction/alpha 5.0

```

```

# Energy sampling
##########################

sources/generic/source1/energy/type "MONOENERGETIC"

# Set energy
sources/generic/source1/energy/energy 3.0e7


# Spatial sampling
##########################

sources/generic/source1/spatial/type "POINT"

# Set particle origin
sources/generic/source1/spatial/position/x 0.0
sources/generic/source1/spatial/position/y 0.0
sources/generic/source1/spatial/position/z -25.0
```

Code 13: Complete source configuration example

The user can find many other pre-configured sources in the *examples* folder. Available generic samplers are listed by type in the following sections.

5.2.2 Spatial source samplers

5.2.2.1 Point

The spatial point sampler needs the (x, y, z) coordinates to fill the position of generation particles. These values are specified in cm. The following is a configuration example of this kind of spatial sampling corresponding to the 1-disc example.

```
# Spatial sampling
##########################

sources/generic/source1/spatial/type "POINT"

# Set particle origin
sources/generic/source1/spatial/position/x 0.0
sources/generic/source1/spatial/position/y 0.0
sources/generic/source1/spatial/position/z -0.0001
```

Code 14: Spatial source POINT sampler

5.2.2.2 Box

This spatial sampler creates a box in which particles are generated with a uniform probability. To define the box, the user must provide the following parameters:

- **position/x,y,z**: Specify the 3D center, in cm, of the sampling box.
- **size/dx,dy,dz**: Sets the box size in each axis, in cm.

An example of configuration file is shown below.

```
# Spatial sampling
##########################

sources/generic/source1/spatial/type "BOX"

```

```

# Set box size
sources/generic/source1/spatial/size/dx 0.75
sources/generic/source1/spatial/size/dy 0.75
sources/generic/source1/spatial/size/dz 1.25

# Set particle origin
sources/generic/source1/spatial/position/x 0.0
sources/generic/source1/spatial/position/y 0.0
sources/generic/source1/spatial/position/z −0.0001
```

Code 15: Spatial source BOX sampler

5.2.2.3 Cylindrical shell

This spatial sampler generates particles within a cylindrical shell. It is defined by subtracting a smaller cylinder from a bigger one where it is included. The sampling region corresponds to the shaded area in the Figure 11.

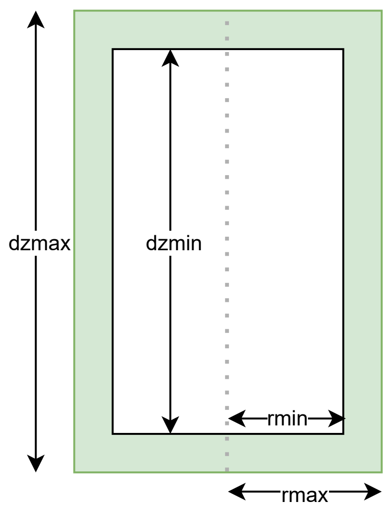


Figure 11: Cylindrical shell sampling region and values to define.

Notice that the center of both cylinders coincide. The parameters to define the region are listed following:

- **type**: Must be set to "CYLINDER".
- **size/rmin**: Inner cylinder radius in cm.
- **size/rmax**: External cylinder radius in cm.
- **size/dzmin**: Inner cylinder height in cm.
- **size/dzmax**: External cylinder height in cm
- **position/x**: X coordinate of the cylinders center in cm.
- **position/y**: Y coordinate of the cylinders center in cm.
- **position/z**: Z coordinate of the cylinders center in cm.

For example, the Code 16 shows the configuration to sample particles in a thin layer of radioactive paint on a brachytherapy seed located at the origin.

```
# Spatial sampling
##########################

sources/generic/source1/spatial/type "CYLINDER"

# Set cyl size
sources/generic/source1/spatial/size/rmin 0.025
sources/generic/source1/spatial/size/rmax 0.025175
sources/generic/source1/spatial/size/dzmin 0.28
sources/generic/source1/spatial/size/dzmax 0.295

# Set position
sources/generic/source1/spatial/position/x 0.0
sources/generic/source1/spatial/position/y 0.0
sources/generic/source1/spatial/position/z 0.0
```

Code 16: Spatial source CYLINDER sampler

5.2.2.4 Measure (1D,2D,3D)

This family of samplers uses 1D, 2D, or 3D distributions to determine the spatial position of generated particles. These distributions can be obtained directly from the *Detection Spatial Distribution* tally (Section 5.4.17).

Regardless of the number of dimensions used, the samplers share the following common parameters:

- **filename**: Specifies the path to the file containing the spatial distribution.
- **dx**: Specifies a displacement, in cm, applied to the distribution along the X axis. This value is optional and defaults to 0.
- **dy**: Specifies a displacement, in cm, applied to the distribution along the Y axis. This value is optional and defaults to 0.
- **dz**: Specifies a displacement, in cm, applied to the distribution along the Z axis. This value is optional and defaults to 0.

The dimension-specific parameters are as follows:

- **3D**:
  - **Type**: The tally type must be set to "3D_MEASURE".
- **2D**:
  - **Type**: The tally type must be set to "2D_MEASURE".
  - **plane**: Specifies the plane to be sampled, using one of the following values: "xy", "xz", "yz".
  - **constant-coordinate**: Specifies the value of the constant coordinate, in cm, according to the selected plane. For example, if the "xy" plane is selected, this parameter fixes the coordinate along the Z axis.
- **1D**:

- **Type**: The tally type must be set to 1D_MEASURE".
- **axis**: Specifies which axis will be sampled. The allowed values are "x", "y", and "z".
- **x**: Sets the constant value along the X axis. Defaults to 0.
- **y**: Sets the constant value along the Y axis. Defaults to 0.
- **z**: Sets the constant value along the Z axis. Defaults to 0.

An example configuration is shown in code 17 for the 2D case.

```

sources/generic/source1/spatial/type "2D_MEASURE"

sources/generic/source1/spatial/filename "source.dat"

sources/generic/source1/spatial/constant-coordinate 0.0
sources/generic/source1/spatial/plane "xy"

sources/generic/source1/spatial/dx 0.0
sources/generic/source1/spatial/dy 0.0
sources/generic/source1/spatial/dz 0.0
```

Code 17: 2D Measure sampler configuration example

5.2.3 Direction samplers

5.2.3.1 Cone

Particles generated by the source are sampled within a conical beam. In this sampling process, the aperture of the cone must be specified to ensure that the particles' directions are uniformly distributed within the corresponding solid angle. The required parameters are the polar angle (*theta*), the azimuthal angle (*phi*), and the semi-aperture angle (*alpha*), which together define the direction and extent of the solid angle. Note that all angles are expected to be provided in degrees.

To simulate a monodirectional source, the semi-aperture must be set to $alpha = 0$. In contrast, to achieve an isotropic source, the semi-aperture should be set to $alpha$=180.

Below is an example of the directional sampling configuration from the 2-plane example file, which corresponds to a source aiming at the Z-axis with a semi-aperture of 5 degrees.

```

# Directional sampling
###########################

sources/generic/source1/direction/type "CONE"


# Set theta
sources/generic/source1/direction/theta 0.0

# Set phi
sources/generic/source1/direction/phi 0.0

# Set oberture (alpha)
sources/generic/source1/direction/alpha 5.0
```

Code 18: Direction CONE sampler

5.2.3.2 Sphere

In this case, the direction of the generated particles is sampled within a section of a sphere. The required parameters for this directional sampler are:

- The **direction** components along each axis (u,v,w).
- The polar aperture, defined by the minimum (*theta0*) and maximum (*theta1*) values of the polar angle, in degrees. Both values must be contained in the interval $[0, 180]$.
- The azimuthal aperture, defined by the minimum value of the azimuthal angle (*phi0*) and its angular increment (*dphi*).

The code snippet below (Code 19) demonstrates an example of a sphere section pointing in the +Z direction with a semi-aperture of 30 degrees across the entire azimuthal interval:

```

# Directional sampling
#########################

sources/generic/source1/direction/type "SOLID_ANGLE"


# Set direction vector
sources/generic/source1/direction/u 0.0
sources/generic/source1/direction/v 0.0
sources/generic/source1/direction/w 1.0

# Set polar overture

sources/generic/source1/direction/theta0   0.0
sources/generic/source1/direction/theta1  30.0


# Set azimutal overture
sources/generic/source1/direction/phi0 0.0
sources/generic/source1/direction/dphi 360.0
```

Code 19: Direction SOLID_ANGLE sampler

5.2.4 Energy samplers

5.2.4.1 Monoenergetic

This energy sampler generates a monoenergetic beam of particles with the specified energy in eV. The only required parameter is the *energy* value, which must be a positive number. An example of its usage can be found in the configuration file shown below. These lines correspond to the example 4-x-ray-tube, file `tube.in`.

```

# Energy sampling
#########################

sources/generic/source1/energy/type "MONOENERGETIC"

# Set energy
sources/generic/source1/energy/energy 1.5e5
```

Code 20: Energy samplers MONOENERGETIC

5.2.4.2 Intervals

Intervals sampler generates energies within the specified spectral lines according to the probability assigned to each one. To define the spectral lines in the configuration file, first, the user must specify the number of intervals, *ninterval*, using a integer value. Then, the energy range of each interval, $[lowE, topE]$, are specified as follows:

- The array named **lowE** contains low energy boundaries of all energy intervals, in eV.
- Top energy limits are defined in the **topE** array, in eV.
- The probabilities of each interval are stored in a third array named **probabilities**.

Therefore, the first element in *lowE* and *topE* arrays define, respectively, the low and upper limits for the first interval or spectral line, and the first position of the *probabilities* array the corresponding probability. This is repeated for each defined interval.

The probabilities are not required to be normalized. The following code shows a configuration corresponding to the 3-detector example, file `detector1.in`.

```

# Energy sampling
##########################

sources/generic/source1/energy/type "INTERVALS"


# Set intervals
sources/generic/source1/energy/nintervals 2


# Set energies
sources/generic/source1/energy/lowE [1.17e6,1.33e6]
sources/generic/source1/energy/topE [1.17e6,1.33e6]

# Set probabilities
sources/generic/source1/energy/probabilities [50.0,50.0]
```

Code 21: Energy samplers INTERVALS

5.2.4.3 File spectrum

This energy sampler has been implemented to provide compatibility between the PenEasy [7] (v.2020-03-25) energy spectrum format and PenRed. Although this one is described in the PenEasy manual [8], it is also described following for the user convenience.

The format consists on a piecewise function where each entry in the spectrum contains two numbers. The first one, specify the starting energy of a channel, and the second its unnormalized probability. The probabilities must be non-negative and don't require to be normalized to unity. The program interprets a negative probability as the end of the spectrum. For example, a valid spectrum format is shown in the code 22, where the character # is interpreted as a comment. Notice that the spectrum file must contain only comments or valid spectrum values until the end of the spectrum. For example, blank lines are not allowed. However, all the text beyond the end of the spectrum is ignored by the sampler.

```
#Spectrum     (character '#' is interpreted as a comment)
1.20e3  10    1st channel: [1.20,1.61]keV
1.61e3  0.0   2nd channel: [1.61,5.00]keV, no emissions
5.00e3  20    monoenergetic line at 5.00 keV
```

```
5.00e3 0      3rd channel: [5.00,6.24]keV, no emissions
6.24e3 33.3   4th channel: [6.24,7.32]keV
7.32e3 5.02   5th channel: [7.32,8.00]keV
8.00e3 10.2   6th channel: [8.00,9.76]keV
9.76e3 15.2   7th channel: [9.76,10.0]keV
10.0e3 0.0    8th channel: [10.0,15.0]keV, no emissions
15.0e3 20     monoenergetic line at 15.0 keV
15.0e3 -1     end of the spectrum
```

Code 22: File energy spectrum format

Notice that the text after both numerical values for each line is ignored by the program. In the previous code, its only added to describe the defined spectrum.

This sampler only requires two configuration values. Like other samplers, the first one is the *type*, which must be set to *"FILE_SPECTRUM"*. The other is the spectrum filename which is introduced as a string, as is shown in the code 23.

```

# Energy sampling
##########################

sources/generic/source1/energy/type "FILE_SPECTRUM"
sources/generic/source1/energy/filename "spectrum.spc"
```

Code 23: File spectrum configuration

5.2.5 Time samplers

5.2.5.1 Decay

This sampler randomly generates event times based on the exponential decay characteristic of a radioactive element. The sampled time is constrained by user-specified initial and final times, and the exponential decay follows the radioisotope's half-life. The parameters required are:

- **halfLife**: The half-life of the radioactive element, in seconds. Must be a positive value.
- **time/time0**: The initial time for the decay interval, in seconds. Must be a positive value.
- **time/time1**: The final time for the decay interval, in seconds. Must be a positive value.

The decay follows an exponential distribution, parameterized by the specified half-life, which governs the probability distribution of event occurrences over time.

5.2.6 Specific samplers

The behavior of specific samplers with respect to generic samplers (**spatial**, **direction**, **energy**, and **time**) varies by implementation. Certain specific samplers, such as the Phase Space File sampler, completely define the particle state and ignore any configured generic samplers. Others, like the Computed Tomography sampler, work in conjunction with generic samplers to determine specific state parameters while handling the remaining aspects internally.

Specific samplers exhibit similar variability in handling particle types. The Phase Space File sampler, for example, dynamically determines both particle type and state from an external file during operation. In contrast, other specific samplers maintain fixed particle types that must be explicitly specified through standard configuration, as demonstrated in Code 10.

The following sections provide detailed descriptions of all available specific samplers and their respective implementations.

5.2.6.1 Phase Space File (PSF)

The Phase Space File (PSF) source reads particle states from files generated by the `PSF` tally, as demonstrated in the 5-accelerator-2 example (Code 24). Configuration requires these parameters:

Core Parameters:

- `type`: Must be set to **PSF**.
- `filename`: Path to the phase space file.
- `Emax`: Maximum particle energy (in eV) stored in the PSF. If the specified energy doesn't cover the whole energy range in the PSF, the program will produce unexpected results.

Variance Reduction:

The `wght-window` parameter defines weight boundaries [w_{min}, w_{max}] to apply Russian Roulette (for weights below minimum) and splitting (for weights above maximum), while `split` sets the splitting multiplier. Therefore, particles whose weight is within the window remain unchanged.

Spatial Transforms (applied in order):

- *rotation*: Specified through ZYZ Euler angles (`omega`, `theta`, `phi`), in degrees.
- *translation*: Defined by `dx`, `dy`, `dz` offsets, in cm.

```
#  Source 1
##########################

sources/generic/source-psf/nhist 1.0e6
sources/generic/source-psf/specific/type "PSF"
sources/generic/source-psf/specific/filename "psf-merged.dat"
sources/generic/source-psf/specific/Emax 7e6

sources/generic/source-psf/specific/wght-window [5e-4, 1e-3]
sources/generic/source-psf/specific/nsplit 10

sources/generic/source-psf/specific/rotation/omega 0
sources/generic/source-psf/specific/rotation/theta 90
sources/generic/source-psf/specific/rotation/phi 0

sources/generic/source-psf/specific/translation/dx 0
sources/generic/source-psf/specific/translation/dy 0
sources/generic/source-psf/specific/translation/dz 10
```

Code 24: Phase Space File Source

5.2.6.2 Computed Tomography (CT)

This source is designed to emit particles along a circumference, describing the movement of a CT scanner movement. To configure it, the source type must be set to "CT" in the configuration file. Particle sampling can be performed in two different ways:

- **Specific source**: particles are sampled from a previously generated Phase Space File (PSF). This source internally uses the PSF source (section 5.2.6.1), and inherits its parameters already defined: *filename*, *Emax*, *wght-window* and *nsplit*.

  Additionally, the PSF spatial transforms rotation and translation can be applied to this source.

- **Generic source**: particles are sampled based on the directional, energy, and spatial distributions described in section 5.2.1. Since CT scans typically emit secondary particles, the configuration file must include the parameter *nSecondaries*. This specifies the number of secondary particles sampled per history before incrementing the history counter.

After sampling (via PSF or generic methods), the CT source requires three parameter groups:

- **Angular information**:
  - `phi-ini` and `phi-end` (in degrees): specify the initial and final angular position of the source. Note that `phi-ini` = 0° corresponds to the source located on x-axis, with emission directed to the negative x-axis, as it is shown in Figure 12, and that `phi-end` must be ≥ `phi-ini`. Therefore, the source configuration must be configured following these requirements.
  - `nProjections`: number of projections during the CT rotation.
- **Position information**:
  - `[CTx0, CTy0, CTz0]`: coordinates, in cm, of the CT rotation center (ISOCENTER).
  - `rad`: CT radious, source-isocenter distance (cm).
- **Time information**:
  - `tmin`: initial time stamp for the first projection (s).
  - `dt`: time interval between projections (s).

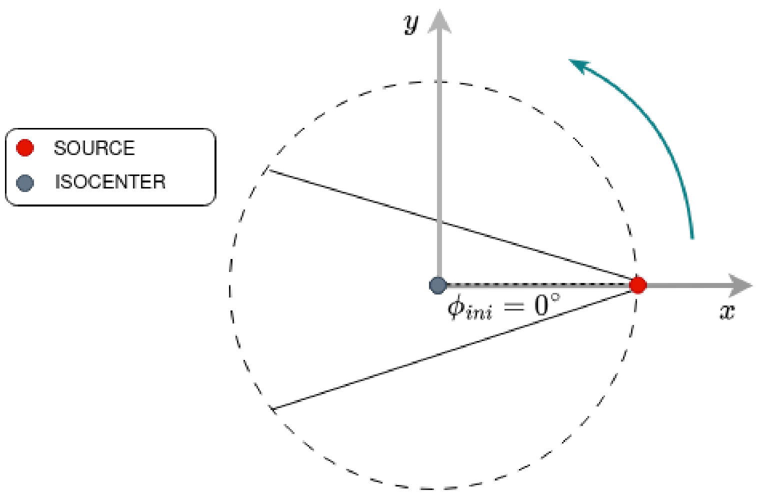


Figure 12: Schematic representation of the CT source emission

To ensure particles are aimed toward the CT center, the sampled particles' initial direction must point along the -X axis.

An example of CT source using PSF generated previously is shown in 25, while an example in which a generic source is used is shown in the code 26.

```
#  Source 1
###########################

sources/generic/source-ct/nhist 1.5e8
sources/generic/source-ct/specific/type "CT"
sources/generic/source-ct/specific/psf/filename "psf-merged.dat"
sources/generic/source-ct/specific/psf/Emax 1.0e5

sources/generic/source-ct/specific/psf/wght-window [5e-4, 1e-3]
sources/generic/source-ct/specific/psf/nsplit 100

# Angular information
#################################
sources/generic/source-ct/specific/phi-ini 0
sources/generic/source-ct/specific/phi-end 360
sources/generic/source-ct/specific/nProjections 360


# Position information
#################################
sources/generic/source-ct/specific/CTx0 0
sources/generic/source-ct/specific/CTy0 0
sources/generic/source-ct/specific/CTz0 0
sources/generic/source-ct/specific/rad 52.0

# Time information
#################################
sources/generic/source-ct/specific/tmin 0.0
sources/generic/source-ct/specific/dt 120.0
```

Code 25: PSF Computed Tomography Source

```
#  Source 1
##########################

sources/generic/source-ct/nhist 1e8
sources/generic/source-ct/specific/type "CT"

sources/generic/source-ct/specific/nSecondaries 80

sources/generic/source-ct/kpar "gamma"


# Directional sampling
##########################
sources/generic/source-ct/direction/type "CONE"

# Set theta
sources/generic/source-ct/direction/theta 90.0

# Set phi
sources/generic/source-ct/direction/phi 180.0

# Set oberture (alpha)
sources/generic/source-ct/direction/alpha 11


# Energy sampling
##########################

sources/generic/source-ct/energy/type "MONOENERGETIC"

# Set energy
sources/generic/source-ct/energy/energy 1.20e5

# Spatial sampling
##########################

sources/generic/source-ct/spatial/type "POINT"

# Set particle origin
sources/generic/source-ct/spatial/position/x 0.0
sources/generic/source-ct/spatial/position/y 0.0
sources/generic/source-ct/spatial/position/z 0.0


# Angular information to do
###############################
sources/generic/source-ct/specific/phi-ini 0
sources/generic/source-ct/specific/phi-end 360
sources/generic/source-ct/specific/nProjections 360



# Position information
###############################
sources/generic/source-ct/specific/CTx0 19.15
sources/generic/source-ct/specific/CTy0 19.15
sources/generic/source-ct/specific/CTz0 14.8
sources/generic/source-ct/specific/rad 106.0



# Time information
```

```
##########################################

sources/generic/source-ct/specific/tmin 0.0
sources/generic/source-ct/specific/dt 120.0
```
Code 26: Generic Computed Tomography Source

5.2.6.3 PENNUC

The `PENNUC` source has been adapted from the original FORTRAN code published in [9] and developed by CIEMAT, Laboratoire National Henri Becquerel, and Universitat de Barcelona. This tool enables simulation of radioactive nuclide decay and subsequent use of the resulting particles in generic simulations. The decay characteristics for each nuclide, provided in the `PENNUC` format, can be obtained from a public database[4].

A complete example demonstrating the use of the `PENNUC` source can be found in the directory:

`examples/quadrics/3-detector-2/`

For spatial sampling, the `PENNUC` source requires a generic spatial sampler, such as the box sampler (Section 5.2.2). Additionally, the source must be configured with the parameters shown in Code 27 and explained below:

- **type**: The source type must be set to `PENNUC`
- **nucleide_filename**: Specifies the file containing decay data downloaded from the nucleide database[5]
- **age**: Optional parameter to enable or disable age recording (default: `false`)
- **source-material**: Optional parameter to restricts particle generation to a specific material, resampling the position if necessary
- **pennuclog_filename**: Optional parameter to specify the log file (default: `pennuc.dat`)

`PENNUC` requires atomic and relaxation information from the material database. By default, the database is embedded in the executable during compilation and the necessary files (`pdatconf.p14` and `pdrelax.p11`) will be generated automatically. If this compilation option is disabled by the user, both files must be manually copied to the simulation directory.

Note that `PENNUC` always samples particle directions isotropically; only the position and the initial time can be controlled through generic samplers.

```

# PENNUC sampling
##########################

sources/generic/source1/specific/type "PENNUC"                          (mandatory)
sources/generic/source1/specific/nucleide_filename "Co-60.nuc"  (mandatory)
sources/generic/source1/specific/age true                                 (optional)
sources/generic/source1/specific/source-material 1                        (optional)
```
Code 27: PENNUC source configuration

[4]http://www.lnhb.fr/home/nuclear-data/nuclear-data-table/
[5]http://www.lnhb.fr/home/nuclear-data/nuclear-data-table/

5.2.6.4 Brachytherapy seeds source

This source simulates brachytherapy treatments by reading seed positions from patient DICOM plans. To use it, the user needs:

1. A pre-generated phase space file (PSF) characterizing the seed emission.
2. A **DICOM based geometry** with seed positioning through DICOM plan data.

In addition, the seed PSF must meet these geometric conditions:

- Seed center positioned at origin (0,0,0).
- Seed orientation aligned with the +Z axis

These conditions ensure a correct transformation during simulation. If the original PSF does not comply, corrective transformations can be applied through configuration of the PSF parameters.

Configuration Parameters:

- **type**: Must be set to `"BRACHY"`.
- **psf** subsection: Configures the internal PSF source (see Section 5.2.6.1).
- **seed-rotation**: Enables/disables per-seed rotation (default: `true`). When enabled, an extra rotation will be applied to the generated particles according to each seed orientation.

An example of a complete configuration is shown in the code 28.

```

#-------------------------------
#  Source 1
##############################

sources/generic/source-psf/nhist 5.0e7
sources/generic/source-psf/specific/type "BRACHY"
sources/generic/source-psf/specific/psf/filename "psf-merged.dat"
sources/generic/source-psf/specific/psf/Emax 1.40e6

sources/generic/source-psf/specific/psf/wght-window [5e-4, 1e-3]
sources/generic/source-psf/specific/psf/nsplit 10

sources/generic/source-psf/specific/seed-rotation false
```

Code 28: Brachytherapy source

Notice that the DICOM is not specified in the source configuration. This is because the DICOM information is taken directly from the geometry. Therefore, this source **is only intended to be used with DICOM based geometries**. In addition, if no DICOM plan file is provided in the specified DICOM, or the plan has not any seed, a warning will be shown in the log and the particles will be created at the $(0, 0, 0)$ cm.

5.3 Geometries

Each simulation only allows to use a single geometry which configuration parameters are specified by the prefix *geometry* to be correctly identified. Like sources, geometries require to specify its type, and other specific parameters.

5.3.1 Geometry visualization

All geometries compatible with the PenRed framework can be visualized using the viewer provided in the following repository,

`https://github.com/PenRed/GeometryViewer`

which includes the documentation to install and use the viewer.

5.3.2 Quadric

This geometry type is used in all the PENELOPE based examples which have been translated to be reproduced with PenRed. For instance, the code 29 shows the geometry configuration for the first simulation example (1-disc). The geometries for this type are defined in a external file following as is explained in the PENELOPE manual [3] in the Chapter 6 **Constructive quadric geometry**. Moreover, PenRed is compatible with the PENELOPE geometry files, allowing to use them directly from PENELOPE simulations.

```
geometry/type "PEN_QUADRIC"
geometry/input-file "disc.geo"
geometry/processed-geo-file "report.geo"

geometry/dsmax/1 1.0e-4
geometry/kdet/1  1
```

Code 29: Complete geometry configuration example

As other configuration examples, the type is specified by the *type* parameter. Then, the *input-file* parameter specify the relative path to the file where the geometry has been defined. The other parameters *processed-geo-file*, *dsmax* and *kdet* are optional and described following. The first one, *processed-geo-file*, specify a file to generate a geometry report. Secondly, *dsmax* specify the maximum distance allowed for electrons and positrons to jump when class II transport is active. This one is used in thin regions to ensure a minimum number of steps in the specified body. Notice that the body where the *dsmax* is applied is specified via its alias in the parameter path, `geomtry/dsmax/*alias*`. For instance, in the previous code, the *dsmax* is applied to the body with the alias "1". Finally, the parameter *kdet* specify the detector identifier which the body belongs. Again, the body is specified using its alias in the parameter path, `geometry/kdet/*alias*`.

Optionally, it is possible to specify local absorption energies for any body and particle. In the code 30 we shown an example where the local energy absorption for each particle is set to 10 keV for the body with alias “2”. Notice that the most restrictive absorption energy is applied, either the assigned to the corresponding material or the one assigned to the body.

```
geometry/eabs/2/electron   1.0e4
geometry/eabs/2/gamma      1.0e4
geometry/eabs/2/positron   1.0e4
```

Code 30: Configuration of geometry absorption energy example

5.3.3 Triangular Mesh Surfaces

This type of geometry uses triangular mesh surfaces to define the geometry system bodies. For this purpose, PenRed defines its own format for input files. However, with the provided Blender plugin (Section 8.1), several mesh formats can be easily transformed to the PenRed one.

5.3.3.1 Input file format

In the examples folder, several triangular mesh based geometry files can be found.

```
examples/triangular_surfaces
```

These geometry files must contain the following elements, in order,

- A line with a integer which indicates the total number of bodies in the geometry.
- Then, the specification of each body in the defined geometry follows. For each one, the information to be specified is,
  - **Body metadata**: Consists of a line with the following information, in order:
    - Material number
    - Number of triangles
    - Number of vertex
    - Body name
    - Parent body name
    - Number of vertex groups

    An example is shown in the Code 31. Notice that lines beginning with # are considered as comments and are ignored when the input is processed. In all geometries, one body acts as the **world**, i.e. it has no parent and all other bodies are included inside it. To specify which body is considered as the world, its parent name must be set to **void**. Therefore, **void** can't be used as a body name.

```

#MAT  #NFACES #NVERTEX  #NAME    #PARENT NAME   #N VERTEX GROUPS
 002    40796    20400    Capsule    Phantom           5
```

Code 31: Body metadata for triangular mesh geometries.

  - **Vertex Groups:** Each vertex group consists of a first line with the group name followed by the number of vertices included in this group, along with a list of the vertex indexes, one per line. An example is illustrated in Code 32, where three vertex groups are defined.

```
# VERTEX GROUPS
#NAME  #NVERTEX
 back     0004
 0002
 0003
 0006
 0007
#NAME  #NVERTEX
 front_down     0002
 0001
 0005
#NAME  #NVERTEX
 front_up    0002
 0000
 0004
```

Code 32: Vertex group format.

Notice that this section is absent in files with zero vertex groups. An example with vertex groups can be found at the following path:
`src/lib/x-ray/source/baseAnode.geo`
Here, the vertex groups are utilized to construct an x-ray anode at runtime

- **Vertex list**: A list of all mesh vertex. Each one is specified in a single line with the following format,

  `vertex_number X Y Z`

  An example is shown in the Code 33.

```
# VERTEX LIST
# Index   (X Y Z)
0000 +0.000000E+00 −8.000000E−02 +7.566000E−02
0001 +0.000000E+00 −8.000000E−02 −1.731400E−01
0002 +2.513000E−03 −7.996000E−02 +7.566000E−02
0003 +2.513000E−03 −7.996000E−02 −1.731400E−01
0004 +5.023000E−03 −7.984200E−02 +7.566000E−02
.
.
.
```

Code 33: Body vertex list in triangular mesh geometries.

  Notice that vertex coordinates are relative to the geometry origin $(0, 0, 0)$, not to the body center.

- **Triangle list**: A list with all mesh triangles. As vertex, each one is specified in a single line, which contains the three vertex indexes that defines the triangle.
  Notice that the order of the vertex is used to calculate the direction of the triangle normal vector. The normals are assumed to point outward from the body. Most 3D design softwares uses this schema by default and provide tools both to check the geometry normals and to recalculate them.
  An input example is shown in the Code 34.

```
# FACES(triangles)
 499 001 003
 499 003 599
 599 003 005
 599 005 699
 699 005 007
 699 007 799
.
.
.
```

Code 34: Body triangle list in triangular mesh geometries.

5.3.3.2 Configuration

The mandatory configuration parameters to use this type of geometry are shown in the code 36 and are listed following,

- **type**: Must be set to **MESH_BODY**.
- **input-file**: Path to the input file with the geometry description. This must follow the format described in the section 5.3.3.1.

In addition, the `report-file` parameter can be specified to print a geometry report.

Regarding specific parameters for each body, the user can specify the `dsmax` and `kdet` parameters as is done for quadric geometries.

If vertex groups have been defined in the geometry, the vertices belonging to those groups can be transformed via the configuration file. These transformations are organized in groups to streamline their ordering and must be defined using the following pattern inside the geometry configuration section:

```
transforms/body-name/transformation-group/path value
```

where *transforms* is a constant text identifier, *body-name* identifies a body inside the geometry by its alias, *transformation-group* is a user-defined name for this transformation group, *path* depends on the parameter to be set, and *value* is the value assigned to the parameter. The possible *paths* are listed below:

- **index**: Assigns an index to this transformation group. Transformation groups are applied following a growing index order.
- **vertex-group**: Specify which vertex group is affected by the transformation group. Note that each transformation group can be applied to only one vertex group.
- **transforms**: This subsection contains all the transformations to be applied to this group. Each one must be defined using the following pattern within this subsection,

  `transformation-name`/path value

  where *transformation-name* is a user defined name for this transformation. Each transformation must contain the following elements:

  - **index**: As groups, the transformations are ordered inside each group to be applied in ascending index order.
  - **type**: Defines the transformation type. Actually, the available types are:
    - **TRANSLATION**: Apply a generic translation. Requires the following additional elemenets:
      - **ds**: Translation distance in cm.
      - **u**: X component of the translation direction.
      - **v**: Y component of the translation direction.
      - **w**: Z component of the translation direction.
    - **TRANSLATION_X**: Translation in the X axis. Only requires the *ds* parameter.
    - **TRANSLATION_Y**: Translation in the Y axis. Only requires the *ds* parameter.
    - **TRANSLATION_Z**: Translation in the Z axis. Only requires the *ds* parameter.
    - **SCALE**: Scales the distance between each vertex and the group mass center a factor specified by the parameter *factor*.
    - **SCALE_X**: Scales only in the X axis, with Y and Z remaining constant. Requires the parameter *factor*.
    - **SCALE_Y**: Scales only in the Y axis, with X and Z remaining constant. Requires the parameter *factor*.
    - **SCALE_Z**: Scales only in the Z axis, with X and Y remaining constant. Requires the parameter *factor*.

* **SCALE_XY**: Scales only in the plane XY, with Z constant. Requires the parameter *factor*.
* **SCALE_XZ**: Scales only in the plane XZ, with Y constant. Requires the parameter *factor*.
* **SCALE_YZ**: Scales only in the plane ZY, with X constant. Requires the parameter *factor*.

An example of transformations applied to a body named *anode* is shown at Code 35.

```
transforms/anode/backEnlarge/index 0
transforms/anode/backEnlarge/vertex-group "back"
transforms/anode/backEnlarge/transforms/enlargeY/type "TRANSLATION_Y"
transforms/anode/backEnlarge/transforms/enlargeY/index 0
transforms/anode/backEnlarge/transforms/enlargeY/ds -1.0
```

Code 35: Vertex group transformation configuration example.

In this kind of geometries, to check if a particle crosses a body, this one is split in several regions containing a subset of the body triangles, allowing to check only the triangles belonging to crossed regions. In turn, this regions are grouped in super-regions, following the same approach. That clustering avoids checking many triangles and saves computational time.

To control how the clustering is done, the user can specify two parameters to control how these regions are constructed, which are described following,

- **RegionElements**: Specify the objective number of triangles in each region for the specified body.
- **SuperRegions**: Specify the objective number of regions in each super-region for the specified body.

These parameters can be different for each body. For example, in the code 36, the body named `Capsule_cyl` has an objective number of triangles per region of 50 and will try to group 20 regions in each super-region. Notice that the real number of elements in each cluster could be different depending on the body structure. For example, regions with a small number of triangles are merged with boundary regions.

```

geometry/type "MESH_BODY"                    (mandatory)
geometry/input-file "seedMesh.geo"           (mandatory)
geometry/report-file "report.geo"            (optional)

geometry/RegionElements/Capsule_cyl 50       (optional)

geometry/SuperRegions/Capsule_cyl 20         (optional)
```

Code 36: Triangular mesh geometry configuration example.

5.3.3.3 Limitations

Unlike in quadric geometries, the bodies surfaces in triangular mesh geometries can't cross the surface of another body. The equivalent in quadric geometries should be consider all bodies as modules. However, the surface of two different bodies can be in contact.

5.3.4 Voxel

Voxelized geometries consist of a 3D matrix of regular prism elements with a specific material and density factor for each one. The density factor is used to specify heterogeneities in the density of the materials. For instance, a voxel with a density factor of 1.10 will have a density 10% greater than the material's nominal density (specified in the material file or the configuration). Likewise, a voxel with a density factor of 0.8 is considered to have a density 20% lower than the nominal material density. To use the nominal material density in a voxel, this factor must be set to 1.

This type of geometry accepts the following parameters:

- **type**: Must be set to **VOXEL**.
- **filename**: Specify the path to the file containing the geometry definition.
- **ascii**: Specify if the geometry to read is in ASCII (*true*) or binary (*false*) format. This parameter is optinal and defaults to binary.
- **nvoxels/nx, nvoxels/ny, nvoxels/nz**: These three parameters specify the number of voxels in the x, y, and z axes, respectively. These parameters are optional, as that information is already read from the geometry file. The only purpose of these parameters is to double-check the correctness of the specified file, especially in the binary case.
- **voxe-size/dx, voxe-size/dy, voxe-size/dz**: These three parameters specify the size of voxels (in cm) in the x, y, and z axes, respectively. As before, these parameters are optional and are only used to double-check the geometry file.
- **dsmax**: This optional section allows the user to set a *dsmax* value, in cm, for all voxels with a specific material index. This must be specified as follows:

  ```
  dsmax/mat-index value
  ```

- **enclosure-margin**: This parameter defines the margin, in cm, of a homogeneous enclosure containing the voxelized geometry (Figure 13). It is intended to take into account border effects, such as backscatter.

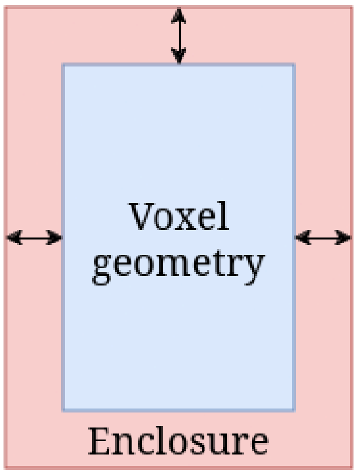


Figure 13: Voxelized geometry enclosure schema

- **enclosure-material**: Specify the enclosure material index.
- **print-ASCII**: If this boolean parameter is set to *true*, the processed geometry will be saved in a file named *voxelsASCII.rep* in ASCII format. This option is intended to be used with binary formats.

An example of configuration file for voxelized geometries is shown in the code 37.

```
geometry/type "VOXEL"
geometry/filename "3blocks.vx"
geometry/nvoxels/nx 60
geometry/nvoxels/ny 60
geometry/nvoxels/nz 80
geometry/voxel-size/dx 0.2
geometry/voxel-size/dy 0.2
geometry/voxel-size/dz 0.1

geometry/enclosure-margin 20
geometry/enclosure-material 5

geometry/dsmax/1 0.05
geometry/dsmax/2 0.02
geometry/dsmax/3 0.02
geometry/dsmax/4 0.50
```

Code 37: Configuration of voxelized geometry example

5.3.4.1 Binary format

Voxelized binary geometries consist of binary data generated by the *pen_voxelGeo* class. An example of how to create a geometry file using that class can be found in the test code:

src/tests/geometry/voxels/read_Dump.cpp

This example creates a randomly filled voxel geometry, stores it in a file, and finally loads the file to compare it with the originally created geometry. Additionally, another example can be found in the utility *geo2voxel*, whose source file is located at:

src/utilities/geometry/geo2voxel.cpp

This utility instantiates a geometry with the type and configuration specified by the user and creates a voxelized geometry file according to that geometry.

As shown in the mentioned codes, the function to fill a voxel geometry class is named `setVoxels`, illustrated in code 38. The initialization of voxel materials and densities is not shown. The required parameters are:

- A three-element array $(nvox)$ with the number of voxels in each axis (nx, ny, nz).
- The voxel sizes in each axis $(sizes)$ (dx, dy, dz) in cm.
- A one-dimensional array with the material assignment of each voxel $(voxMats)$, containing $nx \times ny \times nz$ elements.
- A one-dimensional array with the density factor of each voxel, also containing $nx \times ny \times nz$ elements
- A verbose level.

Once the voxel mesh has been created, the geometry file can be printed in binary format, ready for simulation, using the function `dump2File`. Additionally, the voxel geometry information can be printed in ASCII format using the function `printImage`.

```

unsigned nvox[3] = ...
double sizes[3] = ...
unsigned* voxMats = ...
double* voxDensFact = ...

pen_voxelGeo voxelgeo;
int err = voxelgeo.setVoxels(nvox,sizes,voxMats,voxDensFact,3);
if(err != 0){
    printf("Error using 'setVoxels': %d\n",err);
    return -1;
}
//Print ASCII file
voxelgeo.printImage("voxelGeo.ascii");
//Dump binary file
voxelgeo.dump2File("voxelGeo.bin");
```

Code 38: Set voxels function and geometry file creation

5.3.4.2 ASCII format

The ASCII format for voxelized geometries consists of the following elements, in order:

- Three positive, non-zero integer values representing the number of voxels in the x, y, and z axes.
- Three positive, non-zero numeric values specifying the size of a voxel in the x, y, and z axes.
- Two values for each voxel: first, an integer specifying the voxel material index; second, a numeric value corresponding to the density factor. Both values must be positive and greater than zero, as void material is not allowed inside a voxelized geometry.

The first lines of an example file in ASCII format for a voxelized geometry are shown in Code 39. This example represents a voxelized matrix of $(100 \times 200 \times 120)$ voxels with a voxel size of $(0.1 \times 0.1 \times 0.2)$ cm.

```

100 200 120
0.1 0.1 0.2
1 1.0
1 1.0
1 1.1
2 0.8
2 0.9
.
.
.
```

Code 39: ASCII voxelized geoemtry example.

5.3.5 DICOM

Medical images are usually stored using the international standard of Digital Imaging and Communications in Medicine (DICOM). PenRed implements a DICOM geometry module to convert the DICOM file to a voxel geometry automatically, which is ready to be simulated directly. As other PenRed components, DICOM geometries require a specific configuration structure to be used. This configuration includes the following parameters,

- **type**: Geometry type, must be set to "DICOM".
- **directory**: Specifies the relative path where the DICOM images are stored. Notice that all the DICOM images found in that folder are expected to be from the same image.
- **calibration**: This optional parameter is only used for CT images. This must be specified as an array of numbers which belong to a polynomial calibration to convert from Hounsfield Units (HU) to density (g/cm^3). If the calibration is not specified for CT images, the raw data will be used and the density must be assigned using other techniques, as we will see below.
- **default/material**: Specifies default material index for all voxels which material has not been assigned by other methods.
- **default/density**: Specifies default density for all voxels which density has not been assigned by any of the available methods.
- **intensity-ranges**: Provides a subsection to assign a material index and density value to all voxels inside the specified pixel value ranges. This subsection consists of the following parameters,
  - **material**: Material index to assign.
  - **density**: Density value (g/cm^3) to assign.
  - **low**: Lower range pixel value to assign this material and density.
  - **top**: Upper range pixel value to assign this material and density.

  So, all voxels with intensity values in the range $[low, top)$ will be assigned with the material index *material* and the density value *density*. To differentiate between ranges, each one requires a unique name. This name must not be the material name, but is advisable for debug purposes. An example of this configuration is shown in code 40, where we define an interval named *Air*.

```
geometry/intensity-ranges/Air/material 1
geometry/intensity-ranges/Air/low -2000
geometry/intensity-ranges/Air/top -500
geometry/intensity-ranges/Air/density 0.001290
```

Code 40: DICOM intensity ranges configuration

- **ranges**: Subsection analogous to *intensity-ranges* but using density ranges instead of pixel values. Thus it could be used to specify the voxels material indexes via density ranges. Notice that it is required a calibration curve to previously convert HU units to densities. This subsection consists of the following parameters,
  - **material**: Material index to assign.
  - **density-low**: Minimum density value for this range.
  - **density-top**: Maximum density value for this range.

    All voxels with a density value between $(density - low, density - top]$ will be assigned with the material index *material*.

- **contours**: Subsection that allows to the user to assign materials and densities according to contours stored inside the DICOM image. Each subsection of this type defines a single contour which name is specified in the corresponding parameters paths. Notice that the contour name must coincide with the contour name stored in the DICOM file. This subsection contains the following parameters,

  - **material**: Default material to assign to this contour. This is an optional field. If it is not provided, the existing material assign will not be overwritten.
  - **density**: Default density to assign to this contour (g/cm^3). This is an optional field. If it is not provided, the voxel density will be not overwritten.
  - **priority**: A priority value to control which contours are overwritten by other ones. Contours with lower priority values will be overwritten by contours with higher priority values.
  - **intensity-ranges**: Material and density can also be assigned defining intensity ranges specific for each contour. This method overwrites the specified default material and density for the contour. To define the ranges, use the same pattern explained for global **intensity-ranges**.
  - **ranges**: Materials can also be assigned defining density ranges specific for each contour. This method overwrites the specified default material for the contour. To define the ranges, use the same pattern explained for global **ranges**.

  For example, to configure a contour named *target*, with air in some regions, the configuration file should contain something like the lines shown in code 41.

```
geometry/contours/target/material 1
geometry/contours/target/density 1.05
geometry/contours/target/priority 1.0

geometry/contours/target/intensity-ranges/Air/material 2
geometry/contours/target/intensity-ranges/Air/low -2000
geometry/contours/target/intensity-ranges/Air/top -500
geometry/contours/target/intensity-ranges/Air/density 0.001290
```

Code 41: DICOM contour configuration

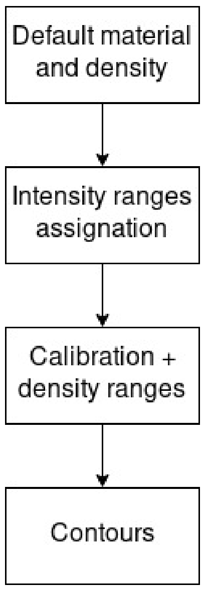
  To assign a uniform material and density in the contour, just don't specify any range.

- **enclosure-margin**: Like the voxel based geometry case, specify the distance between the DICOM mesh corner and the limit of an external enclosure, **in cm**.

- **enclosure-material**: Specify the enclosure material index.

- **print-ASCII**: This true/false optional configuration parameter, can be set to *true* to print the processed DICOM in ASCII format. The data will be stored in a file named *dicomASCII.rep*.

Notice that this geometry type presents several ways to assign material indexes and densities to voxels. Due to this characteristic, exists a method hierarchy where preferential methods overwrite the others. This preference is shown in the Figure 14.



Default material and density
Intensity ranges assignation
Calibration + density ranges
Contours

Figure 14: DICOM voxels material an density assign methods hierarchy.

Notice also that the allowed image modalities for DICOM geometry type are, by the moment, Computed Tomography (CT), Ultrasound (US), Radiotherapy Structure Set (RT-STRUCT) and Radio-therapy Plan (RTPLAN). Positron emission tomography (PT) is also accepted, but not for geometry construction purposes. Instead, PET images can be used to create a spatial sampling.

A DICOM example where voxel density and material are set using only intensity ranges can be found at the *example* folder. This one reproduces the GEANT IV DICOM simulation example using the DICOM image developed at [10]. So, the user must download that DICOM to reproduce the example.

5.3.5.1 Constraints

As segmentation by density or intensity ranges can sometimes produce undesired results, such as small clusters of materials, thin erroneous material layers at some interfaces, etc., the user can specify constraints to control these effects.

The constraints are specified with the following pattern:

```
constraints/constraint-name/path/to/element value
```

where *constraint-name* is a user-defined name for the constraint, and the available elements to specify are listed below:

- **material**: This mandatory parameter specifies the material index to which the constraint will be applied.
- **min-volume**: Sets the minimum volume, in cm^3, for voxel clusters of this material. If a group of voxels of this material has a smaller volume, it will be overwritten with boundary materials.
- **max-volume**: Sets the maximum volume, in cm^3, for voxel clusters of this material. If a group of voxels of this material has a larger volume, it will be overwritten with boundary materials.

- **max-clusters**: Specifies the maximum number of disconnected voxel clusters of the specified material. If more clusters are found, the clusters with the smallest volume will be overwritten with boundary materials.

Note that the constraints are applied after the density and intensity range segmentation. Therefore, contour segmentation will overwrite the results of these constraints.

5.3.6 Combo

The geometry type "COMBO" provides the capability to merge different geometries of any type in a single one. There is no limit to the number of geometries that can be combined.

To determine which geometry exists in each region, each one must be assigned with an integer priority in the range $[0, N)$, where N is the number of geometries to combine. Then, the geometry with priority 0 will overwrite geometries with priorities $[1, N)$, the priority 1 will overwrite the priorities $[2, N)$, and so on.

Regarding geometries with higher priority numbers, these will overwrite only the void regions of geometries with lesser priority index. Therefore, a particle will only "see" the geometry with priority n, at the position (x, y, z), if all geometries with priority indexes in the range $[0, n)$ are void at (x, y, z).

Notice that is not possible to define a "transparent" material to be overwrite by higher priority indexes as is done for void. This feature has not been implemented because its high impact on simulation speed.

Note: To avoid conflicts between body names, the bodies are renamed in the COMBO geometry. Each body is assigned a name following this pattern:

```
geoName_bodyName
```

where *geoName* represents the name of the geometry to which the body belongs, and *bodyName* is the name of the body within that geometry. However, when referencing bodies within the context of a specific geometry, you must use the local name, *bodyName*. This rule applies, for example, to the definition of detector or dsmax parameters.

The parameters to configure this type of geometry are listed below,

- **type**: Must be set to "COMBO"
- **geometries**: Is a subsection containing the information of each combined geometry. Each geometry is identified by a name. Therefore, to specify the parameters of geometry *accelerator* we must use the path,

  ```
  geometries/accelerator/path/to/parameter
  ```

  The parameters to be specified for each geometry are:

  - **priority**: The priority integer index in the range $[0, N)$, where N is the number of combined geometries.
  - **config**: A subsection containing all the required parameters to configure the geometry. Therefore, this subsection depends on the type of the geometry to be combined.

An example of configuration is shown in the Code 42, where three quadric geometries, named $1, 2, 3$, are combined.

```
geometry/type "COMBO"

geometry/geometries/1/priority 2
geometry/geometries/2/priority 1
geometry/geometries/3/priority 0

geometry/geometries/1/config/type "PEN_QUADRIC"
geometry/geometries/1/config/input-file "1.geo"
geometry/geometries/1/config/processed-geo-file "report1.geo"

geometry/geometries/2/config/type "PEN_QUADRIC"
geometry/geometries/2/config/input-file "2.geo"
geometry/geometries/2/config/processed-geo-file "report2.geo"

geometry/geometries/3/config/type "PEN_QUADRIC"
geometry/geometries/3/config/input-file "3.geo"
geometry/geometries/3/config/processed-geo-file "report2.geo"
```

Code 42: Geometry “COMBO” type configuration example.

Although combining geometries is generally slower, it could simplify the optimization in some cases, for example allowing the usage of modules in combined quadric geometries. Therefore, in some particular cases, in addition to simplify the geometry construction, it could speed-up the simulation.

A more interesting usage case is the combination of different types of geometries. For example, the Code 43 shows the combination of a seed constructed using quadric geometries with a DICOM based geometry where the materials are assigned via contours. In this example, the seed geometry overwrites the DICOM.

```

geometry/type "COMBO"

geometry/geometries/seed/priority 0
geometry/geometries/DICOM/priority 1

geometry/geometries/seed/config/type "PEN_QUADRIC"
geometry/geometries/seed/config/input-file "seed.geo"
geometry/geometries/seed/config/processed-geo-file "report.geo"

geometry/geometries/DICOM/config/type "DICOM"
geometry/geometries/DICOM/config/directory "test_mamaHDR"

#Contour assign
geometry/geometries/DICOM/config/enclosure-margin 20
geometry/geometries/DICOM/config/enclosure-material 4

geometry/geometries/DICOM/config/default/material 4
geometry/geometries/DICOM/config/default/density  1.2000E-03

geometry/geometries/DICOM/config/contours/body/material 1
geometry/geometries/DICOM/config/contours/body/density 1.0
geometry/geometries/DICOM/config/contours/body/priority 2.2

geometry/geometries/DICOM/config/contours/heart/material 2
geometry/geometries/DICOM/config/contours/heart/density 1.5
geometry/geometries/DICOM/config/contours/heart/priority 2.6

geometry/geometries/DICOM/config/contours/skin/material 3
geometry/geometries/DICOM/config/contours/skin/density 1.1
geometry/geometries/DICOM/config/contours/skin/priority 2.3

geometry/geometries/DICOM/contours/left lung/material 4
```

```
geometry/geometries/DICOM/contours/left lung/density 1.2E-03
geometry/geometries/DICOM/contours/left lung/priority 2.3

geometry/geometries/DICOM/contours/right lung/material 4
geometry/geometries/DICOM/contours/right lung/density 1.2E-03
geometry/geometries/DICOM/contours/right lung/priority 2.3

geometry/geometries/DICOM/contours/left ribs/material 5
geometry/geometries/DICOM/contours/left ribs/density 1.92
geometry/geometries/DICOM/contours/left ribs/priority 2.3

geometry/geometries/DICOM/contours/right ribs/material 5
geometry/geometries/DICOM/contours/right ribs/density 1.92
geometry/geometries/DICOM/contours/right ribs/priority 2.3

geometry/geometries/DICOM/contours/ptv/material 6
geometry/geometries/DICOM/contours/ptv/density 1.0
geometry/geometries/DICOM/contours/ptv/priority 2.5
```

Code 43: Geometry “COMBO” with quadric and DICOM geometries.

5.4 Tallies

Configurations for tallies are similar to sources. The pattern to configure tallies is exemplified in the code 44, where *tallies* is a constant text, *tally-name* is a user defined name for the tally, and *parameter/path* and *value* sets the tally parameters, whose depends on tally type.

```
tallies/tally-name/parameter/path value
```

Code 44: Tally configuration pattern

As sources and geometries, exists several tally types, so the user must specify the parameter *type* on each created tally. In addition to tally type, a path to save the tally results can be specified via the *outputdir* parameter. Both parameters are common to all tallies, and are shown in the Code 45.

```
tallies/tally-name/type "TYPE_STRING"
tallies/tally-name/outputdir "path/to/results/dir/"
```

Code 45: Tally common parameters

Notice that, in the *outputdir* value, the remaining path after the last slash (“/” or “\” for unix and windows based systems respectively) will be appended as a prefix to the results filename instead of be interpreted as a directory. The Table 8 shows some cases to exemplify this behaviour, where the left column represents the value of *outputdir* specified in the configuration file, and the right column the final file path in the OS file system. If the specified path does not exists, i.e. one or more directories have been not created by the user, PenRed will try to append the prefix “*not-found*” to the filename and create the corresponding file in the execution directory, in order to avoid losing the simulation progress.

outputdir path	**final path**
`dir1/`	dir1/filename
`dir1`	dir1filename
`dir1/dir2/`	dir1/dir2/filename
`dir1/dir2`	dir1/dir2filename

Table 8: Tally *outputdir* behaviour examples.

An example of complete tally configuration is shown in the Code 46, which corresponds to a cylindrical dose distribution tally. This one requires limits for radial distance (*rmin* to *rmax*), number of radial bins *nbinsr*, limits for z axis (*zmin* to *zmax*) and the number of z bins (*nbinsz*).

```
tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
tallies/cylDoseDistrib/print-xyz true
tallies/cylDoseDistrib/rmin 0.0
tallies/cylDoseDistrib/rmax 30.0
tallies/cylDoseDistrib/nbinsr 60
tallies/cylDoseDistrib/zmin 0
tallies/cylDoseDistrib/zmax 30.0
tallies/cylDoseDistrib/nbinsz 60
```

Code 46: Tally configuration pattern

An aspect to consider when we are choosing the limits, is that the behaviour of the intervals is $[min, max)$. Thus, in the previous example, *rmax* and *zmax* are out of their respective intervals.

The next subsections describe briefly the data measured in each tally and provide an example to use them at the configuration file.

5.4.1 Radial and Cylindrical Dose Distribution

This tally measures the absorbed dose in (eV/g) for each radial bin in the range of $[rmin, rmax)$, in cm and can be configured to measure the depth absorbed dose distribution. Regarding the units, all results are expressed in (eV cm/g) per history. Below is shown the available parameters of this tally,

- **type "CYLINDRICAL_DOSE_DISTRIB"**: Type name of the tally.
- **print-xyz**: This variable can be set to **true** to print more information in the results file. If activated, two extra values per bin coordinate are printed, which corresponds to the coordinates low and the average values. For z coordinates, the average value is considered at the middle point of the bin. For the r coordinate, the average is weighted with a weight proportional to the radius r. If $print - xyz$ is not specified, is set to **false** by default.
- **rmin**: Minimum value of the radial coordinate. Must be greater than zero and lower than *rmax*.
- **rmax**: Maximum value of the radial coordinate.
- **nbinsr**: Number of radial bins. Must be at least 1.
- **zmin**: Minimum value of the depth coordinate. Must be lower than *zmax* when *nbinsz* is set greater to zero.
- **zmax**: Maximum value of the depth coordinate. Must be set to *zmin* when depth absorbed dose is not measured.
- **nbinsz**: Number of depth bins. Must be set to zero when depth absorbed dose is not measured, otherwise, must be at least 1.
- **nbinsPhi**: Number of angular (ϕ) bins. By default, this parameter is set to 1.

Regarding the value types, the limits of radial and depth intervals must be real numbers, while the number of bins in each case must be integers. The output filename of this tally ends with *cylindricalDoseDistrib.dat* but, if we disable the depth absorbed dose measure in the configuration, the output filename will end with *radialDoseDistrib.dat*.

An example of configuration for this tally is shown in the code 47, which has been extracted from the file *disc.in* of the example 1-disc.

```
tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
tallies/cylDoseDistrib/print-xyz true      (optional)
tallies/cylDoseDistrib/rmin 0.0            (mandatory)
tallies/cylDoseDistrib/rmax 0.01           (mandatory)
tallies/cylDoseDistrib/nbinsr 50           (mandatory)
tallies/cylDoseDistrib/zmin 0.0            (mandatory)
tallies/cylDoseDistrib/zmax 0.005          (mandatory)
tallies/cylDoseDistrib/nbinsz 100          (mandatory)
tallies/cylDoseDistrib/nbinsPhi 1          (optional)
```

Code 47: Cylindrical dose distribution tally configuration

5.4.2 Emerging Particles Distribution

This tally measures the energy distribution of particles that leave the geometry within a specific energy range, expressed in particles/(eV·history). The tally considers two cases:

- **Down-bound emerging particles**: Particles whose Z-axis direction (W) is less than or equal to zero.
- **Up-bound emerging particles**: Particles whose Z-axis direction (W) is greater than zero.

Additionally, the tally measures the number of particles that leave the geometry according to their direction, dividing the spectrum into polar and azimuthal intervals in spherical coordinates. In this case, measurements are expressed in particles/(sr·history).

The configuration parameters for this tally are as follows:

- **type "EMERGING_PART_DISTRIB"**: Type name of the tally.
- **emin**: Minimum energy value. Must be lower than *emax*.
- **emax**: Maximum energy value.
- **nBinsE**: Number of energy bins. Must be at least 1.
- **nBinsTheta**: Number of polar bins. Must be greater than zero.
- **nBinsPhi**: Number of azimuthal bin, Must be greater than zero.

Regarding the values types, the energy limits must be positive numbers, while the bin numbers must be integers. The output files of this tally end as: *emergin-downbound.dat*, *emergin-upbound.dat*, *emergin-angle.dat*, with information for theta and phi values and *emergin-polar-angle.dat* with only angular information of theta values.

The code 48 is an example of this tally, which has been extracted from the configuration file *detector1.in* of the example 3-detector-1:

```
tallies/EmergingPartDistrib/type "EMERGING_PART_DISTRIB"
tallies/EmergingPartDistrib/emin 0.0             (mandatory)
tallies/EmergingPartDistrib/emax 1.45e6          (mandatory)
tallies/EmergingPartDistrib/nBinsE 280           (mandatory)
tallies/EmergingPartDistrib/nBinsTheta 45        (mandatory)
tallies/EmergingPartDistrib/nBinsPhi 18          (mandatory)
```

Code 48: Emerging particles tally configuration

5.4.3 Energy Deposition Body

This tally measures the energy, in eV per history, deposited in each body. To use it, only the *type* parameter, to select the tally itself, is required,

- **type "EDEP_BODY"**: Type name of the tally.

The output filename ends with *bodyEnergyDeposition.dat*.

A configuration example of this tally is show in the code 49, which belongs to the file *plane.in* of the example 2-plane.

```
tallies/bodyEDep/type "EDEP_BODY"
```

Code 49: Energy deposition in body tally configuration

5.4.4 Energy Deposition Material

This tally measures the energy, in eV per history, deposited in each material. To use it, only the *type* parameter, to select the tally itself, is required,

- **type "EDEP_MAT"**: Type name of the tally.

The corresponding output file ends with *materialEnergyDeposition.dat*

The code 50 shows a configuration example obtained from the configuration file *detector1.in* belonging to the example 3-detector-1.

```
tallies/matEDep/type "EDEP_MAT"
```

Code 50: Energy deposition in material tally configuration

5.4.5 Impact Detector

This tally can measure different magnitudes, including the fluence spectrum, particle energy spectrum, energy deposition spectrum, and particle age. Each is described below:

- Fluence: Measures the spectral fluence integrated over the detector volume, expressed in cm/eV. The output file provides the fluence for each particle type as well as the total fluence within the specified detector. The filename follows the format:*fluenceTackLength-num.dat*, where *num* is the corresponding detector number.

- Energy spectrum: Reports the energy spectrum of particles entering the specified detector volume. Particles generated inside the detector, such as secondary particles, are not included in this tally. The units are given in 1/(eV·history). The output file contains the energy spectrum for each particle type. The filename follows the format: *spectrum-impdet-num.dat*, where *num* is the number of the assigned detector.

- Age: Records the age distribution of particles impacting the designated detector, with units in 1/(seconds·history). The output file, named *age-impdet-num.dat*, contains the probability distribution across different time intervals for all simulated particles. As in the previous cases, num identifies the detector number.
- Energy deposition: Reports the energy deposition spectrum measured in the specified detector. The units are given in 1/(eV·history). The output file, named *energyDeposition-impdet-num.dat*, contains the probability distribution across energy intervals, considering all simulated particles.

Regarding the configuration parameters, these are listed below:

- **type "IMPACT_DET"**: Type name of the tally.
- **detector**: Detector index where the measurements of this tally are taken.
- **fluence**: Must be set to **true** to obtain fluence measurements, otherwise set it to **false**. If this information is not specified, **false** value will be assigned by default.
- **emin**: Minimum energy value for tallied particles in fluence, energy spectrum, energy deposition and **age spectrum** measurements.
- **emax**: Maximum energy value for tallied particles in fluence, energy spectrum, energy deposition and **age spectrum** measurements. Must be greater than *emin*.
- **nbin-energy**: Number of energy bins, which is used for fluence, energy spectrum and energy deposition. Must be at least 1.
- **spectrum**: Must be set to **true** to obtain spectrum measurements, otherwise set it to **false**. If this information is not specified, **false** is set as default value.
- **age**: Must be set to **true** to obtain age measurements, otherwise set it to **false**. If this information is not specified, **false** is set as default value.
- **enegy-dep**: Must be set to **true** to obtain energy deposition measurements, otherwise set it to **false**. If this information is not specified, **false** value will be assigned by default.
- **linearScale-spc**: Determines if energy spectrum measurements will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). When this parameter is not specified, linear scale is selected by default (**true**).
- **linearScale-edep**: By default is set to **true**, meaning that the energy deposition will be tallied using a linear scale. Instead, if set to **false**, the tally will be created following a logarithmic scale.
- **linearScale-age**: Determines if the output measurements for the age spectrum are expressed using a linear scale (**true**) or a logarithmic scale (**false**). If not specified, a linear scale is selected by default (**true**).
- **nbin-age**: Number of age interval bins. Is only required if **age** is active and must be at least 1.
- **age-min**: Minimum value of the age interval.
- **age-max**: Maximum value of the age interval. Must be greater than *age-min*.

Regarding the parameter types, the interval limits in each case must be real numbers and the bin numbers must be specified as integers.

To exemplify the tally usage, next are two examples to show diferent configurations for this tally. First, code 51 shows a part of the configuration file *plane.in* of the 2-plane example, where we the fluence, spectrum and age information are tallied. If fluence, spectrum, age or energy deposition are activated, their corresponding fields are mandatory, otherwise are neither used or expected.

```
tallies/ImpactDetector/type "IMPACT_DET"
tallies/ImpactDetector/detector 1                  (mandatory)
tallies/ImpactDetector/fluence true                (optional)
tallies/ImpactDetector/emin 1.0e5                  (mandatory/optional)
tallies/ImpactDetector/emax 3.5e7                  (mandatory/optional)
tallies/ImpactDetector/nbin-energy 100             (mandatory/optional)
tallies/ImpactDetector/linearScale-fln true        (optional)
tallies/ImpactDetector/spectrum true               (optional)
tallies/ImpactDetector/age true                    (optional)
tallies/ImpactDetector/linearScale-age false       (mandatory/optional)
tallies/ImpactDetector/nbin-age 100                (mandatory/optional)
tallies/ImpactDetector/age-min 1.0e-9              (mandatory/optional)
tallies/ImpactDetector/age-max 1.0e-8              (mandatory/optional)
```

Code 51: Impact detector tally configuration example 1

Secondly, code 52 shows some lines of the configuration file *detector1.in* that belongs to the example 3-detector-1. In this example, only the energy deposition information is tallied.

```
tallies/ImpactDetector/type "IMPACT_DET"
tallies/ImpactDetector/detector 1                  (mandatory)
tallies/ImpactDetector/emin 0.0e0                  (mandatory/optional)
tallies/ImpactDetector/emax 1.45e6                 (mandatory/optional)
tallies/ImpactDetector/nbin-energy 280             (mandatory/optional)
tallies/ImpactDetector/energy-dep true             (optional)
tallies/ImpactDetector/linearScale-edep true       (optional)
```

Code 52: Impact detector tally configuration example 2

5.4.6 Spatial Dose Distribution

This tally measures the 3D absorbed dose distribution along the intervals $[xmin, xmax)$, $[ymin, ymax)$, $[zmin, zmax)$ in cm. The units of dose values are eV/g per history. For each coordinates, the user must select the number of bins used to report the data in each axis (nx, ny, nz). In addition, the tally reports the depth dose distribution along the z coordinate in eV/(g/cm^2) .

The available tally parameters to configure are explained below,

- **type "SPATIAL_DOSE_DISTRIB"**: Type name of the tally.
- **xmin**: Minimum value of coordinate x.
- **xmax**: Maximum value of coordinate x. Must be greater than *xmin*.
- **nx**: Number of x bins. Must be, at least, 1.
- **ymin**: Minimum value of coordinate y.
- **ymax**: Maximum value of coordinate y. Must be greater than *ymin*.
- **ny**: Number of y bins. Must be at least 1.

- **zmin**: Minimum value of coordinate z.
- **zmax**: Maximum value of coordinate y. Must be greater than *ymin*.
- **nz**: Number of z bins. Must be at least 1.
- **print-depthDose**: Enables/disables printing depth dose information. Notice that this depth dose is discretized only in the Z axis, i.e. scores all energy depositions regardless the X and Y component. Therefore, in voxelized geometries, the enclosure is considered in the calculus. By default is disabled.

Considering the parameters type, those who specify coordinate values must be real numbers, while the number of bins for each coordinate must be specified as integers. For plotting purposes, two values per bin coordinate are given: the low and the middle point of each bin. Output filenames end with *spatialDoseDistrib-3D.dat* for 3D absorbed dose distribution and *depth-dose.dat* for depth dose distribution. To exemplify the configuration, the code 53 shows an example extracted from the configuration file *disc.in* that belongs to the example 1-disc-novr.

```
tallies/SpatialDoseDistrib/type "SPATIAL_DOSE_DISTRIB"
tallies/SpatialDoseDistrib/print-xyz true          (optional)
tallies/SpatialDoseDistrib/xmin 0.0                (mandatory)
tallies/SpatialDoseDistrib/xmax 1.0                (mandatory)
tallies/SpatialDoseDistrib/nx 1                    (mandatory)
tallies/SpatialDoseDistrib/ymin 0.0                (mandatory)
tallies/SpatialDoseDistrib/ymax 1.0                (mandatory)
tallies/SpatialDoseDistrib/ny 1                    (mandatory)
tallies/SpatialDoseDistrib/zmin 0.0                (mandatory)
tallies/SpatialDoseDistrib/zmax 0.005              (mandatory)
tallies/SpatialDoseDistrib/nz 100                  (mandatory)
tallies/SpatialDoseDistrib/print-depthDose false   (optional)
```

Code 53: Spatial Dose Distribution tally configuration

5.4.7 Angular Detector

This tally reports the angular energy spectrum in a specified detector. In this tally, the polar and azimuthal angles of particles are defined relative to the Z-axis (0, 0, 1) and the particle's direction. Therefore, these angles are **not related to the detector's orientation**. The recorded particles are filtered by energy and angular intervals. The energy spectra of particles are tallied in units of 1/(eV·sr·particle). Below are the parameters used to configure this tally:

- **type "ANGULAR_DET "**: Tally type name.
- **detector**: Detector index where the angular detector will be calculated.
- **emin**: Minimum energy value.
- **emax**: Maximum energy value. Must be greater than *emin*.
- **theta1**: Minimum value of polar angle.
- **theta2**: Maximum value of polar angle. The polar interval must be in the range $(0, 180)$ and *theta2* must be greater than *theta1*.
- **phi1**: Minimum value of azimuthal angle.

- **phi2**: Maximum value of azimuthal angle. The azimuthal interval must be in the range $(0, 360)$ or $(-180, 180)$ and *phi2* must be greater than *phi1*.
- **nBinsE**: Number of energy bins.
- **linearScale**: Determines if the output measurements are expressed in linear scale (**true**) or in logarithmic scale (**false**). If scale is not specified, linear scale will be set.

The expected parameter types are integers for *detector* and and the number of bins, and real numbers for the energy and angular limits. The output filename for this tally ends with *spc-angdet-num.dat* where *num* is the number of the assigned detector.

The code 54 belongs to the configuration of the example `1-disc-novr`, where this tally is used. The configuration file is named *disc.in*.

```
tallies/AngularDetector/type "ANGULAR_DET"
tallies/AngularDetector/detector 1                (mandatory)
tallies/AngularDetector/emin 0.0                  (mandatory)
tallies/AngularDetector/emax 40.5e3               (mandatory)
tallies/AngularDetector/theta1 90.0               (mandatory)
tallies/AngularDetector/theta2 180.0              (mandatory)
tallies/AngularDetector/phi1 0.0                  (mandatory)
tallies/AngularDetector/phi2 360.0                (mandatory)
tallies/AngularDetector/nBinsE 200                (mandatory)
tallies/AngularDetector/linearScale true          (optional)
```

Code 54: Angular detector tally configuration

5.4.8 Particle Generation

This tally reports information about the number of primary and secondary particles simulated. First, counts the number of primary particles that escape up bound and down bound and the number of absorbed particles. In addition, calculates the probabilities for secondary particles to go up bound, down bound and be absorbed.

This tally doesn't require to specify any parameter in the configuration file, only the tally type,

- **type "SECONDARY_GEN"**: Tally type name.

The output filename ends with *particleGeneration.dat*.

An example of the configuration file of this tally is code 55. This line is the same for all examples where this tally will be used.

```
tallies/secondary/type "SECONDARY_GEN"
```

Code 55: Particle generation tally configuration

5.4.9 Spherical Dose Distribution

This tally reports the absorbed dose distribution, in (eV/g) per history, in a spherical distribution, i.e. the bins follows spherical coordinates. By default, a single bin is used for each angular coordinate and only a radial distribution is tallied in the range $[rmin, rmax)$, specified in cm. The number of radial bins are denoted by *nbin* and, optionally, the user can activate the boolean $print - xyz$ (true) to print different values for radial coordinate in the output file: the low value and the average value. This average value is weighted proportionally to r^2.

Tally configuration parameters are listed following:

- **type "SPHERICAL_DOSE_DISTRIB "**: Type name of the tally.
- **print-xyz**: Can be set to **true** to print different values for radial coordinate in the output file i.e the low value and the average value. This average value is weighted proportionally to r^2.
- **rmin**: Minimum value of radial coordinate. Must be greater than zero.
- **rmax**: Maximum value of radial coordinate. Must be greater than *rmin*.
- **nr**: Number of radial bins. Must be at least 1.
- **ntheta**: Number of polar (θ) bins. By default, this parameter is 1.
- **nphi**: Number of azimuth (ϕ) bins. By default, this parameter is 1.

Regarding the parameter types, all limits are expected to be real numbers and the number of bins integers. The output filename ends with *sphericalDoseDistrib.dat*. Any of the provided examples use this tally. However, a configuration example is shown in the code 56.

```
tallies/SphericalDose/type "SPHERICAL_DOSE_DISTRIB)"
tallies/SphericalDose/print-xyz true            (mandatory)
tallies/SphericalDose/rmin 0.0                  (mandatory)
tallies/SphericalDose/rmax 0.06                 (mandatory)
tallies/SphericalDose/nr 65                     (mandatory)
tallies/SphericalDose/ntheta 1                  (optional)
tallies/SphericalDose/nphi 1                    (optional)
```

Code 56: Spherical dose distribution tally configuration

5.4.10 Phase Space File (PSF)

This tally generates a particle phase space file storing all particles impacting the specified detector. Additionally, the impacting particles can be filtered by energy range and particle type.

The available tally parameters for the configuration file are listed below:

- **type "PSF "**: Type name of the tally.
- **detector**: Detector index where this tally will be calculated.
- **emin**: Minimum energy value.
- **emax**: Maximum energy value. Must be greater than *emin*.
- **particles**: Section to enable or disable particle types recording. In this section, the user can specify the following parameters:
  - **default**: This optional boolean value enables (true) or disables (false) the particle recording for all particle types. Defaults to true.
  - ***particle-name***: This optional boolean parameter, where *particle-name* must be replaced by a valid particle name (e.g., *gamma*), disables (false) or enables (true) recording of the specified particle type. If defined, it overrides the *default* parameter.

```
tallies/psf/type "PSF"       (mandatory)
tallies/psf/detector 1       (mandatory)
tallies/psf/emin 0.0         (mandatory)
tallies/psf/emax 6.1e6       (mandatory)
tallies/psf/particles/default false    (optional)
tallies/psf/particles/gamma true       (optional)
```

Code 57: Example of phase space file tally configuration where only gamma particles are recorded

This tally can be used with multiple threads. In that case, each thread will temporarily store its PSF particles in an independent file. When the simulation finishes, all files will be concatenated automatically, ordered by thread ID in ascending order.

5.4.11 Kerma track length estimator

This tally is based on the equivalence of particle fluence and the total photon path length per unit volume. A complete description of the estimator can be found at [11]. The estimator could be tallied using three type of meshes: Voxel or cartesian based, cylindrical and spherical meshes. The required configuration for this tally is listed below.

- **type "KERMA_TRACK_LENGTH"**: Type name of the tally.
- **emin**: Photon minimum energy to be considered (eV).
- **emax**: Photon maximum energy to be considered (eV).
- **dataFiles**: One data filename with the μ_{en}/ρ (cm^2/g) coefficients per tallied material is required. However, if the file does not exists, the tally will create the corresponding μ_{en}/ρ data automatically. Then, the data will be stored in the specified file.

  Each filename is specified following this pattern:

  ```
  path/to/tally/dataFiles/material-number "filename"
  ```

  The data files format consists of two columns. The first one stores the energy in eV and the second the corresponding μ_{en}/ρ value in (cm^2/g). The provided points must contain the energy range $[emin, emax]$. However, is not required a constant energy distance between points. Furthermore, there are no limit to the number of points in each data file. Notice that materials with no μ_{en}/ρ data filename provided will be ignored by the tally. To obtain the μ_{en}/ρ coefficients manually, the utility `mutren` can be used. This one is provided in the PenRed package and described in the section 8.2. On the other hand, the user can let it be calculated automatically, as discussed before.

- **cartesian**: If this optional section exists, the tally will record the estimator using a regular voxel mesh. The required parameters are listed below:
  - **cartesian/nx**: Number of bins on the X axis.
  - **cartesian/xmin**: Init mesh point on X axis in cm.
  - **cartesian/xmax**: Limit mesh point on X axis in cm.
  - **cartesian/ny**: Number of bins on the Y axis.
  - **cartesian/ymin**: Init mesh point on Y axis in cm.
  - **cartesian/ymax**: Limit mesh point on Y axis in cm.

  - **cartesian/nz**: Number of bins on the Z axis.
  - **cartesian/zmin**: Init mesh point on Z axis in *cm*.
  - **cartesian/zmax**: Limit mesh point on Z axis in *cm*.

- **cylindrical**: If this optional section exists, the tally will record the estimator using a cylindrical mesh. The required parameters are listed below:
  - **cylindrical/nr**: Number of radial bins.
  - **cylindrical/rmin**: Cylinder minimum radius in *cm*.
  - **cylindrical/rmax**: Cylinder maximum radius in *cm*.
  - **cylindrical/nphi**: Number of angular (ϕ) bins.
  - **cylindrical/nz**: Number of Z bins.
  - **cylindrical/zmin**: Init mesh point on Z axis in *cm*.
  - **cylindrical/zmax**: Limit mesh point on Z axis in *cm*.

- **spherical**: If this optional section exists, the tally will record the estimator using a spherical mesh. The required parameters are listed below:
  - **spherical/nr**: Number of radial bins
  - **spherical/rmin**: Sphere minimum radius in *cm*.
  - **spherical/rmax**: Sphere maximum radius in *cm*.
  - **spherical/ntheta**: Number of polar angular (θ) bins.
  - **spherical/nphi**: Number of azimuth angular (ϕ) bins.

Notice that more than one mesh type could be active at the same tally. In these cases, the tally will generate one report for each mesh type. In code 58 we show a configuration example for that tally.

```
tallies/kermaTrackLength/type "KERMA_TRACK_LENGTH"          (mandatory)
tallies/kermaTrackLength/emin 1.0e3                         (mandatory)
tallies/kermaTrackLength/emax 1.5e6                         (mandatory)
tallies/kermaTrackLength/dataFiles/1 "mu-Water1000.mat"     (mandatory)

#Optional section:
tallies/kermaTrackLength/cylindrical/nr 10                  (mandatory)
tallies/kermaTrackLength/cylindrical/rmin 0.1               (optional)
tallies/kermaTrackLength/cylindrical/rmax 10.0              (mandatory)
tallies/kermaTrackLength/cylindrical/nphi 10                (mandatory)
tallies/kermaTrackLength/cylindrical/nz 10                  (mandatory)
tallies/kermaTrackLength/cylindrical/zmin -10.0             (mandatory)
tallies/kermaTrackLength/cylindrical/zmax  10.0             (mandatory)

#Optional section:
tallies/kermaTrackLength/cartesian/nx 10                    (mandatory)
tallies/kermaTrackLength/cartesian/xmin -10.0               (mandatory)
tallies/kermaTrackLength/cartesian/xmax  10.0               (mandatory)
tallies/kermaTrackLength/cartesian/ny 10                    (mandatory)
tallies/kermaTrackLength/cartesian/ymin -10.0               (mandatory)
tallies/kermaTrackLength/cartesian/ymax  10.0               (mandatory)
tallies/kermaTrackLength/cartesian/nz 10                    (mandatory)
tallies/kermaTrackLength/cartesian/zmin -10.0               (mandatory)
tallies/kermaTrackLength/cartesian/zmax  10.0               (mandatory)

#Optional section:
```

```
tallies/kermaTrackLength/spherical/nr 10                    (mandatory)
tallies/kermaTrackLength/spherical/min 0.2                  (optional)
tallies/kermaTrackLength/spherical/rmax 10.0                (mandatory)
tallies/kermaTrackLength/spherical/ntheta 10                (mandatory)
tallies/kermaTrackLength/spherical/nphi 10                  (mandatory)
```

Code 58: Kerma track length tally configuration

Notice that the elements inside an optional section are mandatory only if the section exists. For example, if the section `tallies/kermaTrackLength/cartesian` doesn't exists, any of the cartesian parameters are required, such as *cartesian/nx* or *cartesian/xmin*.

Regarding the parameter types, all bin number are expected to be integers, while the corresponding limits are expected to be real numbers.

5.4.12 CT tally

This tally has been created to collect the particles that arrive at a virtual detector, providing a file with a CT sinogram and a material energy deposition report by projection. The tally information needed in the configuration file is described below and in Figure 15.

- **type "CT_SINOGRAM"**: Type name of the tally.
- **emin**: Photon minimum energy to be collect (eV).
- **emax**: Photon maximum energy to be collect (eV).
- **npixels**: Number of pixels of the virtual detector.
- **pixel-depth**: Size of pixel deph (cm).
- **r-inner**: Radius of the virtual detector. Corresponds to the isocenter-detector distance (cm).
- **xOrigin, yOrigin, zOrigin**: x,y and z values of the CT origin. Corresponds to the position of the CT isocenter (cm).
- **phi-ini**: Initial angular position of the CT detector, in degrees. Must be set in front of the CT source with an angular increment of 180 degrees.
- **phi-end**: Final angular position of the CT detector, in degrees. Must be set in front of the CT source with an angular increment of 180 degrees.
- **nProjections**: Number of CT projections.
- **aperture**: Angle subtended by the detector with respect to the isocenter, in degrees.

  Usually, the angular aperture (β) is provided by CT manufacturers as the angle subtended with respect the CT source instead of with respect to the isocenter. The change to the aperture needed by the configuration file (α) can be done easily if the distance source-detector is twice the distance isocenter-detector. In this case, the user can apply the theorem of the central angle: The central angle subtended by two points on a circle is twice the inscribed angle subtended by those points. This theorem only holds when the source point is in the major arc. If this point is in the minor arc, it is, between the detector limits, the relationship between these two angles is that the inscribed angle is the supplement of half the central angle.

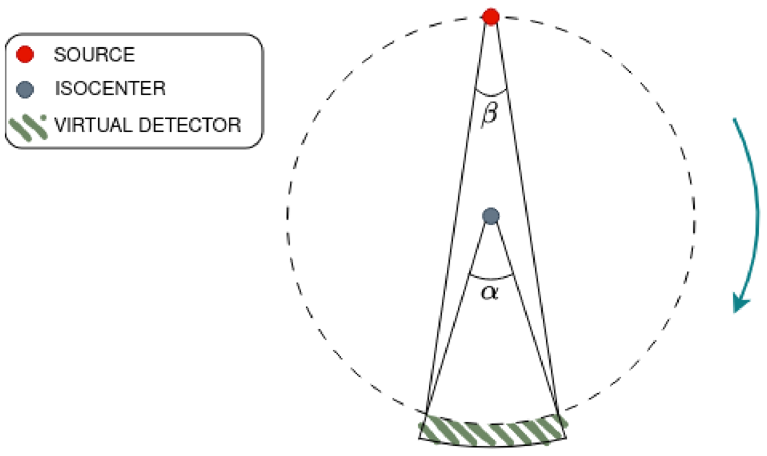


Figure 15: Schematic representation for the CT tally parameters

- **particle**: particle type that will be collect by the detector.
- **tmin**: Minimum time stamp of the particles of the first projection (s).
- **timeInterval**: Time interval between projections (s).
- **scattered**: This parameter offers the possibility to collect all the photons in the energy range if it is set to true or collect only photons which have not been suffer scatter. This is an optional parameter which is set to true by default.

```
tallies/CTsinogram/type "CT_SINOGRAM"                          (mandatory)
tallies/CTsinogram/emin 0.0                                    (mandatory)
tallies/CTsinogram/emax 1.0e5                                  (mandatory)
tallies/CTsinogram/npixels 700                                 (mandatory)
tallies/CTsinogram/pixel-depth 4                               (mandatory)
tallies/CTsinogram/r-inner 52.0                                (mandatory)
tallies/CTsinogram/xOrigin 0.0                                 (mandatory)
tallies/CTsinogram/yOrigin 0.0                                 (mandatory)
tallies/CTsinogram/zOrigin 0.0                                 (mandatory)
tallies/CTsinogram/phi-ini 180                                 (mandatory)
tallies/CTsinogram/phi-end 540                                 (mandatory)
tallies/CTsinogram/angularStep 0.5                             (mandatory)
tallies/CTsinogram/aperture 44.0                               (mandatory)
tallies/CTsinogram/particle "gamma"                            (mandatory)
tallies/CTsinogram/tmin 0.0                                    (mandatory)
tallies/CTsinogram/timeInterval 120.0                          (mandatory)
tallies/CTsinogram/scattered true                              (mandatory)
```

Code 59: CT tally configuration

5.4.13 DICOM kerma tally (Dose Volume Histogram)

This tally has been implemented to obtain the Dose Volume Cummulative Histogram (DVH) for each contoured structure of a DICOM image. Therefore, this tally requires a DICOM based geometry to be used. In addition, the specified DICOM must include a RTSTRUCT file with the contour information.

To obtain the absorbed dose in Gy for each volume, this tally calculates the kerma, using the kerma track length tally described in the section 5.4.11. However, because the voxelized structure of DICOM images, it uses only the cartesian scoring method. The tally information needed in the configuration file is described following:

- **type "DICOM_KERMA_TRACK_LENGTH"**: Type name of the tally.
- **kerma/emin**: Photon minimum energy to be considered (eV).
- **kerma/emax**: Photon maximum energy to be considered (eV).
- **kerma/dataFiles**: One data file with the μ_{en}/ρ (cm^2/g) coefficients per tallied material is required. This parameter is detailed in the Kerma track length estimator tally, section 5.4.11.
- **prescribedDose**: Treatment prescribed dose, in Gy. If this parameter has not been specified, it is set to 1 Gy.
- **MCkerma2Dose**: Conversion factor from kerma MC simulation units (eV/g per history) to absolute dose units, which must be set in Gy/(eV/g per history).
  The *MCkerma2Dose* factor is calculated using the equation 2.

$$MCkerma2Dose = \frac{1}{100}\frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \tag{2}$$

where $S_{k,PLAN}$ is the *source air-kerma strength* [12] which can be found in the DICOM plan in U units ($1U = cm^2 \cdot cGy/h$). The $S_{k,MC}$ value is the *source air-kerma strength* obtained for the source with Monte Carlo simulation. The last one can be calculated with the following relation,

$$S_{k,MC} = K_{MC}(d)d^2 \quad \left[\frac{eV \cdot cm^2}{g \cdot history}\right] \tag{3}$$

where K_{MC} is the kerma in air in a thin air cell, such as a ring with $0.1cm$ width and $0.1cm$ depth, located on the seed transversal axis at a reference distance d, which is usually set to $d = 10$ cm or $d = 100$ cm. Also, the source must be located in a vacuum media. This configuration example is shown in the figure 16.

Thus, the resulting units for the S_k quotient are,

$$\begin{aligned}
&\frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[\frac{g \cdot history}{eV \cdot cm^2}\right]\left[\frac{cm^2 \cdot cGy}{h}\right][h] \\
&\frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[\frac{cGy \cdot history}{(eV/g)}\right]
\end{aligned} \tag{4}$$

Converting to Gy, we obtain the units of the equation 2.

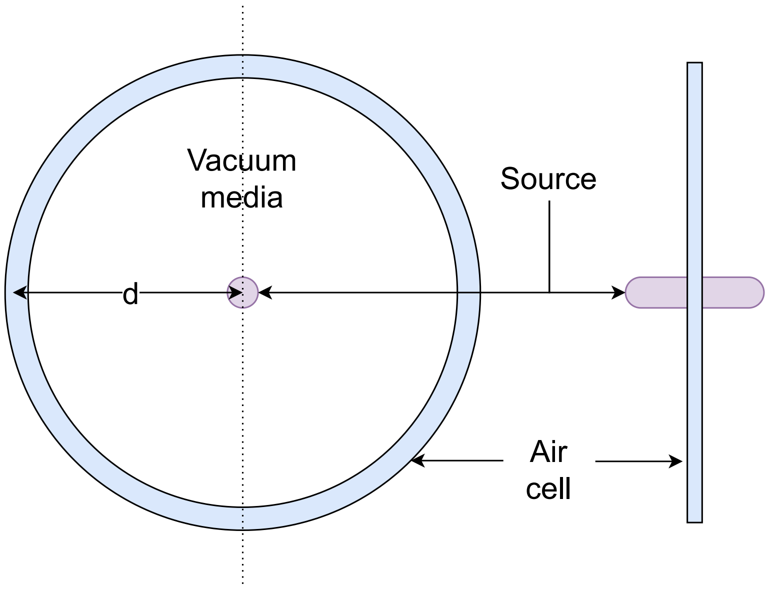


Figure 16: Geometry schema to calculate the $S_{k,MC}$ factor using a ring air cell situated at a reference distance d. The left image shows the central Z plane. The right image shows the central X or Y plane.

$$MCkerma2Dose = \frac{1}{100}\frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[\frac{Gy \cdot history}{(eV/g)}\right] \tag{5}$$

If the conversion factor is not specified, it will be set to 1.

- **DVH-maxDose**: Maximum dose value for the DVHs. If it is not specified, its value is set to 3 times the prescribedDose.
- **DVH-bins**: Number of bins for the DVHs. If this parameter has not been specified, it is set to 200.

```

tallies/DICOMkerma/type "DICOM_KERMA_TRACK_LENGTH"             (mandatory)

tallies/DICOMkerma/kerma/emin   2.5e3                          (mandatory)
tallies/DICOMkerma/kerma/emax   1.39950E+06                    (mandatory)
tallies/DICOMkerma/kerma/dataFiles/1 "prostate-mutren.dat"     (mandatory)
tallies/DICOMkerma/kerma/dataFiles/2 "uretra-mutren.dat"       (mandatory)
tallies/DICOMkerma/kerma/dataFiles/3 "rectum-mutren.dat"       (mandatory)
tallies/DICOMkerma/kerma/dataFiles/4 "bladder-mutren.dat"      (mandatory)
tallies/DICOMkerma/kerma/dataFiles/5 "body-mutren.dat"         (mandatory)
tallies/DICOMkerma/kerma/dataFiles/6 "bone-mutren.dat"         (mandatory)

tallies/DICOMkerma/prescribedDose 15                           (optional)
tallies/DICOMkerma/MCkerma2Dose 0.06831                        (optional)
tallies/DICOMkerma/DVH-maxDose 40                              (optional)
tallies/DICOMkerma/DVH-bins 1000                               (optional)
```

Code 60: DICOM kerma tally configuration

Notice that the DICOM is not specified in the source configuration. This is because the DICOM image is taken directly from the geometry.

5.4.14 Tracks

The *Track* tally stores the simulated particle tracks in separate files by particle type. Note that a single history could produce long tracks, especially for electrons, and the generated files can use a large amount of disk space if the number of tracked histories is large. It is recommended not to track more than 50 histories, depending on the energy, simulated particles, and geometry. The tally parameters are:

- **type**: Must be set to **TRACK**.
- **nhists**: Specify the number of histories to be tracked.

Take into account that only particles reaching the geometry system will be tracked. That is, if a particle is sampled in a void region and never reaches a non-void body, it will not appear in the corresponding track file.

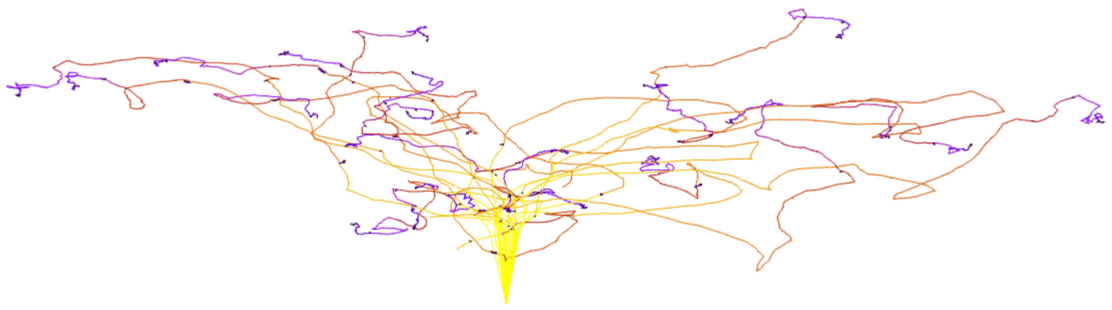


Figure 17: Example of electron tracks in a water medium. The track color shows changes in the electron energy.

In the generated files, particle tracks are separated by two blank lines, and each line contains the whole state of a particle at the specified point.

5.4.15 Primary and Scatter Separated (PSS)

This one is a wrapper tally which score the contribution of primary, scattered and multi-scattered radiation in three independent counters. To be able to combine it with different counting procedures, it delegates the counting process to another existing tally type. However, it has been designed for tallies scoring energy/dose deposition and will not be compatible with all existing tallies.

To determine if a particle is considered primary, scattered or multi-scattered radiation, a counter or bookkeeping (bkp) is used for each generated particle. However, only interactions and productions of the selected particle type will increase the counter. For example, for photons as the main particle, the following rules applies:

- Primary photons (γ_1) start with $bkp = 0$. Any photon interaction (e.g., Compton) occurring outside the source generates $bkp+ = 1$ (e.g., $bkp_{\gamma_1} = 1$)
- Any secondary particle (e.g., Compton electrons), e_1, inherits the generation of the incident photon (e.g., $bkp_{e_1} = 0$).
- Any photon produced by these secondary particles (e.g., bremsstrahlung photon), γ_2, needs $bkp+ = 1$ (e.g., $bkp_{\gamma_2} = 1$).

- The source materials (e.g., active core and encapsulation) don't play any role in the bkp of primary photons. However, once the primary particle is scattered once in the phantom (e.g., water), the source starts acting like any other material, adding +1 to the scattered photons' bkp.

The configuration of this tally contains the following parameters,

- **type "PSS"**: Type name of the tally.
- **PSS-particle**: Specify which particle must be considered as main radiation. If it is not specified, photons are selected by default.
- **PSS-source-mats**: Consists of an array with the material indexes considered as source materials. If it is not specified, no material will be considered as source.
- **PSS-tally**: Specify the tally type to be used to perform the scoring. For example **SPATIAL_DOSE_DISTRIB** could be used. As discused, not all tallies are compatible with the PSS. The following tally types are compatible,

  - "ANGULAR_DET"
  - "CYLINDRICAL_DOSE_DISTRIB"
  - "DICOM_DOSE_DISTRIB"
  - "DICOM_KERMA_TRACK_LENGTH"
  - "EDEP_BODY"
  - "EDEP_MAT"
  - "EMERGING_PART_DISTRIB"
  - "KERMA_TRACK_LENGTH"
  - "SPATIAL_DOSE_DISTRIB"
  - "SPHERICAL_DOSE_DISTRIB"

  and the non compatible types are,

  - "CT_SINOGRAM"
  - "IMPACT_DET"
  - "PSF"
  - "PSS"
  - "SECONDARY_GEN"
  - "TRACK"

  If the specified tally is not in either of the two lists a warning will be shown and the correct behaviour of the tally is not guaranted.

In addition to the **PSS** parameters, the specific parameters of the selected tally must be specified to configure them, as is shown in the Code 61. In this example, a cylindrical dose tally is used and materials 1 and 2 are considered as source materials.

```
tallies/cylDoseDistribPSS/type "PSS"
tallies/cylDoseDistribPSS/PSS-source-mats [1,2]
tallies/cylDoseDistribPSS/PSS-tally "CYLINDRICAL_DOSE_DISTRIB"
tallies/cylDoseDistribPSS/print-xyz true
tallies/cylDoseDistribPSS/rmin 0.0
tallies/cylDoseDistribPSS/rmax 0.01
tallies/cylDoseDistribPSS/nbinsr 50
tallies/cylDoseDistribPSS/zmin 0.0
tallies/cylDoseDistribPSS/zmax 0.005
tallies/cylDoseDistribPSS/nbinsz 100
```

Code 61: PSS tally configuration example combined with a cylindrical dose distribution

5.4.16 Detection Spatial Distribution

This tally records the energy spectrum of particles reaching a specified detector within a 3D grid. The available configuration parameters are as follows:

- **type**: Must be set to "DETECTION_SPATIAL_DISTRIB"
- **spatial/nx**: Specifies the number of spatial bins to use along the X axis. This value is optional and defaults to 1.
- **spatial/xmin**: Sets the origin of the spatial grid along the X axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/xmax**: Sets the limit of the spatial grid along the X axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/ny**: Specifies the number of spatial bins to use along the Y axis. This value is optional and defaults to 1.
- **spatial/ymin**: Sets the origin of the spatial grid along the Y axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/ymax**: Sets the limit of the spatial grid along the Y axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/nz**: Specifies the number of spatial bins to use along the Z axis. This value is optional and defaults to 1.
- **spatial/zmin**: Sets the origin of the spatial grid along the Z axis, in cm. This is required only if *spatial/nz* is specified.
- **spatial/zmax**: Sets the limit of the spatial grid along the Z axis, in cm. This is required only if *spatial/nz* is specified.
- **detector**: Specifies the detector index where the data will be collected. The detector must be previously defined in the geometry configuration.
- **energy/nbins**: Specifies the number of energy bins to register the spectrum. This value is optional and defaults to 1.
- **energy/emin**: Sets the lower limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **energy/emax**: Sets the upper limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.

- **particle**: Specifies the particle to be registered by name (text identifier).
- **printBins**: This optional boolean value enables/disables printing the spatial and energy bin indexes in the results file. The default is disabled.
- **printCord**: This optional boolean value enables/disables printing the bin coordinates, both spatial and energetic, in the results file. The default is enabled.

A configuration sample is shown in the Code 62, corresponding to the example *quadrics/9-measure-source/fromSource*.

```
tallies/SpatialDetectior1/type "DETECTION_SPATIAL_DISTRIB"
tallies/SpatialDetectior1/spatial/nx 100
tallies/SpatialDetectior1/spatial/xmin -8.0
tallies/SpatialDetectior1/spatial/xmax  8.0
tallies/SpatialDetectior1/spatial/ny 50
tallies/SpatialDetectior1/spatial/ymin -4.0
tallies/SpatialDetectior1/spatial/ymax  4.0
tallies/SpatialDetectior1/detector 1
tallies/SpatialDetectior1/particle "gamma"
tallies/SpatialDetectior1/energy/nbins 100
tallies/SpatialDetectior1/energy/emin 10e3
tallies/SpatialDetectior1/energy/emax 1.5e5
```

Code 62: Detection spatial distribution tally configuration

5.4.17 Emerging Spherical Distribution

This tally records the energy spectrum of particles emerging from the geometry system within a spherical grid. In addition, the last interaction position of emerging particles is recorded in a spatial 3D grid. The available configuration parameters are as follows:

- **type**: Must be set to "EMERGING_SPHERICAL_DISTRIB"
- **spatial/nx**: Specifies the number of bins to use along the X axis within the spatial distribution. This value is optional and defaults to 1.
- **spatial/xmin**: Sets the origin of the spatial grid along the X axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/xmax**: Sets the limit of the spatial grid along the X axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/ny**: Specifies the number of spatial bins to use along the Y axis within the spatial distribution. This value is optional and defaults to 1.
- **spatial/ymin**: Sets the origin of the spatial grid along the Y axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/ymax**: Sets the limit of the spatial grid along the Y axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/nz**: Specifies the number of spatial bins to use along the Z axis within the spatial distribution. This value is optional and defaults to 1.
- **spatial/zmin**: Sets the origin of the spatial grid along the Z axis, in cm. This is required only if *spatial/nz* is specified.

- **spatial/zmax**: Sets the limit of the spatial grid along the Z axis, in cm. This is required only if *spatial/nz* is specified.
- **theta/nbins**: Specifies the number of polar bins. This value is optional and defaults to 1.
- **theta/min**: Sets the minimum polar angle to tally, in degrees. This value defaults to 0.
- **theta/max**: Sets the maximum polar angle to tally, in degrees. This value defaults to 180.
- **phi/nbins**: Specifies the number of azimuthal bins. This value is optional and defaults to 1.
- **phi/min**: Sets the minimum azimuthal angle to tally, in degrees. This value defaults to 0.
- **phi/max**: Sets the maximum azimuthal angle to tally, in degrees. This value defaults to 360.
- **energy/nbins**: Specifies the number of energy bins to register the spectrum. This value is optional and defaults to 1.
- **energy/min**: Sets the lower limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **energy/max**: Sets the upper limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **particle**: Specifies which particles will be registered by name (text identifier). The particle name is specified in the path and a boolean value is set to enable/disable it:

  `particle/${particle-name} value`

  By default, all particles are disabled.
- **printBins**: This optional boolean value enables/disables printing the spatial and energy bin indexes in the results file. The default is disabled.
- **printCord**: This optional boolean value enables/disables printing the bin coordinates, both spatial and energetic, in the results file. The default is enabled.

A configuration sample is shown in the Code 63.

```
tallies/angularEmerging/type "EMERGING_SPHERICAL_DISTRIB"
tallies/angularEmerging/energy/min 5.0e3
tallies/angularEmerging/energy/max 50e3
tallies/angularEmerging/energy/nbins 90
tallies/angularEmerging/theta/nbins 720

tallies/angularEmerging/spatial/nx 1
tallies/angularEmerging/spatial/xmin -10.0
tallies/angularEmerging/spatial/xmax  10.0
tallies/angularEmerging/spatial/ny 1
tallies/angularEmerging/spatial/ymin -10.0
tallies/angularEmerging/spatial/ymax  10.0
tallies/angularEmerging/spatial/nz 200
```

```
tallies/angularEmerging/spatial/zmin    0.0
tallies/angularEmerging/spatial/zmax 100.0

tallies/angularEmerging/particle/gamma true
```

Code 63: Emerging spherical distribution tally configuration

5.5 Variance Reduction

In some cases, simulations may take an excessive amount of computational time depending on the statistical nature of the quantity being measured. To artificially increase the probability of desired events occurring, variance reduction techniques (VR) are often applied. PenRed implements the very same variance reduction (VR) techniques as the original PENELOPE FORTRAN code along with several additional methods. For generic simulations, these include interaction forcing (*IF*), x-ray splitting, and bremsstrahlung splitting. For phase space file based simulations, PenRed also adds particle splitting and the Russian Roulette technique. Notice that variance reduction for phase space files is configured via the corresponding particle source parameters, as outlined in section 5.2. Additionally, PenRed implements generic splitting and Russian Roulette as VR modules and provides the capability to add custom VR techniques.

PenRed has two types of VR techniques. First, the context specific techniques, implemented within the particle class, such as interaction forcing and bremsstrahlung splitting. Then, there are VR techniques implemented as independent modules, such as x-ray splitting, generic splitting or Russian Roulette.

5.5.1 Context specific

To configure context specific VR techniques, the involved parameters are specified in the configuration file using the prefix *VR*, as shown in Code 64. In this code, the field *vr-technique* defines the specific VR technique to apply, such as interaction forcing (*IForcing*) or bremsstrahlung splitting (*bremss*). The *name* field serves as a user-defined identifier for this VR technique. Finally, the fields *parameter/path* and *value* vary based on the selected VR type, with detailed descriptions provided in the following sections for each technique.

```
VR/vr-technique/name/parameter/path value
```

Code 64: Variance reduction configuration pattern

5.5.1.1 Interaction forcing (IF) configuration

The configuration for this VR technique is shown in Code 65, corresponding to the example *4-x-ray*. The parameters are described as follows:

- **particle**: Specifies the type of particle to which the technique will be applied.
- **interaction**: Sets the numerical identifier for the interaction type, as explained in section 4.4.
- **factor**: Defines the amplification factor for interaction forcing. A negative value indicates that a particle with energy $E = EPMAX$ is expected to interact an average of *+factor* times as it decelerates to rest (for electrons and positrons) or across a mean free path (for photons), similar to the handling in the PENELOPE main program.
- **min-weight**: Sets the minimum particle weight required to apply the IF technique; particles below this weight threshold will not be forced.

- **max-weight**: Sets the maximum particle weight for applying the IF technique; particles above this weight threshold will not be forced.

- **bodies/body-alias**: Defines the bodies in which interaction forcing will be enabled or disabled. To specify a body, add its alias under the *bodies* section and set its value to *true* to *enable* or *false* to *disable* IF in that body. By default, IF is *false* for all bodies. Multiple body aliases can be specified in this section to control IF for various bodies, as shown in Code 65, where bodies with aliases *1*, *2*, and *3* are enabled.

- **materials/material-index**: Similar to the *bodies* section, the *materials* section allows users to enable or disable IF for specific materials. Here, material indices are used instead of body aliases to identify the materials.

```
VR/IForcing/VR1/particle "electron"
VR/IForcing/VR1/interaction 2
VR/IForcing/VR1/factor 400
VR/IForcing/VR1/min-weight 0.1
VR/IForcing/VR1/max-weight 2
VR/IForcing/VR1/bodies/1 true
VR/IForcing/VR1/bodies/2 true
VR/IForcing/VR1/bodies/3 true
```

Code 65: Interaction forcing configuration for bodies

To apply interaction forcing to materials, replace *bodies* with *materials* and specify the desired material index as the *value* parameter. For instance, to apply the previously described IF technique to material with index 2, the configuration would be as shown in Code 66.

```
VR/IForcing/VR1/particle "electron"
VR/IForcing/VR1/interaction 2
VR/IForcing/VR1/factor 400
VR/IForcing/VR1/min-weight 0.1
VR/IForcing/VR1/max-weight 2
VR/IForcing/VR1/materials/2 true
```

Code 66: Interaction forcing configuration for materials

Note that interaction forcing will be applied to all bodies containing an enabled material, even if the corresponding *bodies* parameter is set to *false*.

5.5.1.2 Bremsstrahlung splitting configuration

This technique requires a single parameter to control the number of cloned particles and several others to specify where it is applied, either for bodies (see Code 67) or materials (see Code 68). The parameters are described as follows:

- **splitting**: Sets the *splitting-factor*, which determines the number of times bremsstrahlung photons will be cloned.

- **bodies/body-alias**: Defines the bodies where splitting will be enabled or disabled. To specify a body, add its alias to the *bodies* section and set its value to *true* to *enable* or *false* to *disable* splitting in that body. By default, splitting is set to *false* for all bodies. Multiple body aliases can be specified in this section to control splitting in various bodies. For example, Code 67 shows an example where bodies with aliases *1*, *2*, and *3* are enabled.

- **materials/material-index**: Similar to the *bodies* section, the *materials* section allows the user to enable or disable splitting for specific materials. In this case, the material index is used to specify the material, rather than the body alias, as shown in code block 68.

```
VR/bremss/split4/splitting 4
VR/bremss/split4/bodies/1 true
VR/bremss/split4/bodies/2 true
VR/bremss/split4/bodies/3 true
```

Code 67: Bremsstrahlung splitting configuration for bodies and materials

```
VR/bremss/split2/splitting 2
VR/bremss/split2/materials/1 true
```

Code 68: Bremsstrahlung splitting configuration for bodies and materials

5.5.2 Generic VR techniques

Generic VR techniques can be applied to all particle state types. Each technique, when used in the main PenRed program, follows the configuration pattern shown in code block 69.

```
VR/generic/vr-name/parameter/path value
```

Code 69: Generic VR technique parameter configuration pattern

where ‘vr-name‘ specifies the name of the VR instance for identification purposes. The difference with specific VR techniques, is that generic techniques are applied to the basic particle state. Therefore, all of them can be used on any particle.

5.5.2.1 Splitting

The splitting VR technique works by cloning a particle and reducing its weight by a factor equal to the number of clones created. For example, if a particle with weight $WGHT$ is split into 10 clones (including the original particle), each resulting particle will have a weight of $WGHT/10$. All relevant parameters are listed in Code 70 and are detailed below.

- **type**: Specifies the VR type, which must be set to *”SPLITTING”*.
- **minWght**: Defines the minimum particle weight for applying splitting. Particles with a weight lower than this threshold will not undergo splitting.
- **maxWght**: Defines the maximum particle weight for applying splitting. Particles with a weight higher than this threshold will not undergo splitting.
- **materials/mat-name**: Subsection for configuring splitting parameters for materials. Note that *mat-name* is an identifier within the VR instance and does not correspond to the material name in the geometry configuration. Instead, the material is identified by its integer index, *mat-index*. The number of particle clones for each material is set by the splitting parameter. This parameter can be repeated for different material indexes.
- **bodies/body-alias/splitting**: In this subsection, *body-alias* must identify a valid body within the configured geometry. The specified value of *splitting* will be applied to this body. Note that configurations set in the *bodies* section override those in the *materials* section. This parameter can be repeated for different bodies.

```
VR/generic/vr-name/type "SPLITTING"                          (mandatory)
VR/generic/vr-name/minWght 0.05                              (mandatory)
VR/generic/vr-name/maxWght 21.0                              (mandatory)

VR/generic/vr-name/materials/mat-name/mat-index 2             (optional)
VR/generic/vr-name/materials/mat-name/splitting 20            (optional)

VR/generic/vr-name/bodies/body-alias/splitting 20             (optional)
```

Code 70: Splitting configuration for bodies and materials

A sample configuration is provided in the *8-fake-chamber-vr* example, as it is shown in Code 71

```
VR/generic/splitting/type "SPLITTING"
VR/generic/splitting/minWght 0.05
VR/generic/splitting/maxWght 21.0
VR/generic/splitting/materials/activeVolume/mat-index 2
VR/generic/splitting/materials/activeVolume/splitting 20
```

Code 71: Splitting configuration for bodies and materials

Note that splitting is applied only when a particle enters a specified material or body by crossing an interface. Consequently, splitting does not apply to secondary particles generated within a material, nor is it triggered in bodies without boundary interfaces. To enforce an interface, users can create a detector or assign different materials.

5.5.2.2 Russian roulette

The Russian roulette technique selectively eliminates particles based on a probability defined by the **prob** parameter in the configuration. If a particle survives, its weight is adjusted by multiplying it by a factor of 1/prob. The parameters for the Russian roulette technique are detailed below and summarized in Code 72. An example configuration can be found in the *8-fake-chamber-vr* example.

- **type**: Specifies the VR type, which must be set to *"RUSSIAN_ROULETTE"*.
- **minWght**: Defines the minimum particle weight for applying Russian roulette. Particles with a weight below this threshold will not be affected.
- **maxWght**: Defines the maximum particle weight for applying Russian roulette. Particles with a weight above this threshold will not be affected.
- **materials/mat-name**: Subsection for configuring Russian roulette parameters for materials. Note that *mat-name* is an identifier within the VR instance and does not correspond to the material name in the geometry configuration. Instead, the material is identified by its integer index, *mat-index*. The particle's survival probability is set by the *prob* parameter. This parameter can be specified for multiple material indexes.
- **bodies/body-alias/prob**: Here, *body-alias* identifies a valid body in the configured geometry, and *prob* defines the particle's survival probability within that body. Note that configurations in the *bodies* section override those in the *materials* section. This parameter can be specified for multiple bodies.

```
VR/generic/RR/type "RUSSIAN_ROULETTE"
VR/generic/RR/minWght 0.01
VR/generic/RR/maxWght 19.9
VR/generic/RR/materials/activeVolume/mat-index 1
```

```
VR/generic/RR/materials/activeVolume/prob 0.05
```
Code 72: Russian roulette configuration for bodies and materials

As with the *splitting* technique, Russian roulette is only triggered when a particle crosses an interface.

5.5.3 Specific VR techniques

A specific VR technique is applied to a particular particle state type. Each technique, when integrated within the main PenRed program, follows the configuration pattern shown in code 73.

```
VR/type/vr-name/parameter/path value
```
Code 73: Specific VR technique parameter configuration pattern

Currently, PenRed implements specific VR techniques for photons. Therefore, the only valid value for the *type* field in this context is *photon*.

```
VR/photon/vr-name/parameter/path value
```
Code 74: Photons VR technique parameter configuration pattern

The *vr-name* field specifies a unique identifier for each VR instance and is solely used for identification purposes.

5.5.3.1 X-Ray splitting

X-ray splitting follows a configuration pattern similar to that used for bremsstrahlung. An example with all available parameters is shown in Code 75 and described below:

- **type**: Specifies the VR technique type, which must be set to *"XRAY_SPLITTING"*.
- **bodies/body-alias/splitting**: Defines the bodies where x-ray splitting will be applied. To specify a body, add the *body-alias* to the *bodies* section. The *splitting-factor* value sets the number of particle clones created for each x-ray. Multiple body aliases can be specified in this section to control x-ray splitting in the desired bodies.
- **materials/mat-name**: Similar to the *bodies* section, the *materials* section allows the user to specify x-ray splitting parameters for materials. Note that *mat-name* serves only as an identifier within the VR instance and does not correspond to the material name in the geometry configuration. The material is identified by its integer index, *mat-index*. As with the *bodies* section, the *splitting-factor* parameter sets the number of clones created for each x-ray. This parameter can be specified for different material indexes.

```
VR/photon/vr-name/type "XRAY_SPLITTING"

VR/photon/vr-name/bodies/body-alias/splitting splitting-factor

VR/photon/vr-name/materials/mat-name/mat-index imat
VR/photon/vr-name/materials/mat-name/splitting splitting-factor
```
Code 75: X-ray splitting configuration for bodies and materials

A part of the configuration file corresponding to the *1-disc-vr* example, is shown in Code 76. This one enables x-ray splitting for the body with alias *1* cloning each x-ray produced twice.

```
VR/photon/x-ray/type "XRAY_SPLITTING"
VR/photon/x-ray/bodies/1/splitting 2
```

Code 76: Example of X-ray splitting configuration for bodies

5.6 Simulation parameters

Simulation parameters are specific to the main configuration of the simulation, unlike source, tally, geometry, or material configurations. This section describes the available parameters for the *pen_main* program, which control the overall behavior and execution of the simulation. All parameters follow the pattern of the code 77.

```
simulation/parameter/path value
```

Code 77: Simulation parameters configuration pattern

5.6.1 Dump

PenRed provides the capability to dump the current state of the entire simulation in binary format. This dump can be used to:

- Resume a simulation after computer crash.
- Store the final results in binary format.
- Combine the results of simulations with the same configuration.

To configure this feature, the following parameters are available:

- **dump-interval**: Specify the time, in seconds, between successive dumps
- **dump2read**: Expects a string with the name of a dump file to resume a previous simulation. If multiple threads are used, the simulation must be resumed with the same number of threads as the original simulation. Each thread will attempt to read a file with the specified name prefixed by *thN*, where 'N' is the thread number. For example, if the filename is *dump.dat*, the file for the thread number 2 must be named *th2dump.dat*.
- **dump2write**: Specifies the filename used to save the dumps. Each thread will automatically append the *thN* prefix to the filename, where *N* is the thread number. If not specified, dump files will default to *dump.dat*.
- **dump2ascii**: A boolean value that must be used with the 'dump2read' parameter. If enabled, the simulation will not be resumed. Instead, the program will load the state stored in the specified dump file and extract the tally contents using the usual ASCII data reports. It is disabled by default.
- **finalDump**: A boolean value that forces a dump in each thread at the end of the simulation. It is disabled by default.
- **ascii-results**: A boolean value that enables or disables result reports in ASCII format. If disabled, the result reports will be written to a dump file named *results.dump*. Additionally, no partial results will be printed during the simulation. It is enabled by default.

- **partial-results**: If *ascii-results* is enabled, this boolean value can enable or disable the ASCII printing of partial results. It is disabled by default.

An example of dump configuration can be found in the code 78, where the simulation will resume the stored state at dump file *dump.dat* and store the new generated dumps to *dump2.dat*. New dumps will be generated every 3600 s and the final dump is disabled.

```
simulation/dump-interval 3600
simulation/dump2read "dump.dat"
simulation/dump2write "dump2.dat"
simulation/finalDump false
simulation/dump2ascii false
simulation/ascii-results true
```

Code 78: Dump configuration parameters

5.6.1.1 Adding dumps

Another feature related with dumped simulations is the possibility to add dump files. This can be done via the command line with the option `--addDumps`. To use this option, the user must provide the usual configuration file of the simulation, and all the dump files to be added as follows,

```
./pen_main config.in --addDumps dumpFile1 dumpFile2 dumpFile3
```

where can be specified as many files as required. Notice that no simulation will be carried on when this option is specified in the command line. Also, all added dump files must have been generated using the same configuration. As result, a new dump file named `mergedDump.dump` will be generated with the added information from all specified files. Furthermore, if `ascii-results` is enabled, the corresponding ASCII reports will be saved too.

5.6.2 Threading

The following parameters control the multithreading capabilities of the simulation:

- **threads**: Expects an integer value that specifies the number of threads to be used. If this parameter is set to 0 or less, the number of threads will be calculated automatically according to the value returned by the function,

  `std::thread::hardware_concurrency()`

  By default a single thread is used. In MPI executions, this value is used to specify the default number of threads in each MPI rank.

- **thread-affinity**: This parameter expects a boolean value to enable or disable CPU affinity. This feature is only available in Unix environments and if threads are implemented via the pthreads package. Is disabled by default.

- **simulation/rank/N/threads**: This parameter specifies the number of threads to be used in the MPI rank number N, where N is the desired rank. This parameter overrides the *threads* parameter in that rank. An example is shown in the code 80.

The number of histories to simulate on each source will be distributed among all specified threads.

```
simulation/threads 2
simulation/thread-affinity true
```

Code 79: Number of threads specification

Notice that MPI can be combined with multi-threading. However, the number of MPI processes is not specified in the configuration file. Instead, is specified via the *mpirun* parameters (see section 2). So, when both kinds of parallelism are combined, the number of threads spawned in each MPI rank is calculated according to the parameters specified in the configuration file. This approach "suggests" to the user to create a single MPI process for each node in a distributed memory infrastructure and use threads instead of more MPI processes. With this method, each node will use less memory because data bases will be shared by all threads in the same node. Furthermore, the memory access will be more efficient due to the memory sharing between threads.

In addition, it is possible to specify a different number of threads for each MPI process. For example, the configuration in the code 80 specify two threads for each MPI rank except for the ranks 1 and 2, which will use 4 and 8 threads respectively.

```
simulation/threads 2
simulation/rank/1/threads 4
simulation/rank/2/threads 8
```

Code 80: Number of threads specification

5.6.3 Seeds

The user can specify the initial random generator seeds using the parameters *seed*1 and *seed*2. Alternatively, a seed pair provided by the function *rand0* [13] can be selected via the *seedPair* parameter. Note that in multi-threading and/or MPI simulations, only the *seedPair* parameter can be used to set the initial seeds. This restriction ensures that each random number chain remains statistically independent.

```
simulation/seed1 1
simulation/seed2 1
simulation/seedPair 12
```

Code 81: Initial seeds

By default, the *rand0* function provides 1001 seed pairs. If additional seeds or specific selections are required, a custom seed file can be provided. Each line in this file must contain two integers representing a seed pair, which will be assigned as initial seeds for the simulation threads. To use a custom seed file, specify its path with the *simulation/seeds-file* parameter. For example, to read seeds from a file named *seeds.txt*, add the following line to the configuration file:

```
simulation/seeds-file "seeds.txt"
```

Note: In MPI simulations, each process must have access to an identical copy of the seed file to ensure proper seed assignment.

5.6.4 Simulation time

It is possible to limit the maximum allowed simulation time. If this limit is reached, PenRed will produce the corresponding dump files with the simulation state and finish its execution. To specify that time, only one parameter is required,

- **max-time**: Maximum simulation time in seconds.

For example, the Code 82 shows a configuration line specifying a maximum simulation time of 5 minutes.

```
simulation/max–time 300
```

Code 82: Maximum simulation time

5.7 Logs

By default, logs are saved in two separate files: one for the configuration process, named *config.log*, and another for the simulation, named *simulation.log*. These log files can be modified using the following configuration parameters:

- **log/configuration**: Expects a **string** value, specifying the file where the configuration log will be saved. The default is *config.log*.
- **log/simulation**: Expects a **string** value, specifying the file where the simulation log will be saved. The default is *simulation.log*.

In both cases, if an empty string is provided as the value:

```
log/configuration ""
log/simulation ""
```

the configuration and/or simulation logs will be redirected to the standard output.

5.8 Image output

The provided main program allows to export some of the simulation outputs as images. Which outputs can be exported as images depends on each component implementation, for example, many tallies uses this capability. To enable the image exporter, the field *images* must be included in the *simulation* section followed by the format to enable/disable, as is shown in the code 83. Actually, the two available image formats are *MHD* [14], which is supported by many viewers and can be read with the ITK library, and *GNU*, which consists of a format ready to be plotted as a matrix with the Gnuplot [15] tool. Each format can be enabled or disabled independently of the others. For instance, the code 83 shows how to enable both formats

```
simulation/images/mhd true
simulation/images/gnu true
```

Code 83: Image export

5.9 Load balance parameters

As simulation parameters, load balance parameters are main dependent. This optional feature can be enabled during the ccmake configuration with the option *WITH_LB*. Actually, the only mandatory parameter to configure the load balance system is *balance-interval* (code 84). This one specify the minimum balance time interval in seconds. Specifying this simple parameter, PenRed simulation threads and MPI processes, if enabled, will be balanced automatically.

In addition, PenRed simulations can be balanced using a centralised server to provide a method to balance simulations across the internet. To start a balance server, the PenRed package provides a balance server example code in `src/utilities/LB/LBserver.cpp`. In addition, in the configuration, the user must specify the balance server hostname or IP as string (`loadBalance/host`) and port (`loadBalance/port`). Also, a worker number identification must be specified (`loadBalance/worker`).

Finally, if the SSL support has been enabled at the ccmake configuration with the parameter *WITH_SSL*, the communications between workers and the balance server will be secured with the SSL protocol. This feature requires a set of certificates listed below.

- **CA-cert**: Certification chain file of the trusted CA.
- **cert-file**: Filename of the worker certificate file.
- **key-file**: Filename of the worker key file.
- **key-password**: Worker key password.
- **hostname**: Expected server certificate hostname.

```
loadBalance/balance-interval 500   (mandatory)
loadBalance/host "server-hostname" (optional)
loadBalance/port 5555              (mandatory if enabled "host")
loadBalance/worker 0               (mandatory if enabled "host")
loadBalance/CA-cert                (optional)
loadBalance/cert-file              (mandatory if enabled "CA-cert")
loadBalance/key-file               (mandatory if enabled "CA-cert")
loadBalance/key-password           (optional)
loadBalance/hostname               (optional)
```

Code 84: Load balance parameters

6 Python Wrapper (pyPenred)

The *pyPenred*[6] module provides Python bindings for PenRed, enabling direct integration of radiation transport simulations into Python workflows. These bindings support scripted simulation control, parameter optimization, and embedding within larger applications like the Blender plugin (Section 8.1). The implementation uses pybind11 for efficient C++/Python interoperability.

The API documentation include some examples and can be found online or can be built within the package. Read the corresponding readme at `src/bindings/python/pyPenred` for more information.

6.1 Installation

The simplest installation method is via pip:

```
pip install pyPenred
```

But for systems requiring compilation from source, or to compile user-made modifications, platform-specific scripts are provided:

- Linux/macOS: `installPyPenred.sh`
- Windows: `installPyPenred.bat`

These scripts configure the build with `BUILD_PYTHON_MODULES` enabled, compile and install the pyPenred module following these steps:

[6]`https://pypi.org/project/pyPenred/`

1. Verification of Python 3.8+ development headers (Must be installed in the system).
2. Automatic download of pybind11 if missing.
3. Compilation of the module.
4. Installation via `pip install .` from `src/bindings/python/pyPenred`.

Note: For systems with multiple Python versions, ensure the target Python version's development packages are installed. The binding will compile against the first available Python configuration found by CMake.

6.2 Configuration Format

The wrapper implements a bi-directional conversion between PenRed's native configuration format and Python dictionaries. Parameter paths with slashes (e.g., `/geometry/phantom`) map to nested dictionary structures:

```
/geometry/phantom/material -> config['geometry']['phantom']['material']
```

For example, consider the equivalent configurations in Code 85 (native format) and Code 86 (YAML representation).

```
# Source #
sources/generic/source1/nhist 1.0e9
sources/generic/source1/kpar "gamma"
sources/generic/source1/record-time true

# Directional sampling
sources/generic/source1/direction/type "CONE"
sources/generic/source1/direction/theta 0.0
sources/generic/source1/direction/phi 0.0
sources/generic/source1/direction/alpha 5.0

# Energy sampling
sources/generic/source1/energy/type "MONOENERGETIC"
sources/generic/source1/energy/energy 3.0e7

# Spatial sampling
sources/generic/source1/spatial/type "POINT"
sources/generic/source1/spatial/position/x 0.0
sources/generic/source1/spatial/position/y 0.0
sources/generic/source1/spatial/position/z -25.0

#  GEOMETRY PARAMETERS  #

geometry/type "PEN_QUADRIC"
geometry/input-file "plane.geo"
geometry/processed-geo-file "report.geo"
geometry/kdet/1  1

#          MATERIALS          #

materials/water/number 1
materials/water/eabs/electron 1.0e5
materials/water/eabs/positron 1.0e5
materials/water/eabs/gamma 1.0e4
materials/water/C1 0.05
materials/water/C2 0.05
materials/water/WCC 2.0e3
```

```
materials/water/WCR 2.0e3
materials/water/filename "water.mat"

# SIMULATION PARAMETERS #

simulation/threads 2
simulation/max-time 600
```

Code 85: Configuration with the usual penRed format for the example *2-plane*.

```
geometry:
    input-file: "plane.geo"
    kdet:
        1: 1
    processed-geo-file: "report.geo"
    type: "PEN_QUADRIC"
materials:
    water:
        C1:      5.00000E-02
        C2:      5.00000E-02
        WCC:      2.00000E+03
        WCR:      2.00000E+03
        eabs:
            electron:      1.00000E+05
            gamma:      1.00000E+04
            positron:      1.00000E+05
        filename: "water.mat"
        number: 1
simulation:
    max-time: 600
    threads: 2
sources:
    generic:
        source1:
            direction:
                alpha:      5.00000E+00
                phi:      0.00000E+00
                theta:      0.00000E+00
                type: "CONE"
            energy:
                energy:      3.00000E+07
                type: "MONOENERGETIC"
            kpar: "gamma"
            nhist:      1.00000E+09
            record-time: true
            spatial:
                position:
                    x:      0.00000E+00
                    y:      0.00000E+00
                    z:     -2.50000E+01
                type: "POINT"
```

Code 86: Configuration file in YAML format for the example textit2-plane.

The conversion utilities support:

- Direct loading of native PenRed configuration files
- YAML serialization/deserialization
- Programmatic dictionary manipulation

As shown in Code 87, users can load an existing configuration, modify parameters through dictionary assignment, and export back to standard PenRed format while maintaining full compatibility with standalone simulations. Additionally, the dictionaries can be used directly to configure simulations, as will be shown in the next section.

```python
import pyPenred

# Load PenRed internal or YAML file formats and store it in a dictionary
d = pyPenred.readConfigFile("config.in")

# Modify the dictionary...

# Export the modified dictionary back to PenRed format as string
configText = pyPenred.simulation.dict2SectionString(d)

# Write final configuration to a file
with open("finalConf.in", 'w') as f:
   f.write(configText)
```

Code 87: Load configuration to dictionary.

6.3 Usage

Currently, the *pyPenred* wrapper does not implement all the features of the C++ penRed library. For example, it lacks support for MPI and the capability to sum dumps. However, all core simulation components, such as tallies, geometries, and samplers, are fully functional. As a result, simulations configured for penRed can also be executed using *pyPenred*.

The easiest way to run a simulation is by using the *runFromFile* function. This function starts the simulation and periodically reports its status at regular intervals. It accepts four optional parameters for configuration:

- **configFile**: A string specifying the path to the configuration file. Defaults to *config.in*.
- **statusTime**: Time interval, in seconds, for reporting the simulation status.
- **configLog**: A string specifying the path to the configuration log file. Defaults to *config.log*.
- **simulationLog**: A string specifying the path to the simulation log file. Defaults to *simulation.log*.

For example, if the configuration file *config.in* is located in the current directory, the following Python script can be used to configure and start the simulation:

```python
import pyPenred
pyPenred.runFromFile("config.in")
```

Code 88: How to run a simulation from pyPenred and a penRed configuration file.

In addition to the usual format used by penRed for configurations, this function accepts also *YAML* files. These YAML files must follow the structure shown at the Section 6.2.

The previous example is quite simplistic, as it does not offer much control to the user. However, the definition of the *runFromFile* function (Code 89) can serve as a foundation for building more complex workflows.

```python
def runFromFile(configFile = "config.in",
                statusTime = 20,
                configLog = "config.log",
                simulationLog = "simulation.log"):

    #Set logs
    simulation.setConfigurationLog(str(configLog))
    simulation.setSimulationLog(str(simulationLog))

    #Create simulation object
    simu = simulation.create()

    #Try to get the configuration from yaml or penred internal format
    try:
        f = open(configFile)
        d = yaml.load(f, Loader=yaml.SafeLoader)
        confSetErr = simu.configure(d)
    except:
        confSetErr = simu.configFromFile(configFile)

    if(confSetErr != 0):
        print("Invalid configuration file format. See config log\n")
        return 1

    print("Configuration set\n")

    #Start the simulation asyncronously
    err = simu.simulate(True)
    if(err != 0):
        print("Error on simulation configuration. See config.log\n")
        return 2

    #Simulation started, check status every 30 seconds
    print("Simulation started\n")
    while simu.isSimulating():
        try:
            time.sleep(statusTime)
        except:
            time.sleep(120)
        status = simu.stringifyStatus()
        for e in status:
            print(e)

    print("Simulation finished")
```
Code 89: Function *runFromFile*.

First of all, the function sets the log files for both the configuration and simulation steps:

```python
    #Set logs
    simulation.setConfigurationLog(str(configLog))
    simulation.setSimulationLog(str(simulationLog))
```
Code 90: Function *runFromFile* setting logs.

Next, a simulation object is created, which handles the simulation configuration and execution.

```python
    #Create simulation object
    simu = simulation.create()
```
Code 91: Function *runFromFile* creating simulation object.

To read the configuration from the specified file, the function first attempts to parse it as a YAML format, using the resulting dictionary to configure the simulation object via the method *configure*. If the parsing fails, it then tries to parse the same file using the penRed internal format using the method *configFromFile*. Note that while the parsed configuration is stored in the simulation object, the configuration step has not yet begun. This means that sources, tallies, geometries, and other simulation components are not created at this stage. Consequently, any component-specific configuration errors will not be detected at this point.

```
    #Try to get the configuration from yaml or penred internal format
    try:
        f = open(configFile)
        d = yaml.load(f, Loader=yaml.SafeLoader)
        confSetErr = simu.configure(d)
    except:
        confSetErr = simu.configFromFile(configFile)
```

Code 92: Function *runFromFile* parsing and storing configuration.

Once the configuration is parsed and saved in the simulation object, the simulation process can be initiated. The *pyPenred* wrapper allows for running the simulation in both blocking and asynchronous modes through the *simulate* method of the *simulation* object. This method takes one optional boolean parameter to enable (true) or disable (false) the asynchronous simulation mode. In Code 94, the simulation is executed in asynchronous mode to retrieve its status periodically.

When the *simulate* method is invoked, the stored configuration is utilized to construct all the simulation elements. As a result, this is the point at which potential configuration errors may arise, and they will be logged in the configuration log file.

```
    #Start the simulation asyncronously
    err = simu.simulate(True)
    if(err != 0):
        print("Error on simulation configuration. See config.log\n")
        return 2
```

Code 93: Function *runFromFile* starting simulation.

Finally, the simulation status is checked and reported periodically. The method *isSimulating* returns *True* if a simulation is currently running, or *False* otherwise. It is important to note that the state of the simulation object cannot be changed during a simulation, including the stored configuration. The other used method, *stringifyStatus*, returns a string representation of the current simulation status.

```
    #Simulation started, check status every 30 seconds
    print("Simulation started\n")
    while simu.isSimulating():
        try:
            time.sleep(statusTime)
        except:
            time.sleep(120)
        status = simu.stringifyStatus()
        for e in status:
            print(e)

    print("Simulation finished")
```

Code 94: Function *runFromFile* creating simulation object.

In the previous example, the entire configuration was set using a single dictionary or configuration file. However, it is also possible to override specific component configurations using the following methods from the *simulation* object:

- **configContext**: Sets the context configuration.

- **configSource**: Sets the source configuration.
- **configGeometry**: Sets the geometry configuration.
- **configTally**: Sets the tally configuration.
- **configVR**: Sets the variance reduction configuration.
- **setSimConfig**: Sets the simulation parameters.

Each of these methods accepts a single dictionary as an argument. Note that when configuring a specific component, the global dictionary key used for the entire simulation should be omitted. For instance, the dictionary passed to *configGeometry* should not include a top-level key like *geometry*. Instead, it should directly contain the component-specific settings.

6.4 API

Information about the package's functions and classes can be accessed using the *pydoc* utility. For example, to list the functions and methods of the *simulation* module, run:

```
python -m pydoc pyPenred.simulation
```

7 Examples

PenRed has been tested against the original PENELOPE FORTRAN code, on which it is based, using, among other methods, the provided PENELOPE examples. Some parts of the configuration files of these examples have been shown in previous sections to exemplify the different samplers options, source, materials, geometry and tally definitions. At this section, all of the examples included in the distributed package are briefly described.

To execute the examples, geometry file, configuration file and material files are required. They must be at the same directory. Moreover, the executable file created after the program compilation is needed. As we have explained at section 2, the executable is compiled at *src/compiled/mains/pen_main*. To run the simulation, the executable must be copied at the same folder where the example will be run with the other required files. After that, user can start the execution as follows,

./pen_main path/to/configuration/file

7.1 1-disc

This example presents a point source of electrons and a homogeneous disc phantom of Cu. The source is a monoenergetic beam with energy of $40KeV$ and it is located at $(x, y, z) = (0, 0, -0.0001)$ cm. The phantom size is a radius of 0.01 cm at plane XY and height of 0.005 cm at z axis with the base at $z = 0$ cm.

7.1.1 1-disc-no-vr

This is the first version of the disc example, without variance reduction techniques. The required material files to execute this example is only *Cu.mat* file, and the configuration file is named disc.in. The geometry is read from the geometry file named *disc.geo*. The output of this example execution are the cylindrical and spatial dose distribution, the emerging particles distribution, the impact detector tallies for fluence and energy spectrum and the material energy deposition information.

7.1.2 1-disc-vr

The second version of the disc example includes variance reduction techniques. The material and geometry file are the same than the first version and, although the configuration file has the same name, there are some differences between these two versions. This second one is an example of the interaction forcing, x-ray and bremsstrahlung splitting capabilities of the PenRed code. At the configuration file is specified which is the body that suffers the variance reduction techniques and the kind of particle and interaction to force, electrons in this case. The output of this example execution is the same than at the first version. Notice that, when variance reduction techniques are used, the number of histories of the simulation is reduced because of the splitting applied to them.

7.2 2-plane

This example is a semi-infinite water plane with a spherical detector inside. The source is a photon monoenergetic beam of 30 MeV. The detector located in the water plane is defined as an impact detector for fluence measurements. No VR is applied in the plane example. The single material required to execute this example is the *Water.mat* file. The geometry information is read from the *plane.geo* file. The output of this example are the emerging particles distribution, cylindrical and spatial dose distribution, the impact detector tallies corresponding to fluence and energy spectrum measures and output files for material and body energy deposition.

7.3 3-detector

The geometry of this example consists of a NaI cylindrical detector with 5.08 cm of diameter and 5.08 cm height, with 1.27 cm Fe backing. A point-like Co-60 gamma-ray source emits a photon pencil beam in the $-Z$ direction with equiprobable energies 1.17 and 1.33 MeV. The photons impinge on the NaI crystal from above. No VR is applied in this example. Materials required to execute this example are *NaI.mat* and *Fe.mat* files and the geometry is defined in the *detector.geo* file. In this case, the output files correspond to the material and energy deposition and to the emerging particles distribution.

7.4 4-x-ray-tube

The fourth example corresponds to a simple x-ray generator. It consists of a wolframium anode, a filter of aluminium and a silicon detector. The source emits a monoenergetic electron beam of 150 KeV directed to the anode to produce bremsstrahlung photon beam. For this purpose VR techniques are applied to produce splitting in this body. The filter is added to increase the average energy of the resulting beam, and the detector is defined to measure fluence, energy spectrum, and particle age. Materials required for this example are *W.mat*, *Al.mat*, *Si.mat*. The geometry information is in the *tube.geo* file. The output files are the emerging particles distribution, body and material energy deposition and detector measurements of spectrum, fluence and particle age.

7.5 5-accelerator

The accelerator example simulates a simple electron accelerator and calculates the dose distribution in a water phantom in two steps described following.

7.5.1 5-accelerator-1

In the first step, a phase space file (PSF) is generated at a plane beyond the bottom of the accelerator head, using a planar impact detector. The geometry description of the *accel.geo* file consists of a tungsten target in which impinge an electron beam of 6 MeV, a collimator of the same material and a water phantom. The whole geometry is defined inside an air enclosure. Materials needed to run this simulation are *W.mat*, *H2O.mat* and *Air.mat*. The output files are the body and material energy deposition, the emerging particles distribution and the impact detector spectrum. Moreover, a *psf-merged.dat* file is generated with the PSF information.

7.5.2 5-accelerator-2

In the second step, the initial particle states from that PSF is read and the dose distribution in the water phantom is obtained. The same materials and geometry file are used in this second step. In this case the output files are the emerging particle distribution, body and material energy deposition and the spatial dose distribution in the water phantom.

7.5.3 5-accelerator-3

This example has been created to test the phase space file translation and rotation capabilities. Two variants of this example are created, 5-accelerator-orig and 5-accelerator-rot_trans. The geometry used, *accel.geo*, consists of the same geometry of the previous example `5-accelerator-2` but with some modifications. In this case the water phantom is composed by different water slices with 2 cm of thickness instead of a solid water cube. In the first variation of this example `5-accelerator-orig`, the phase space file and the geometry of the problem have not been rotated nor translated. An `EDEP_BODY` tally type has been set to obtain the deposited energy in each body of the geometry, including the different water slices. Moreover, three impact detector have been created with the tally type `IMPACT_DET` to obtain fluence and spectrum.

In the second variation, `5-accelerator-rot_trans`, the same geometry is used, but a rotation and translation is applied.

- Rotation:
  - omega = 20 degrees
  - theta = 45 degrees
  - phi = 20 degrees
- Translation:
  - y shift = 10 cm

The same rotation and translation have been applied to the phase space file in the config.in file. The same EDEP_BODY and IMPACT_DET tallies used in the first variation have been set

7.6 6-polarisation

This example reproduces the Namito et al.'s (1993) scattering experiment with polarised photons. The source of this example is a monoenergetic polarised photon beam of 40 KeV, defined using the stokes parameters. Interaction forcing is used in this example. The geometry file corresponds to *gpol.geo* and the material files are *Cu.mat* and *Vacuum.mat*. The output files of the execution of this example are the emerging particle distribution, the material energy deposition and the energy spectrum.

7.7 7-aba

This example consist of a ^{60}Co gamma rays source on a three layer cylinder phantom of water-Al-water. The source has equiprobable energies of 1.17 and 1.33 MeV. No VR is applied in this example. The geometry description of the cylinders is defined in the *3discs.geo*, and the material files needed are *water.mat* and *Al.mat*. The output files of this example corresponds to the emerging particles distribution, the impact detector energy deposition, the spatial dose distribution and the material and body energy deposition.

7.8 8-fake-chamber

This example describes an ionization chamber geometry inside a water phantom of $30 \times 30 \times 30$ cm^3. The source is a monoenergetic photon beam of 4 MeV. The source-surface distance between the source and the surface of the water phantom is 90 cm. The ionization chamber is located 10 cm from the water surface. The geometry file of this example is the *chamber.geo* and the materials used for its simulation are *air.mat*, *pmma.mat*, *water.mat*. The same files are used for both versions of this example, without and with VR respectively.

7.8.1 8-fake-chamber-novr

In the first version of this example no VR is applied. The output files are the spatial dose distribution and the material energy deposition.

7.8.2 8-fake-chamber-vr

In the second version of this example VR is applied. Splitting and russian roulette are used. The output files are the same that in the first version, the spatial dose distribution and the material energy deposition.

8 Utilities

This section describes the utilities provided with the PenRed package that can be useful for the user. The code of each utility described in this section can be found in the corresponding folder in the directory

```
src/utilities/
```

8.1 Blender Plugin

This utility consists of a Python plugin designed for use within the Blender[7] software. The plugin utilizes Blender's UI and 3D design tools to streamline the creation of both Quadric

[7]`https://www.blender.org/`

(see Section 5.3.2) and Triangular Mesh Surface (see Section 5.3.3) geometries. Furthermore, the plugin includes functionality to configure a complete simulation directly within the Blender UI, enabling users to export both the geometry file and the configuration file. The plugin is located in the following directory:

```
src/utilities/Blender
```

The directory contains the latest version of the plugin, which is compatible with Blender LTS version 4.2.3[8]. Within the plugin directory, the folder corresponding to the specific Blender version includes the plugin source code, a zip file named *penred.zip* (ready for installation via Blender's add-on system), and the plugin documentation source. The documentation is also available online at `https://penred.github.io/PenRed/`. When constructing a geometry in Blender for export to PenRed, it is important to note that Blender's default units are already interpreted as centimeters. As a result, no additional unit transformations are required. Specific considerations for each geometry type will be discussed in the following sections. However, this manual does not provide an introduction to Blender usage, as it is extensively documented elsewhere, and numerous tutorials are available online.

To use this plugin effectively, users are expected to have a basic understanding of Blender. Key skills include navigating the 3D viewport, creating and modifying objects, establishing relationships between objects, editing meshes, and installing add-ons.

8.2 Mutren

The Mutren utility is based on the original `mutren.f` code of the PENELOPE package. This one, calculates the μ_{en} coefficients of the specified material for the specified energies. It consists of a command line program which takes the following parameters as input,

./mutren material-file energy-spectrum-file tolerance sim-time

Each parameter is described following,

- **material-file**: Path to the input material file.
- **energy-spectrum-file**: Path to the energy spectrum file. The format of this file consists of rows where each one contains a single energy in eV.
- **tolerance**: Relative error to stop the simulation, usually set to 0.1%
- **sim-time**: Allowed time to simulate each provided energy in seconds.

The program will calculate the coefficients for each provided energy and store them in a file named `mutren.dat`. Notice that the simulation of each energy point will finish if the objective tolerance has been achieved or if the simulation time reaches the specified limit (**sim-time**).

8.3 iaeaPSF

PenRed writes and reads the PSF in its own binary format. However, the International Atomic Energy Agency (IAEA) provides a standard PSF format. To be able to run IAEA PSF with PenRed, in the utilities folder, user can found iaeaPSF folder which includes the

[8] `https://www.blender.org/download/releases/4-2/`

IAEA package of routines needed to obtain *iaea2PenRed* and *penRed2iaea* conversion tools. These tools can be found at

src/compiled/iaeaPSF/

The compilation can be set ON or OFF in the ‘compile.sh’ file. On the one hand, the *iaea2PenRed* program allows to convert PSF in IAEA format to PSF in internal binary PenRed format. To do it, user needs both header and phase space file of the IAEA standard format, “.IAEAheader” and “.IAEAphsp” respectively. To execute this tool the command line needed is

./iaea2PenRed input filename (without extension) and output file name

On the other hand, the *penRed2iaea* program converts a PenRed PSF in to IAEA PSF. In this case the comand line to execute it is

./penRed2iaea input filename and output file name (without extension)

providing two new files with extensions “.IAEAheader” and “.IAEAphsp”, respectively.

8.4 geo2voxel

This utility converts any geometry type in a voxelized one. As program arguments, the `geo2voxel` requires a configuration file which format is the same as `pen_main`, i.e. the format described in the section 3. This configuration file, must include all the configuration parameters to load a geometry of any type, following the descriptions provided in the section 5.3, and the parameters listed below,

- **voxelized/nx**: Number of voxels in the X axis.
- **voxelized/ny**: Number of voxels in the Y axis.
- **voxelized/nz**: Number of voxels in the Z axis.
- **voxelized/dx**: Voxel size in the X axis, in cm.
- **voxelized/dy**: Voxel size in the Y axis, in cm.
- **voxelized/dz**: Voxel size in the Z axis, in cm.
- **voxelized/ox**: Voxelized geometry origin in the X axis, in cm.
- **voxelized/oy**: Voxelized geometry origin in the Y axis, in cm.
- **voxelized/oz**: Voxelized geometry origin in the Z axis, in cm.
- **voxelized/granularity**: Granularity used to determine the material and density of each voxel.
- **materials/material-name/ID**: Identification number of the material. There the *material-name* must be specified by the user and it is used only to distinguish different materials in the configuration file, i.e. has no influence in the converted geometry. This parameter must be repeated for different materials.

- **materials/material-name/density**: Density value of each material of the geometry, in g/cm^3. This parameter must be repeated for different materials. This density is assigned to the material number with the same *material-name*.

where *granularity* and the number of bins are expected to be integers, and the remaining parameters real numbers. The origin of the voxelized geometry (ox, oy, oz) is interpreted as the position of the left bottom corner of the lower Z voxelized geometry plane i.e., is not considered as the center of the first voxel of the mesh. Then, the granularity specify how many points per axis are generated inside each voxel to determine the material and density of each one. The density is obtained from the mean value of all generated points, while the material assignation will be done according to the material index with more points inside the voxel.

An example of configuration to generate a voxelized geometry from a quadric geometry file is shown in the code 95, where 4^3 points are generated in each voxel to determine the material and density and the center of the voxelized mesh has been set to the $(0, 0, 0)$ of the quadric geometry.

```
geometry/type "PEN_QUADRIC"
geometry/input-file "disc.geo"
geometry/processed-geo-file "report.geo"

voxelized/nx 300
voxelized/ny 300
voxelized/nz 120

voxelized/dx 0.1
voxelized/dy 0.1
voxelized/dz 0.2

voxelized/ox -15.0
voxelized/oy -15.0
voxelized/oz -12.0

voxelized/granularity 4

materials/water/ID 1
materials/water/density 0.99821
```

Code 95: Complete `geo2voxel` configuration example to convert a quadric geometry.

8.5 geo2mesh

The 'geo2mesh' utility converts any geometry file compatible with PenRed into a simplified Wavefront file that can be used with the PenRed-GUI program. To accomplish this, the program creates a regular voxelized mesh with material information, allowing for the creation of the final triangular surface boundary. To use this utility, the user needs an input configuration file with the following parameters:

- **geometry**: This section contains information to configure the geometry, and its contents vary depending on the geometry type. It is structured as the used for a simulation.

- **nx**: The number of voxels used in the X-axis.

- **ny**: The number of voxels used in the Y-axis.

- **nz**: The number of voxels used in the Z-axis.

- **dx**: The voxel size in the X-direction, measured in centimeters.
- **dy**: The voxel size in the Y-direction, measured in centimeters.
- **dz**: The voxel size in the Z-direction, measured in centimeters.
- **ox**: The X-coordinate of the mesh origin, measured in centimeters.
- **oy**: The Y-coordinate of the mesh origin, measured in centimeters.
- **oz**: The Z-coordinate of the mesh origin, measured in centimeters.
- **granularity**: The number of points created within each voxel along each axis to assign the material number. It is advisable to use lower values to minimize computational costs. Typically, a value of 2 or 3 is sufficient.
- **smooth/steps**: The number of smoothing steps to apply to the mesh. This parameter is optional and is set to 50 by default.
- **smooth/factor**: The strength of the smoothing filter. This is also optional and is set to 0.2 by default.

Notice that the mesh origin is located at the first corner in the direction of increasing coordinates of the first mesh voxel. To use the configuration file, run the program as follows:

```
./geo2mesh config-file
```

the program will generate two files: one containing vertex information named 'geo.obj' and another with material information named 'geoMats.mtl'. The code 96 shows an example of configuration file to convert the 'tube.geo' geometry of the quadric example to a mesh.

```

##########################
#   GEOMETRY PARAMETERS   #
##########################

geometry/type "PEN_QUADRIC"
geometry/input-file "tube.geo"
geometry/processed-geo-file "report.geo"

##########################
#   GEO2MESH PARAMETERS   #
##########################

ox -10
oy -10
oz -10

dx 0.1
dy 0.1
dz 0.1

nx 200
ny 200
nz 200

granularity 3

smooth/steps 20
```

```
smooth/factor 0.2
```
Code 96: Complete `geo2mesh` configuration example to convert a quadric geometry.

8.6 range

The `range` utility prints the electron, positron and photon ranges for the specified material and energies. The corresponding code is located in the folder `penMats`. The program must be executed via the command line as follows,

```
./range material-file E1 E2 E3
```

where *range* is the executable name, *material-file* is the material file with the material information, the same as the used for simulations, and the following arguments ($E1, E2, E3$) are a list of energies to calculate the corresponding ranges. Notice that in the execution example we have used 3 energies. However, any number of energies can be used.

The interpretation of the returned range depends on the particle type:

- **gamma**: The range is defined as the particle mean free path (inverse attenuation coefficient), in cm.
- **electrons and positrons**: The range is interpreted as the CSDA range, in cm.

8.7 PSF spectrum

The `psf_spectre` utility extracts the energy spectrum distribution of the particles registered in a PSF file, and the corresponding source code can be found in the `psf/psf_spectre.cpp` file. This utility is used via the command line as follows,

```
./psf_spectre PSFfilename emin emax nbins
```

where *psf_spectre* is the utility executable, *PSFfilename* the path to the PSF file to be processed, *emin* the minimum energy, in eV, of the extracted spectrum, *emax* the corresponding spectrum maximum energy and *nbins* the number of spectrum bins.

8.8 PSF to ASCII

The `psf2ascii` utility converts a binary PSF file into a ASCII file format. The source code can be found in `psf/psf2ascii.cpp` and must be used via the command line as follows,

```
./psf2ascii inputFile outputFile
```

where *psf2ascii* is the executable, *inputFile* the binary PSF file path and *outputFile* the filename of the generated ASCII file.

8.9 Registers

The `registers` folder contains a set of sources which compiled programs show all the available types of a specific component. The shown types correspond to the ones used in the configuration files to specify the component *type* parameter. All the available registers are summarised in the Table 9. This ones are executed with no arguments via the command line.

Executable	Registered information
`regGenericVR`	Generic VR types
`regPhotonVR`	Specific photon VR types
`regGeometries`	Geometry types
`regSamplers`	Particle sampler types
`regTallies`	Tally types

Table 9: Available register programs with the corresponding registered information.

8.10 CompositionsDB

The *compositionsDB* utility provide access to the composition databases (DB). If no arguments are provided, the names of all available databases are listed (Code 97):

```
./compositionsDB
usage: ./compositionsDB [composition-DB] [material-name]
Registered data bases:
 + ICRP_AF
 + ICRP_AM
```

Code 97: Composition utility DB listing.

For now, the two available databases are the ICRP compositions for the adult female and male phantoms, *ICRP_AF* and *ICRP_AM*, respectively [16]. If a database is specified, all materials belonging to it are listed. (Code 98):

```
./compositionsDB ICRP_AF
usage: ./compositionsDB [composition-DB] [material-name]
Registered compositions in database 'ICRP_AF':
  + Adrenal_left
  + Adrenal_right
  + ET1(0-8)
  + ET1(8-40)
  + ET1(40-50)
  + ET1(50-surface)
  + ET2(-15-0)
  + ET2(0-40)
  + ET2(40-50)
  + ET2(50-55)
  + ET2(55-65)
  + ET2(65-surface)
  + Oral_mucosa_tongue
  + Oral_mucosa_mouth_floor
  + Oral_mucosa_lips_and_cheeks
  + Trachea
  + BB(-11--6)
  + BB(-6-0)
  + BB(0-10)
  .
  .
  .
```

Code 98: Composition utility: Listing ICRP_AF materials.

Finally, if both the database and material name are provided, the density and composition of the specified material are shown. The composition is specified by pairs (atomic number,fraction by weight), as is shown in the Code 99:

```
./compositionsDB ICRP_AF Humeri_upper_cortical
usage: ./compositionsDB [composition-DB] [material-name]

```

```
  + Material 'Humeri_upper_cortical' (     1.90400E+00 g/cm**3):
     -   1   0.036000
     -   6   0.159000
     -   7   0.042000
     -   8   0.448000
     -  11   0.003000
     -  12   0.002000
     -  15   0.094000
     -  16   0.003000
     -  20   0.213000
```

Code 99: Composition utility DB: Listing material composition.

The user can specify multiple material names in a single execution, obtaining the corresponding composition for each one.

8.11 internalData2YAML

This utility converts a file with the penRed internal data format to a YAML format file. In the process, each slash is converted to a new level in the YAML structure.

```
usage: ./internalData2YAML internalData-filename outputYAML
```

Code 100: internalData2YAML usage.

A Predefined materials

ID	Name	ID	Name	ID	Name
1	Hydrogen	34	Selenium	67	Holmium
2	Helium	35	Bromine	68	Erbium
3	Lithium	36	Krypton	69	Thulium
4	Beryllium	37	Rubidium	70	Ytterbium
5	Boron	38	Strontium	71	Lutetium
6	Amorphous carbon	39	Yttrium	72	Hafnium
7	Nitrogen	40	Zirconium	73	Tantalum
8	Oxygen	41	Niobium	74	Tungsten
9	Fluorine	42	Molybdenum	75	Rhenium
10	Neon	43	Technetium	76	Osmium
11	Sodium	44	Ruthenium	77	Iridium
12	Magnesium	45	Rhodium	78	Platinum
13	Aluminium	46	Palladium	79	Gold
14	Silicon	47	Silver	80	Mercury
15	Phosphorus	48	Cadmium	81	Thallium
16	Sulfur	49	Indium	82	Lead
17	Chlorine	50	Tin	83	Bismuth
18	Argon	51	Antimony	84	Polonium
19	Potassium	52	Tellurium	85	Astatine
20	Calcium	53	Iodine	86	Radon
21	Scandium	54	Xenon	87	Francium
22	Titanium	55	Cesium	88	Radium
23	Vanadium	56	Barium	89	Actinium
24	Chromium	57	Lanthanum	90	Thorium
25	Manganese	58	Cerium	91	Protactinium
26	Iron	59	Praseodymium	92	Uranium
27	Cobalt	60	Neodymium	93	Neptunium
28	Nickel	61	Promethium	94	Plutonium
29	Copper	62	Samarium	95	Americium
30	Zinc	63	Europium	96	Curium
31	Gallium	64	Gadolinium	97	Berkelium
32	Germanium	65	Terbium	98	Californium
33	Arsenic	66	Dysprosium	99	Einsteinium

Table 10: Elements predefined materials extracted from [3]. Material ID corresponds to the element atomic number.

ID	Name
100	Acetone
101	Acetylene
102	Adenine
103	Adipose tissue (ICRP)
104	Air, dry (near sea level)
105	Alanine
106	Aluminum oxide
107	Amber
108	Ammonia
109	Aniline
110	Anthracene
111	B-100 bone-equivalent plastic
112	Bakelite
113	Barium fluoride
114	Barium sulfate
115	Benzene
116	Beryllium oxide
117	Bismuth germanium oxide
118	Blood (ICRP)
119	Bone, compact (ICRU)
120	Bone, cortical (ICRP)
121	Boron carbide
122	Boron oxide
123	Brain (ICRP)
124	Butane
125	N-butyl alcohol
126	C-552 air-equivalent plastic
127	Cadmium telluride
128	Cadmium tungstate
129	Calcium carbonate
130	Calcium fluoride
131	Calcium oxide
132	Calcium sulfate
133	Calcium tungstate
134	Carbon dioxide
135	Carbon tetrachloride
136	Cellulose acetate, cellophane
137	Cellulose acetate butyrate
138	Cellulose nitrate
139	Ceric sulfate dosimeter solution
140	Cesium fluoride
141	Cesium iodide
142	Chlorobenzene
143	Chloroform
144	Concrete, portland
145	Cyclohexane
146	1,2-dichlorobenzene
147	Dichlorodiethyl ether
148	1,2-dichloroethane

149	Diethyl ether
150	N,n-dimethyl formamide
151	Dimethyl sulfoxide
152	Ethane
153	Ethyl alcohol
154	Ethyl cellulose
155	Ethylene
156	Eye lens (ICRP)
157	Ferric oxide
158	Ferroboride
159	Ferrous oxide
160	Ferrous sulfate dosimeter solution
161	Freon-12
162	Freon-12b2
163	Freon-13
164	Freon-13b1
165	Freon-13i1
166	Gadolinium oxysulfide
167	Gallium arsenide
168	Gel in photographic emulsion
169	Pyrex glass
170	Glass, lead
171	Glass, plate
172	Glucose
173	Glutamine
174	Glycerol
175	Graphite
176	Guanine
177	Gypsum, plaster of Paris
178	N-heptane
179	N-hexane
180	Kapton polyimide film
181	Lanthanum oxybromide
182	Lanthanum oxysulfide
183	Lead oxide
184	Lithium amide
185	Lithium carbonate
186	Lithium fluoride
187	Lithium hydride
188	Lithium iodide
189	Lithium oxide
190	Lithium tetraborate
191	Lung (ICRP)
192	M3 wax
193	Magnesium carbonate
194	Magnesium fluoride
195	Magnesium oxide
196	Magnesium tetraborate
197	Mercuric iodide
198	Methane

199	Methanol
200	Mixed wax
201	Ms20 tissue substitute
202	Muscle, skeletal (ICRP)
203	Muscle, striated (ICRU)
204	Muscle-equivalent liquid, with sucrose
205	Muscle-equivalent liquid, without sucrose
206	Naphthalene
207	Nitrobenzene
208	Nitrous oxide
209	Nylon, du Pont elvamide 8062
210	Nylon, type 6 and type 6/6
211	Nylon, type 6/10
212	Nylon, type 11 (rilsan)
213	Octane, liquid
214	Paraffin wax
215	N-pentane
216	Photographic emulsion
217	Plastic scintillator (vinyltoluene based)
218	Plutonium dioxide
219	Polyacrylonitrile
220	Polycarbonate (makrolon, lexan)
221	Polychlorostyrene
222	Polyethylene
223	Polyethylene terephthalate (mylar)
224	Polymethyl methacrilate (lucite, perspex, plexiglass)
225	Polyoxymethylene
226	Polypropylene
227	Polystyrene
228	Polytetrafluoroethylene (teflon)
229	Polytrifluorochloroethylene
230	Polyvinyl acetate
231	Polyvinyl alcohol
232	Polyvinyl butyral
233	Polyvinyl chloride
234	Polyvinylidene chloride (saran)
235	Polyvinylidene fluoride
236	Polyvinyl pyrrolidone
237	Potassium iodide
238	Potassium oxide
239	Propane
240	Propane, liquid
241	N-propyl alcohol
242	Pyridine
243	Rubber, butyl
244	Rubber, natural
245	Rubber, neoprene
246	Silicon dioxide
247	Silver bromide
248	Silver chloride

249	Silver halides in photographic emulsion
250	Silver iodide
251	Skin (ICRP)
252	Sodium carbonate
253	Sodium iodide
254	Sodium monoxide
255	Sodium nitrate
256	Stilbene
257	Sucrose
258	Terphenyl
259	Testes (ICRP)
260	Tetrachloroethylene
261	Thallium chloride
262	Tissue, soft (ICRP)
263	Tissue, soft (ICRU four-component)
264	Tissue-equivalent gas (methane based)
265	Tissue-equivalent gas (propane based)
266	Tissue-equivalent plastic (A-150)
267	Titanium dioxide
268	Toluene
269	Trichloroethylene
270	Triethyl phosphate
271	Tungsten hexafluoride
272	Uranium dicarbide
273	Uranium monocarbide
274	Uranium oxide
275	Urea
276	Valine
277	Viton fluoroelastomer
278	Water, liquid
279	Water vapour
280	Xylene

Table 11: Predefined compounds extracted from [3].

References

[1] V. Giménez-Alventosa, V. Giménez Gómez, S. Oliver, Penred: An extensible and parallel monte-carlo framework for radiation transport based on penelope, Computer Physics Communications 267 (2021) 108065. `doi:https://doi.org/10.1016/j.cpc.2021.108065`.
URL `https://www.sciencedirect.com/science/article/pii/S0010465521001776`

[2] Salvat F., Penelope. a code system for monte carlo simulation of electron and photon transport, Issy-Les-Moulineaux: OECD Nuclear Energy Agengy (2014).

[3] F. Salvat, PENELOPE-2018: A code System for Monte Carlo Simulation of Electron and Photon Transport, OECD/NEA Data Bank, Issy-les-Moulineaux, France, 2019, available from `http://www.nea.fr/lists/penelope.html`.

[4] Dcmtk, `https://github.com/DCMTK/dcmtk`, accessed: 2019-09-28.

[5] R. L. Graham, T. S. Woodall, J. M. Squyres, Open mpi: A flexible high performance mpi, in: R. Wyrzykowski, J. Dongarra, N. Meyer, J. Waśniewski (Eds.), Parallel Processing and Applied Mathematics, Springer Berlin Heidelberg, Berlin, Heidelberg, 2006, pp. 228–239.

[6] W. Gropp, Mpich2: A new start for mpi implementations, in: Proceedings of the 9th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface, Springer-Verlag, Berlin, Heidelberg, 2002, p. 7.

[7] J. Sempau, A. Badal and L. Brualla, A PENELOPE-based system for the automated Monte Carlo simulation of clinacs and voxelized geometries–application to far-from-axis fields, Med. Phys. 38 (2011) 5887–5895.
URL `http://dx.doi.org/10.1118/1.3643029`

[8] J. Sempau, Penelope/peneasy user manual, Tech. rep., Version 2019-01-01.

[9] E. García-Toraño, V. Peyres, F. Salvat, Pennuc: Monte carlo simulation of the decay of radionuclides, Computer Physics Communications 245 (2019) 106849. `doi:https://doi.org/10.1016/j.cpc.2019.08.002`.
URL `https://www.sciencedirect.com/science/article/pii/S0010465519302267`

[10] V. Giacometti, S. Guatelli, M. Bazalova-Carter, A. Rosenfeld, R. Schulte, Development of a high resolution voxelised head phantom for medical physics applications, Physica Medica 33 (2017) 182 – 188. `doi:https://doi.org/10.1016/j.ejmp.2017.01.007`.
URL `http://www.sciencedirect.com/science/article/pii/S1120179717300078`

[11] J. F. Williamson, Monte carlo evaluation of kerma at a point for photon transport problems, Medical Physics 14 (4) (1987) 567–576. `arXiv:https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1118/1.596069`, `doi:10.1118/1.596069`.
URL `https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.596069`

[12] N. Ravinder et al, Dosimetry of interstitial brachytherapy sources: recommendations of the AAPM radiation therapy committee task group no 43, Med. Phys. 22 (1995) 209–34. `doi:10.1118/1.597458`.

[13] A. Badal, J. Sempau, A package of linux scripts for the parallelization of monte carlo simulations, Computer Physics Communications 175 (6) (2006) 440 – 450. `doi:https://doi.org/10.1016/j.cpc.2006.05.009`.
URL `http://www.sciencedirect.com/science/article/pii/S001046550600230X`

[14] Mhd/raw image format, `https://itk.org/Wiki/ITK/MetaIO/Documentation`, accessed: 2022-07-15.

[15] Gnuplot, `http://www.gnuplot.info/`, accessed: 2022-07-15.

[16] M. Zankl, Adult male and female reference computational phantoms (icrp publication 110), Japanese Journal of Health Physics 45 (4) (2010) 357–369.