

PenRed: User Manual

V. Giménez-Alventosa¹, V. Giménez Gómez², and S. Oliver³

¹Instituto de Instrumentación para Imagen Molecular (I3M), Centro mixto CSIC - Universitat Politècnica de València, Camí de Vera s/n, 46022, València, Spain, vicent.gimenez@i3m.upv.es

²Departament de Física Teòrica and IFIC, Universitat de València-CSIC, Dr. Moliner, 50, 46100, Burjassot, València, Spain, vicente.gimenez@uv.es

³Instituto de Seguridad Industrial, Radiofísica y Medioambiental (ISIRYM), Universitat Politècnica de València, Camí de Vera s/n, 46022, València, Spain, sanolgi@upvnet.upv.es

October 6, 2021

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 found in the PenRed implementation manual and component manual (in development) respectively.

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 unique physics models and tracking algorithms for charged particles. For further details on the interactions models, the reader is referred to the excellent PENELOPE manual [3]. PenRed has been coded in C++ following an object-oriented

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>

programming paradigm restricted to the C++11 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, the examples located in the **examples** folder, which are ready to be simulated, are described. Finally, the offered utilities in the PenRed package and how to use them are explained in the section 7.

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

Contents

1	Introduction	1
1.1	What is PenRed?	1
1.2	Document structure	2
2	Installation and execution	5
2.1	Linux instalation	5
2.2	Windows instalation	6
2.3	Basic usage	10
2.4	Optional features	10
3	Internal data format	11
3.1	Implementation	11
4	Constants, parameters and definitions	12
4.1	Important definitions	12
4.2	Units	14
4.3	Particle indexes	14
4.4	Particle interaction indexes	15
4.5	Atomic electron shells	15
5	Framework usage	16
5.1	Materials	16
5.1.1	Configuration	17
5.2	Particle sources	18
5.2.1	Spatial source samplers	20
5.2.2	Direction samplers	21
5.2.3	Energy samplers	21
5.2.4	Time samplers	23
5.2.5	Specific Sources	23
5.3	Geometries	24
5.3.1	Quadric	24
5.3.2	Voxel	25
5.3.3	DICOM	26
5.4	Tallies	28
5.4.1	Radial and Cylindrical Dose Distribution	29
5.4.2	Emerging Particles Distribution	30
5.4.3	Energy Deposition Body	31
5.4.4	Energy Deposition Material	31
5.4.5	Impact Detector	31
5.4.6	Spatial Dose Distribution	33
5.4.7	Angular Detector	34
5.4.8	Particle Generation	35
5.4.9	Spherical Dose Distribution	35
5.4.10	Phase Space File (PSF)	36
5.4.11	Kerma track length estimator	37
5.5	Variance Reduction	38
5.5.1	Context specific	39
5.5.2	Specific VR techniques	40

5.5.3	Generic VR techniques	40
5.6	Simulation parameters	42
5.7	Load balance parameters	43
6	Examples	44
6.1	1-disc	44
6.1.1	1-disc-no-vr	44
6.1.2	1-disc-vr	44
6.2	2-plane	45
6.3	3-detector	45
6.4	4-x-ray-tube	45
6.5	5-accelerator	45
6.5.1	5-accelerator-1	45
6.5.2	5-accelerator-2	46
6.5.3	5-accelerator-3	46
6.6	6-polarisation	46
6.7	7-aba	47
6.8	8-fake-chamber	47
6.8.1	8-fake-chamber-novr	47
6.8.2	8-fake-chamber-vr	47
7	Utilities	47
7.1	Mutren	47
7.2	iaeaPSF	48
7.3	geo2voxel	48
7.4	range	49
7.5	PSF spectrum	49
7.6	PSF to ASCII	50
7.7	Registers	50
A	Predefined materials	51

2 Installation and execution

Some PenRed features depends on few external libraries which should be installed to activate the corresponding features (see below). It is the user's responsibility to check that these external dependencies are installed before activating a feature. The user should refer to the appropriate installation guides of the optional packages.

The recommended configuration for compiling PenRed is the following:

- g++ with version greater than GCC 4.7.3 with support for C++ 11 standard. However, PenRed can also be compiled with Intel icc version 2019, clang version 8 and MSVS 2019.
- CMake minimal version 3.4.

2.1 Linux instalation

The installation of PenRed is very easy. First, download PenRed sources from our GitHub repository,

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

The code must be compiled in the `src` folder, which includes a CMake file and a bash script (`compile.sh`) to simplify the installation and compile the code automatically. In this script, you can enable/disable the following main optional features,

- DICOMs: If it is enabled, PenRed capabilities to read and simulate DICOM images will be active. This option requires the library dicom toolkit (dcmtk) [4].
- Multi-threading: This option enables multi-threading capabilities. PenRed implements multi-threading via the standard thread library specified in the C++11 standard. Thus, it is not required any extra library 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 system between threads and MPI processes. This option requires, as least, multi-threading capabilities, because uses threads to handle MPI communications.

Notice that all previous dependencies are optional. The corresponding libraries can be found at most linux package repositories. For example, to compile PenRed with DICOM support in Fedora, you can use the `dnf` command to install the dependencies,

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

Once the dependencies are installed, the compilation can be done using the provided script,

```
bash compile.sh
```

or doing it yourself,

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

```
mkdir build

cd build

ccmake ../

make -jN (with N the number of processes)

make install
```

With `ccmake` you can configure the optional PenRed features with a more friendly interface. But, of course, you can use directly `cmake ../` defining the appropriate flags like,

```
cmake -DWITH_DICOM="ON" -DWITH_MULTI_THREADING="ON"
-DWITH_MPI="OFF" -DWITH_LB="OFF" -DDEVELOPMENT_WARNINGS="OFF" ../
```

Once the code has been compiled, the user can found the executable of our provided main program ready to simulate at,

src/compiled/mains/pen_main

2.2 Windows instalation

To compile the PenRed code using MSVS on windows, first, select “Clonea repository” from the Visual Studio start window (Figure 1).

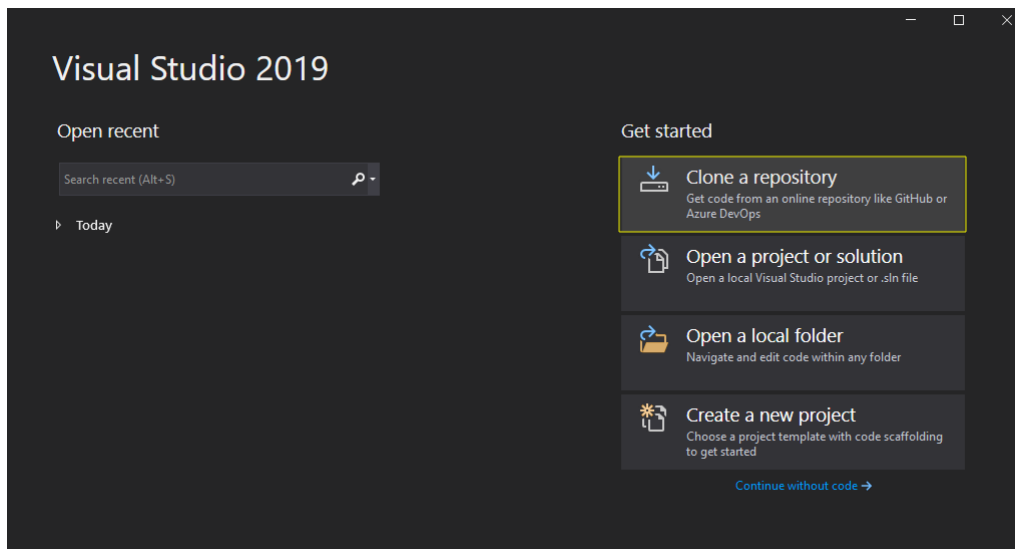


Figure 1: Clone a repository in MSVS

Then, set the PenRed repository url and push on the clone button (Figure 2).

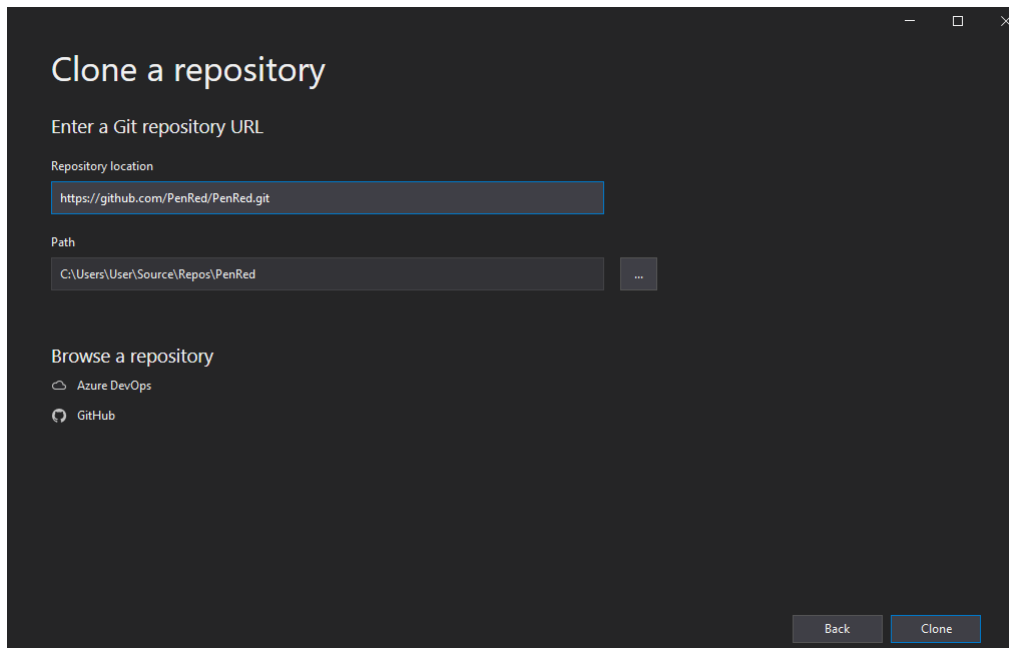


Figure 2: Clone PenRed repository with MSVS

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 3).

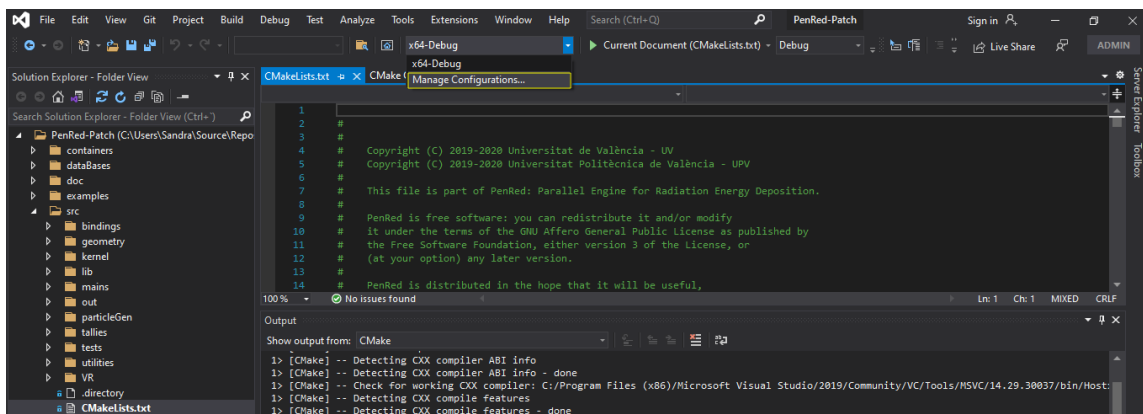


Figure 3: Manage MSVS configurations

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 4).

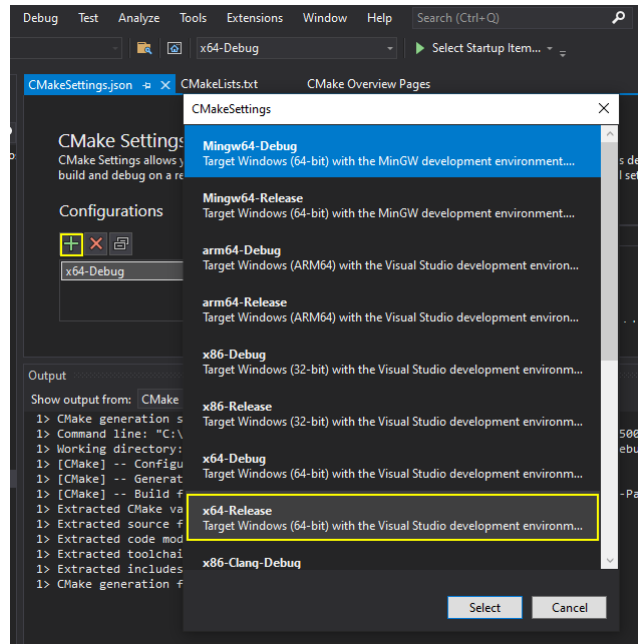


Figure 4: Add configuration in MSVS

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 5).

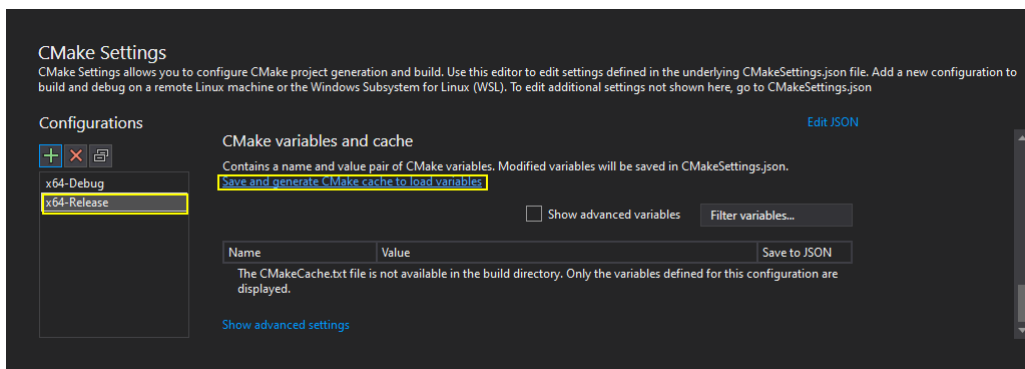


Figure 5: Select configuration in MSVS

Now, we can change all the Cmake configuration variables to enable or disable MPI support and other features (Figure 6).

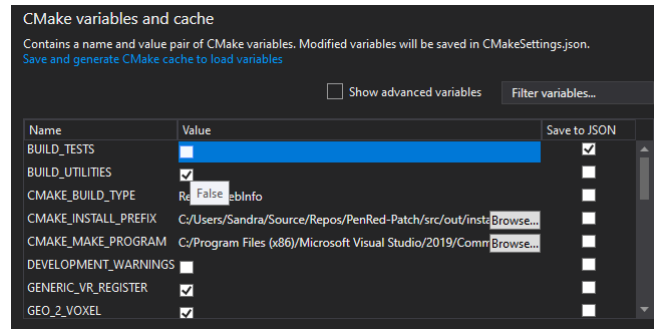


Figure 6: Set compilation flags in MSVS

Finally, build and install PenRed (Figure 7).

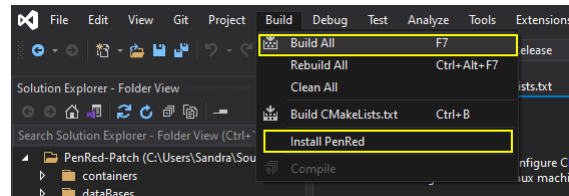


Figure 7: Build and install PenRed in MSVS

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 8).

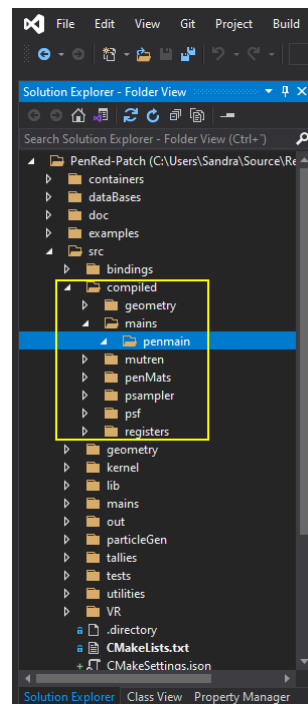


Figure 8: PenRed compiled programs

2.3 Basic usage

To execute the program, the user needs a configuration file and, probably, the required data base files, such as material and geometry files. Their path should be specified at the configuration, remaining the configuration file path as the only program argument. All the details regarding the simulations configuration can be found in the section 5. Moreover, the *examples* folder contain several configuration file examples with the corresponding material and geometry files ready to be executed (section 6). So, to execute the program, the user must use

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

Otherwise, if the MPI capabilities have been enabled at the 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 as specify the hosts where execute the code via the *hostfile* option.

2.4 Optional features

The following list summarise 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:** Switch on or off the compilation with native optimizations. Actually, this flag only works on GCC and Intel C++ compilers.
- **WITH_DICOM:** Handles DICOM capabilities. Requires the *dcmtk* library.
- **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.
- **DEVELOPMENT_WARNINGS:** Enables or disables development warning flags.

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.

```
1 tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"  
2 tallies/cylDoseDistrib/print-xyz true  
3 tallies/cylDoseDistrib/rmin 0.0  
4 tallies/cylDoseDistrib/rmax 30.0  
5 tallies/cylDoseDistrib/nbinsr 60  
6 tallies/cylDoseDistrib/zmin 0  
7 tallies/cylDoseDistrib/zmax 30.0  
8 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 an 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 an 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, let's 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 to the inheritance from the class *A*.

```
1
2 class A{
3     protected:
4         int a1, a2;
5     public:
6         double a3;
7
```

```

8      A() : a1(0),a2(2),a3(5.3){}
9
10     void do1();
11     void do2();
12 };
13
14 class B : public A{
15     private:
16         int b1;
17     public:
18         double b2;
19
20     B() : b1(0),{}
21
22     void doB1();
23     void doB2();
24 };
25

```

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 <i>kpar</i> index (see table 3), only if $ILB[0] > 1$.
ILB[2]	Interaction mechanism <i>ICOL</i> (see tables at section 4.4) that originated the particle, only when $ILB[0] > 1$.
ILB[3]	<p>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,</p> $ILB[3] = Z \times 10^6 + IS1 \times 10^4 + IS2 \times 100 + IS3 \quad (1)$ <p>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 interaction mechanism that caused the initial vacancy in the decaying atom.</p>
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

As we saw at section ??, each particle requires a index identifier provided via the enumeration *pen_KPAR*. Actually PenRed particle indexes and names are summarised at 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 described inside material files. These ones are the same as those used in the original PENELOPE FORTRAN code. Therefore, material files can be generated using the tools and data bases provided at original PENELOPE package, which are also included in the PenRed package. The material data base 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 9 and 10.

5.1.1 Configuration

To specify a material in the configuration file, the user must follow the pattern of code 3, where *materials* is a constant text, *material-name* is a text identifier, or name, selected by the user and, finally, *parameter/path* and *value* are the parameters and values to specify for that material.

```
1 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.

```
1 materials/cu/number 1
2
3 materials/cu/eabs_e- 1.0e3
4 materials/cu/eabs_e+ 1.0e3
5 materials/cu/eabs_gamma 1.0e3
6
7 materials/cu/C1 0.05
8 materials/cu/C2 0.05
9
10 materials/cu/WCC 1.0e3
11 materials/cu/WCR 1.0e3
12
13 materials/cu/filename "Cu.mat"
```

Code 4: Complete material configuration example

As is shown, the parameters to specify for each material are analogous to the required by the PENELOPE FORTRAN version. First, the material *number* is used for the geometry material assignation. This one must be greater than 0, thus material 0 is considered as void. Then, *eabs_e-*, *eabs_e+* and *eabs_gamma* specify the electron, positron and gamma absorption energies respectively. The absorption energy units are *eV*. 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, $C1 \approx 1 - \langle \cos\theta \rangle$, 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.

Finally, the *filename* parameter specify the relative path to the material file, which has been previously created with the material builder program. 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 handle the creation of particles to be simulated. The created particles can be both primary or secondary depending on the selected source. In addition, multiple sources can be specified in the same configuration file. In this case, the sources will be computed sequentially, simulating the first particle of a source after the simulations end of the last particle of the previous source.

To specify a particle source, the user must follow the pattern shown in code 5,

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

Code 5: Source configuration parameters

where *type* must be set to *generic* or *polarized*, whose are used to specify if the source samples generic particle states or gamma polarized states respectively. Via the field *name*, the user specifies a custom name for the particle source. Finally, *parameter/path* specifies the configuration parameter to set with the corresponding *value*. For example, the code 6 shows the basic parameters used by all generic sources type, which name has been set to *source1*.

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

Code 6: Source generic configuration parameters

That source will produce 10^6 primary gamma particles and enables time recording. So the parameter *nhist*, *kpar* and *record-time* sets the number of particles to generate by the source, the particles type (electron, gamma or positron) and if time recording is enabled or disabled respectively.

In addition we can specify if the particles creation must be restricted to a specific body or material using the parameters *source-body* and *source-material* respectively. Both parameters expects an integer index, which specify the source body or material. However, notice that only one restriction can be used for each source. If specified, when a particle is sampled in a different body or material, the spatial sampling will be repeated until the condition is fulfilled.

Note that each parameter requires a specific type. *nhist* requires a number, *kpar* a text (using the double quotes) and *record-time* a boolean (true or false). Now, we have the generic parameters, but the sampler has not knowledge about how to sampling the initial state of the particle, i.e. position, direction, energy, time etc. Therefore, we need to specify the samplers to use, which are classified in spatial, direction, energy, time and specific samplers. Accordingly, to specify what kind of sampler we are configuring, we will use the keywords *spatial*, *direction*, *energy*, *time* and *specific*. Each sampler type is used with a

specific purpose, being spatial, direction, energy and time samplers in charge of determine the particle initial position, direction, energy and live time, respectively. These samplers are cataloged as generic samplers. Nevertheless, specific samplers can set the whole particle state or a portion of it. Therefore, specific samplers can be combined with generic ones depending on the sampler implementation or handle the whole state itself.

Although each sampler has its own parameters, whose can include custom paths, all of them require a parameter named *type*, which sets the sampler to use. For example, to select a mono-energetic energy sampler on the previous source, the line with the required type is shown in code 7. Notice that the source type has been specified after the source name, i.e. with the keyword *energy* and the type value specify which sampler to use among all the available energy samplers.

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

Code 7: Mono-energetic sampler configuration example

In addition, mono-energetic sources only require a single parameter, the particle energy. The next line sets a sampling energy of 30 keV,

```
1 sources/generic/source1/energy/energy 3.0e7
```

Code 8: Mono-energetic sampler configuration example

The other samplers are configured analogously, but using their own parameters. To put it all together, a complete generic source configuration is shown in the code 9, i.e. with no specific sampler. In that code, the lines that begins with *#* are considered comments and ignored by the parser.

```
1 #
2 # Source 1
3 #####
4
5 sources/generic/source1/nhist 1.0e6
6 sources/generic/source1/kpar "gamma"
7 sources/generic/source1/record-time true
8
9 # Directional sampling
10 #####
11
12 sources/generic/source1/direction/type "CONE"
13
14
15 # Set theta
16 sources/generic/source1/direction/theta 0.0
17
18 # Set phi
19 sources/generic/source1/direction/phi 0.0
20
21 # Set oberture (alpha)
22 sources/generic/source1/direction/alpha 5.0
23
24
25 # Energy sampling
26 #####
27
28 sources/generic/source1/energy/type "MONOENERGETIC"
29
30 # Set energy
31 sources/generic/source1/energy/energy 3.0e7
32
33
```

```

34 # Spatial sampling
35 #####
36
37 sources/generic/source1/spatial/type "POINT"
38
39 # Set particle origin
40 sources/generic/source1/spatial/position/x 0.0
41 sources/generic/source1/spatial/position/y 0.0
42 sources/generic/source1/spatial/position/z -25.0

```

Code 9: Complete source configuration example

A source without specific sampler, like the previous one, requires at least spatial, energy and direction samplers, being the time sampler optional. If a specific sampler is used, the required generic samplers depends on the specific sampler itself, and may need some or any generic sampler. In the examples folder we can find configuration files with different sources. In addition, all the available samplers classified by type will be described following.

5.2.1 Spatial source samplers

5.2.1.1 Point

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

```

1 # Spatial sampling
2 #####
3
4 sources/generic/source1/spatial/type "POINT"
5
6 # Set particle origin
7 sources/generic/source1/spatial/position/x 0.0
8 sources/generic/source1/spatial/position/y 0.0
9 sources/generic/source1/spatial/position/z -0.0001

```

Code 10: Spatial source POINT sampler

5.2.1.2 Box

This spatial sampler creates a box where particles are generated with uniform probability. The configuration file contains the information of the box origin introducing the (x, y, z) coordinates values, in cm. In addition, the configuration requires the size of the box in each axis (dx, dy, dz) in cm, which must be positive or zero. All these values are interpreted as doubles. An example of configuration file is shown below.

```

1 # Spatial sampling
2 #####
3
4 sources/generic/source1/spatial/type "BOX"
5
6
7 # Set box size
8 sources/generic/source1/spatial/size/dx 0.75
9 sources/generic/source1/spatial/size/dy 0.75
10 sources/generic/source1/spatial/size/dz 1.25
11
12 # Set particle origin
13 sources/generic/source1/spatial/position/x 0.0
14 sources/generic/source1/spatial/position/y 0.0

```

```
15 sources/generic/source1/spatial/position/z -0.0001
```

Code 11: Spatial source BOX sampler

5.2.2 Direction samplers

5.2.2.1 Cone

Particles generated by the source are sampled in a conical beam. In this sample the overture of the cone must be specified to sample the particles direction uniformly inside the corresponding solid angle. Therefore, the required parameters are the polar (*theta*) and azimuthal (*phi*) angles to determine the solid angle direction, and the semiaperture (*alpha*). Notice that all values are expected in degrees. To reproduce a monodirectional source, the overture must be set to *alpha* = 0.0. However, to obtain a isotropic source, the semiaperture must be set to *alpha* = 180.0. All values are expected to be doubles.

The following lines show the directional sampling of the configuration file of example 2-plane.

```
1
2 # Directional sampling
3 #####
4
5 sources/generic/source1/direction/type "CONE"
6
7
8 # Set theta
9 sources/generic/source1/direction/theta 0.0
10
11 # Set phi
12 sources/generic/source1/direction/phi 0.0
13
14 # Set oberture (alpha)
15 sources/generic/source1/direction/alpha 5.0
```

Code 12: Direction CONE sampler

5.2.2.2 Sphere

In this case the direction of the particles generated are sampled in a sphere section. The information needed in this directional sampler is the direction components in each axis (*u, v, w*), the polar overture introducing the minimum and maximum value of the polar angle (*theta0, theta1*), and the azimuthal overture introducing the minimum value of the phi angle (*phi0*) and its overture (*dphi*). All of this values types must be doubles.

5.2.3 Energy samplers

5.2.3.1 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 is interpreted as double and expected to be positive. 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`.

```
1
2 # Energy sampling
3 #####
4
5 sources/generic/source1/energy/type "MONOENERGETIC"
```

```

6
7 # Set energy
8 sources/generic/source1/energy/energy 1.5e5

```

Code 13: Energy samplers MONOENERGETIC

5.2.3.2 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. A single array named *lowE* will contain all the low energy boundaries of all energy intervals. In the same way, all values for top energy limits must be grouped in an array named *topE*. Finally, the probabilities of each interval are wrote in a third array named *probabilities*. Therefore, the first position of the array *lowE* specify the low boundary for the first interval or spectral line, the first position of the array *topE* the top limit and the first position of the *probabilities* array the corresponding probability. The same is applied to the second position, which specify the second interval or spectral line, and so on. Notice that the probabilities are not required to be normalised, as is shown in the following example code, which corresponds to the 3-detector example, file `detector1.in`. Regarding the variable types, the energy range values and probabilities are doubles and must be, at least, zero.

```

1
2 # Energy sampling
3 #####
4
5 sources/generic/source1/energy/type "INTERVALS"
6
7
8 # Set intervals
9 sources/generic/source1/energy/nintervals 2
10
11
12 # Set energies
13 sources/generic/source1/energy/lowE [1.17e6,1.33e6]
14 sources/generic/source1/energy/topE [1.17e6,1.33e6]
15
16 # Set probabilities
17 sources/generic/source1/energy/probabilities [50.0,50.0]

```

Code 14: Energy samplers INTERVALS

5.2.3.3 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 15, 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.

```

1 #Spectrum      (character '#' is interpreted as a comment)
2 1.20e3 10      1st channel: [1.20,1.61]keV
3 1.61e3 0.0     2nd channel: [1.61,5.00]keV, no emissions
4 5.00e3 20      monoenergetic line at 5.00 keV
5 5.00e3 0       3rd channel: [5.00,6.24]keV, no emissions
6 6.24e3 33.3    4th channel: [6.24,7.32]keV
7 7.32e3 5.02    5th channel: [7.32,8.00]keV
8 8.00e3 10.2    6th channel: [8.00,9.76]keV
9 9.76e3 15.2    7th channel: [9.76,10.0]keV
10 10.0e3 0.0     8th channel: [10.0,15.0]keV, no emissions
11 15.0e3 20      monoenergetic line at 15.0 keV
12 15.0e3 -1      end of the spectrum

```

Code 15: File energy spectrum format

This sampler only requires two configuration values. As the 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 16.

```

1
2 # Energy sampling
3 #####
4
5 sources/generic/source1/energy/type "FILE_SPECTRUM"
6 sources/generic/source1/energy/filename "spectrum.spc"

```

Code 16: File spectrum configuration

5.2.4 Time samplers

5.2.4.1 Decay

This sampler generates, randomly, an exponential timing decay of events from a radioactive element. The time interval is introduced by the user with $[time0, time1]$ and negative values are not allowed. In addition, the *halfLife* of the radioactive element is required.

5.2.5 Specific Sources

5.2.5.1 Phase Space File Source

This type of source uses the phase space file created with the tally type “PSF” to run the simulation, from where the particles will be read. An example where this type of source is used is the example 5-accelerator-2, which configuration is shown in the code 17.

Regarding the required parameters, the *filename* is the relative path to of the file from where the particles will be read, and the parameter *E_{max}* specify the maximum energy value of all particles in the PSF. Then, the two next parameters are related with VR techniques. The first one, *wght-window*, sets the interval to apply splitting and Russian Roulette. How both VR techniques are applied is explained following. First, Russian roulette is applied to particles with a weight value lower than the minimum value of the interval. Instead, splitting is applied to particles with a weight value larger than the maximum value of this interval. Therefore, particles with a weight value inside the interval remain unaltered. The second parameter, *split*, sets the number of splits per particle.

In addition to the VR parameters, the *npartitions* parameter is used to set the number of threads used to execute the simulation. The sampler will use this parameter to split the whole PSF in equal regions to be used by different threads. Finally there are two groups of

optional parameters to be able to rotate and translate the particles in the PSF file. First, the rotation is specified as a ‘ZYZ’ rotation using the Euler angles *omega*, *theta* and *phi*. Then, the user can apply a displacement using the *dx* parameter for translations in the *x* axis, and *dy* and *dz* for *y*, *z* axes respectively. Notice that the rotation is applied before the translation.

```

1 # Source 1
2 #####
3
4 sources/generic/source-psf/nhist 1.0e6
5 sources/generic/source-psf/specific/type "PSF"
6 sources/generic/source-psf/specific/filename "psf-merged.dat"
7 sources/generic/source-psf/specific/Emax 7e6
8
9 sources/generic/source-psf/specific/wght-window [5e-4, 1e-3]
10 sources/generic/source-psf/specific/nsplit 10
11
12 sources/generic/source-psf/specific/npartitions 6
13
14 sources/generic/source-psf/specific/rotation/omega 0
15 sources/generic/source-psf/specific/rotation/theta 90
16 sources/generic/source-psf/specific/rotation/phi 0
17
18 sources/generic/source-psf/specific/translation/dx 0
19 sources/generic/source-psf/specific/translation/dy 0
20 sources/generic/source-psf/specific/translation/dz 10

```

Code 17: Phase Space File Source

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 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 18 shows the geometry configuration for the first simulation example (1-disc). The geometries for this type are defined in an 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.

```

1 geometry/type "PEN_QUADRIC"
2 geometry/input-file "disc.geo"
3 geometry/processed-geo-file "report.geo"
4
5 geometry/dsmax/1 1.0e-4
6 geometry/kdet/1 1

```

Code 18: Complete geometry configuration example

As other configuration examples, the type is specified by the *type* parameter. Then, the *input-file* parameter specifies 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*, specifies a file to generate a geometry report. Secondly, *dsmax* specifies 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 *dsm* is applied is specified via its alias in the parameter path, `geomtry/dsm/*alias*`. For instance, in the previous code, the *dsm* 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 19 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.

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

Code 19: Configuration of geometry absorption energy example

5.3.2 Voxel

Voxelized geometries consists of a 3D matrix of regular prisms 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 nominal density (specified in the material file). 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 original material density in a voxel, this factor must be set to 1.

This kind of geometries requires to set *VOXEL* as geometry type and provide the path to the file where the geometry has been defined (*filename*). The files which stores the voxelized geometry, consists of a binary data dumped by the *pen_voxelGeo* class. One example of how to create a geometry file using that class can be found at the test code,

`src/tests/geometry/voxels/read_Dump.cpp`

which creates a random filled voxel geometry, stores it to a file and, finally, loads the file to be compared with the original created geometry. In addition, another example can be found at the utility *geo2voxel*, which source file is located at,

`src/utilities/geometry/geo2voxel.cpp`

This utility, instantiate a geometry with type and configuration specified by the user, and creates a voxelized geometry file according to that geometry. To do that, the *locate* method is used to map each voxel material and density. As can be found in the mentioned codes, the function to fill a voxel geometry class is named `setVoxels`, and is shown in the code 20, where the initialization of voxels materials and densities is not shown. The required parameters are, a three elements array (*nvox*) with the number of voxels in each axis (*nx, ny, nz*), the voxels sizes in each axis (*sizes*) (*dx, dy, dz*) in cm, a one dimensional array with the material assignation of each voxel (*voxMats*), i.e., with $nx \times ny \times nz$ elements, a one dimensional array with the density factor of each voxel, also with $nx \times ny \times nz$ elements, and a verbose level. Then, once the voxel mesh has been created, the geometry file can be printed in binary format ready to simulate using the function `dump2File`. Also, the voxel geometry information can be printed in ASCII format for visualization purposes using the function `printImage`.

```

1
2 unsigned nvox[3] = ...
3 double sizes[3] = ...
4 unsigned* voxMats = ...
5 double* voxDensFact = ...
6
7 pen_voxelGeo voxelgeo;
8 int err = voxelgeo.setVoxels(nvox, sizes, voxMats, voxDensFact, 3);
9 if(err != 0){
10     printf("Error using 'setVoxels': %d\n", err);
11     return -1;
12 }
13 //Print ASCII file
14 voxelgeo.printImage("voxelGeo.ascii");
15 //Dump binary file
16 voxelgeo.dump2File("voxelGeo.bin");

```

Code 20: Set voxels function and geometry file creation

Regarding the geometry configuration step, the user must specify the parameters (nx, ny, nz) and (dx, dy, dz) to describe the number of voxels and its dimensions, respectively, in each axis. Also, $dsmax$ can be specified as we done for quadric geometries. This one is done following the template,

`../dsmax/nvalue`

where n selects the material where $dsmax$ will be applied and $value$ the $dsmax$ value. An example of configuration file for voxelized geometries is shown in the code 21.

```

1 geometry/type "VOXEL"
2 geometry/filename "3blocks.vx"
3 geometry/nvoxels/nx 60
4 geometry/nvoxels/ny 60
5 geometry/nvoxels/nz 80
6 geometry/voxel-size/dx 0.2
7 geometry/voxel-size/dy 0.2
8 geometry/voxel-size/dz 0.1
9
10 geometry/dsmax/1 0.05
11 geometry/dsmax/2 0.02
12 geometry/dsmax/3 0.02
13 geometry/dsmax/4 0.50

```

Code 21: Configuration of voxelized geometry example

5.3.3 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 22, where we define an interval named *Air*.

```

1 geometry/intensity-ranges/Air/material 1
2 geometry/intensity-ranges/Air/low -2000
3 geometry/intensity-ranges/Air/top -500
4 geometry/intensity-ranges/Air/density 0.001290

```

Code 22: DICOM intensity ranges configuration

- **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:** Material to assign to this contour.
 - **density:** Density to assign to this contour (g/cm^3).
 - **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.

For example, to configure a contour named *target*, the configuration file should contain something like the lines shown in code 23.

```

1 geometry/contours/target/material 1
2 geometry/contours/target/density 1.05
3 geometry/contours/target/priority 1.0

```

Code 23: DICOM contour 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*.
- **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 *dicom.ASCII.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 9.

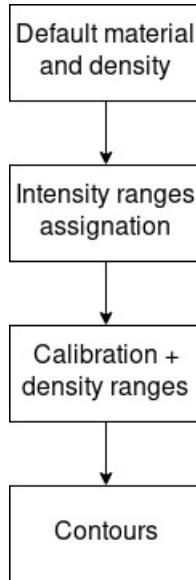


Figure 9: DICOM voxels material and density assign methods hierarchy.

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 (PET) 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 [9]. So, the user must download that DICOM to reproduce the example.

5.4 Tallies

Configurations for tallies are similar to sources. The pattern to configure tallies is exemplified in the code 24, 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. As sources and geometries, exists several tally types, so the user must specify the parameter *type* on each created tally.

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

Code 24: Tally configuration pattern

Code 25 shows an example to configure a cylindrical dose distribution tally, which 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*). The optional parameter *print-xyz* serves to enable (*true*) or disable (*false*) more information about radial coordinates at the output file: the low and the average value of these coordinates.

```
1 tallies /cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
2 tallies /cylDoseDistrib/print-xyz true
3 tallies /cylDoseDistrib/rmin 0.0
4 tallies /cylDoseDistrib/rmax 30.0
5 tallies /cylDoseDistrib/nbinsr 60
6 tallies /cylDoseDistrib/zmin 0
7 tallies /cylDoseDistrib/zmax 30.0
8 tallies /cylDoseDistrib/nbinsz 60
```

Code 25: Tally configuration pattern

An aspect to consider when we are choosing the limits, is that the behaviour of the intervals is [*min*, *max*). Thus *rmin* and *zmin* 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 doubles, 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 26, which has been extracted from the file *disc.in* of the example 1-disc.

```

1 tallies/cylDoseDistrib/type "CYLINDRICAL_DOSE_DISTRIB"
2 tallies/cylDoseDistrib/print-xyz true (optional)
3 tallies/cylDoseDistrib/rmin 0.0 (mandatory)
4 tallies/cylDoseDistrib/rmax 0.01 (mandatory)
5 tallies/cylDoseDistrib/nbinsr 50 (mandatory)
6 tallies/cylDoseDistrib/zmin 0.0 (mandatory)
7 tallies/cylDoseDistrib/zmax 0.005 (mandatory)
8 tallies/cylDoseDistrib/nbinsz 100 (mandatory)
9 tallies/cylDoseDistrib/nbinsPhi 1 (optional)

```

Code 26: Cylindrical dose distribution tally configuration

5.4.2 Emerging Particles Distribution

This tally measures the energy distribution of particles that left the geometry within a specific energy range of $[emin, emax)$, expressed in particles/(eV history). We consider two cases in this tally. First, down bound emerging particles, which W direction is lower or equal to zero. Then, the up bound case when the W value is greater than zero. Also, the tally measures the number of particles that left the geometry per steradian determined by the theta and phi angular values, in degrees, of the particle deviation. In this case, measurements are in particles/(sr history).

The tally information needed in the configuration file is shown below,

- **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 doubles, 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 27 is an example of this tally, which has been extracted from the configuration file *detector1.in* of the example 3-detector-1:

```

1 tallies/EmergingPartDistrib/type "EMERGING_PART_DISTRIB"
2 tallies/EmergingPartDistrib/emin 0.0 (mandatory)
3 tallies/EmergingPartDistrib/emax 1.45e6 (mandatory)
4 tallies/EmergingPartDistrib/nBinsE 280 (mandatory)
5 tallies/EmergingPartDistrib/nBinsTheta 45 (mandatory)
6 tallies/EmergingPartDistrib/nBinsPhi 18 (mandatory)

```

Code 27: Emerging particles tally configuration

5.4.3 Energy Deposition Body

This tally measures the energy, in eV per history, deposited in each body.

The configuration variables of this tally are described following,

- **type “EDEP_BODY”**: Type name of the tally.
- **nBody**: Number of bodies to calculate the deposited energy. All bodies with a index below *nBody* will be tallied.

The *nBody* variable must be an integer. The output filename ends with *bodyEnergyDeposition.dat*.

A configuration example of this tally is show in the code 28, which belongs to the file *plane.in* of the example 2-plane.

```

1 tallies/bodyEDep/type "EDEP_BODY"
2 tallies/bodyEDep/nBody 2 (mandatory)

```

Code 28: 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.

The parameters required to configure this tally are listed below,

- **type “EDEP_MAT”**: Type name of the tally.
- **nmat**: Number of materials to tally the energy deposition. All materials whith a index lesser than *nmat* will be tallied.

This *nmat* value must be an integer. The corresponding output file ends with *materialEnergyDeposition.dat*

The code 29 shows a configuration example obtained from the configuration file *detector1.in* belonging to the example 3-detector-1.

```

1 tallies/matEDep/type "EDEP_MAT"
2 tallies/matEDep/nmat 2 (mandatory)

```

Code 29: Energy deposition in material tally configuration

5.4.5 Impact Detector

This tally is cabaple to measure different magnitudes: fluence spectrum, particle energy spectrum and particle age. Each one is described following,

- **Fluence**: integrates the spectral fluence over the detector volume in cm/eV. The output file contains the fluence for each particle type and the total fluence in the specified detector. The filename ends with *fluenceTackLength-num.dat*, where *num* is the corresponding detector number.

- **Energy spectrum:** reports the energy spectrum of particles which enter in the specified detector volume. Particles created inside the detector, for example secondary particles, are not considered in this tally. Units are expressed in 1/(eV history). The output file contains the energy spectrum for each particle type and the total spectrum in the specified detector is also measured. The filename ends with *spectrum-impdet-num.dat*, where *num* is the number of the assigned detector.
- **Age:** reports the age distribution of particles which impact to the considered detector, in units of 1/(seconds history). The output file that ends with *age-impdet-num.dat*, contains the probability distribution along each time interval over all simulated particles. As in the previous cases, *num* is the number that identifies the the detector.
- **Energy deposition:** reports the energy deposited spectrum measured in the considered detector. Units are expressed in 1/(eV history). The output file ends with *energyDeposition-impdet-num.dat*, where *num* is the number of the detector. This file contains the probability distribution along each energy interval for the specific detector over 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 and energy deposition measurements.
- **emax:** Maximum energy value for tallied particles in fluence, energy spectrum and energy deposition 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.
- **energy-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 doubles and the bin numbers must be specified as integers.

To exemplify the tally usage, next are two examples to show different configurations for this tally. First, code 30 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.

```

1 tallies/ImpactDetector/type "IMPACT_DET"
2 tallies/ImpactDetector/detector 1 (mandatory)
3 tallies/ImpactDetector/fluence true (optional)
4 tallies/ImpactDetector/emin 1.0e5 (mandatory/optional)
5 tallies/ImpactDetector/emax 3.5e7 (mandatory/optional)
6 tallies/ImpactDetector/nbin-energy 100 (mandatory/optional)
7 tallies/ImpactDetector/linearScale-fln true (optional)
8 tallies/ImpactDetector/spectrum true (optional)
9 tallies/ImpactDetector/age true (optional)
10 tallies/ImpactDetector/linearScale-age false (mandatory/optional)
11 tallies/ImpactDetector/nbin-age 100 (mandatory/optional)
12 tallies/ImpactDetector/age-min 1.0e-9 (mandatory/optional)
13 tallies/ImpactDetector/age-max 1.0e-8 (mandatory/optional)

```

Code 30: Impact detector tally configuration example 1

Secondly, code 31 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.

```

1 tallies/ImpactDetector/type "IMPACT_DET"
2 tallies/ImpactDetector/detector 1 (mandatory)
3 tallies/ImpactDetector/emin 0.0e0 (mandatory/optional)
4 tallies/ImpactDetector/emax 1.45e6 (mandatory/optional)
5 tallies/ImpactDetector/nbin-energy 280 (mandatory/optional)
6 tallies/ImpactDetector/energy-dep true (optional)
7 tallies/ImpactDetector/linearScale-edep true (optional)

```

Code 31: 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/cm²).

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 z . Must be greater than $ymin$.
- **nz**: Number of z bins. Must be at least 1.

Considering the parameters type, those who specify coordinate values must be doubles, while the number of bins for each coordinate must be specified as integer. 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 32 shows an example extracted from the configuration file *disc.in* that belongs to the example 1-disc-novr.

```

1 tallies/SpatialDoseDistrib/type "SPATIAL_DOSE_DISTRIB"
2 tallies/SpatialDoseDistrib/print-xyz true (optional)
3 tallies/SpatialDoseDistrib/xmin 0.0 (mandatory)
4 tallies/SpatialDoseDistrib/xmax 1.0 (mandatory)
5 tallies/SpatialDoseDistrib/nx 1 (mandatory)
6 tallies/SpatialDoseDistrib/ymin 0.0 (mandatory)
7 tallies/SpatialDoseDistrib/ymax 1.0 (mandatory)
8 tallies/SpatialDoseDistrib/ny 1 (mandatory)
9 tallies/SpatialDoseDistrib/zmin 0.0 (mandatory)
10 tallies/SpatialDoseDistrib/zmax 0.005 (mandatory)
11 tallies/SpatialDoseDistrib/nz 100 (mandatory)

```

Code 32: Spatial Dose Distribution tally configuration

5.4.7 Angular Detector

This tally reports the angular energy spectrum in a specified detector. This information is tallied in the energy interval $[emin, emax)$, specified in eV, and the angular intervals $[theta1, theta2)$, $[phi1, phi2)$, specified in degrees. The energy spectra of particles is tallied in units of $1/(eV \text{ sr particle})$.

Below, we describe 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 the number of bins, and doubles 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 33 belongs to the configuration of the example **1-disc-novr**, where this tally is used. The configuration file is named *disc.in*.

```

1 tallies/AngularDetector/type "ANGULAR_DET"
2 tallies/AngularDetector/detector 1 (mandatory)
3 tallies/AngularDetector/emin 0.0 (mandatory)
4 tallies/AngularDetector/emax 40.5e3 (mandatory)
5 tallies/AngularDetector/theta1 90.0 (mandatory)
6 tallies/AngularDetector/theta2 180.0 (mandatory)
7 tallies/AngularDetector/phi1 0.0 (mandatory)
8 tallies/AngularDetector/phi2 360.0 (mandatory)
9 tallies/AngularDetector/nBinsE 200 (mandatory)
10 tallies/AngularDetector/linearScale true (optional)

```

Code 33: 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 34. This line is the same for all examples where this tally will be used.

```

1 tallies/secondary/type "SECONDARY_GEN"

```

Code 34: 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 . If this parameter is not specified, will be set to **false** by default.
- **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 doubles 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 35.

```

1 tallies/SphericalDose/type "SPHERICAL_DOSE_DISTRIB"
2 tallies/SphericalDose/print-xyz true (optional)
3 tallies/SphericalDose/rmin 0.0 (mandatory)
4 tallies/SphericalDose/rmax 0.06 (mandatory)
5 tallies/SphericalDose/nr 65 (mandatory)
6 tallies/SphDoseDistrib/ntheta 1 (optional)
7 tallies/SphDoseDistrib/nphi 1 (optional)

```

Code 35: Spherical dose distribution tally configuration

5.4.10 Phase Space File (PSF)

This tally creates a particle phase space file which store all particles that impact at the specified detector. Also, the stored particles can be limited by a energy range.

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

- **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*.

```

1 tallies/psf/type "PSF" (mandatory)
2 tallies/psf/detector 1 (mandatory)
3 tallies/psf/emin 0.0 (mandatory)
4 tallies/psf/emax 6.1e6 (mandatory)

```

Code 36: Phase space file tally configuration

This tally can be used with multiple threads. In that case, each thread will store, temporally, their psf particles in an independent file. Then, when the simulation finishes, all files will be concatenated automatically ordered by the thread ID 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 [10]. 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 file with the μ_{en}/ρ (cm^2/g) coefficients per tallied material is required. 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 file provided will be ignored by the tally. To obtain the μ_{en}/ρ coefficients, the utility **mutren** can be used, which is provided in the PenRed package. A description can be found in the section 7.1.

- **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 37 we show a configuration example for that tally.

```

1 tallies/kermaTrackLength/type "KERMA_TRACKLENGTH" (mandatory)
2 tallies/kermaTrackLength/emin 1.0e3 (mandatory)
3 tallies/kermaTrackLength/emax 1.5e6 (mandatory)
4 tallies/kermaTrackLength/dataFiles/1 "mu-Water1000.mat" (mandatory)
5
6 #Optional section:
7 tallies/kermaTrackLength/cylindrical/nr 10 (mandatory)
8 tallies/kermaTrackLength/cylindrical/rmin 0.1 (optional)
9 tallies/kermaTrackLength/cylindrical/rmax 10.0 (mandatory)
10 tallies/kermaTrackLength/cylindrical/nphi 10 (mandatory)
11 tallies/kermaTrackLength/cylindrical/nz 10 (mandatory)
12 tallies/kermaTrackLength/cylindrical/zmin -10.0 (mandatory)
13 tallies/kermaTrackLength/cylindrical/zmax 10.0 (mandatory)
14
15 #Optional section:
16 tallies/kermaTrackLength/cartesian/nx 10 (mandatory)
17 tallies/kermaTrackLength/cartesian/xmin -10.0 (mandatory)
18 tallies/kermaTrackLength/cartesian/xmax 10.0 (mandatory)
19 tallies/kermaTrackLength/cartesian/ny 10 (mandatory)
20 tallies/kermaTrackLength/cartesian/ymin -10.0 (mandatory)
21 tallies/kermaTrackLength/cartesian/ymax 10.0 (mandatory)
22 tallies/kermaTrackLength/cartesian/nz 10 (mandatory)
23 tallies/kermaTrackLength/cartesian/zmin -10.0 (mandatory)
24 tallies/kermaTrackLength/cartesian/zmax 10.0 (mandatory)
25
26 #Optional section:
27 tallies/kermaTrackLength/spherical/nr 10 (mandatory)
28 tallies/kermaTrackLength/spherical/min 0.2 (optional)
29 tallies/kermaTrackLength/spherical/rmax 10.0 (mandatory)
30 tallies/kermaTrackLength/spherical/ntheta 10 (mandatory)
31 tallies/kermaTrackLength/spherical/nphi 10 (mandatory)

```

Code 37: 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 exist, 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 doubles.

5.5 Variance Reduction

PenRed implements the very same variance reduction (VR) techniques as the original FORTRAN code and some more. These are, for generic simulations, interaction forcing (*IF*), x-ray and bremsstrahlung splitting. Then, phase space file based simulations adds particle

splitting and Russian Roulette techniques. Note that variance reduction for phase space file will be configured via the corresponding particle source parameters, as we saw on section 5.2. In addition, PenRed implements generic splitting and Russian roulette as VR modules and provide the capabilities to add new ones, as is explained in the section 5.5.

We have two types of VR techniques. First, the context specific techniques, which are implemented inside the particle class, such as interaction forcing or bremsstrahlung splitting. On the other hand, we have the VR techniques implemented as independent modules, such as x-ray and generic splitting or Russian roulette.

5.5.1 Context specific

To use the first kind of techniques, the parameters in the configuration file use the prefix *VR*, as shows code 38. There, *vr-technique* can be interaction forcing (*IForcing*) or bremsstrahlung splitting (*bremss*). Then, on *objectToApply*, the user can select *bodies* or *materials* to apply the variance reduction technique to a single body or to an entire material respectively. Next, *name* is a user defined text identifier for this VR technique. Finally, *parameter/path* and *value* depends on the VR type.

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

Code 38: Variance reduction configuration pattern

First, lets see the interaction forcing (IF) configuration. The code 39 shows a complete interaction forcing configuration used in the example *4-x-ray*. As we can see, this example enables interaction forcing in a single body, which alias is specified by the parameter *body* as a string. Then, specify the kind of particle to force using the parameter *particle* and the interaction numerical identifier to force (*interaction*). That identifier can be found on section 4.4. The next parameter, *factor*, sets the interaction forcing amplification factor and, finally, *min-weight* and *max-weight* limits the weight window where apply this VR technique. Notice that a negative *factor* value will be interpreted as in the PENELOPE main program, i.e., is assumed to mean that a particle with energy $E = E_{PMAX}$ should interact, on average, $+factor$ times in the course of its slowing down to rest, for electrons and positrons, or along a mean free path, for photons.

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

Code 39: Interaction forcing configuration for bodies

To set the same interaction forcing on materials, simply substitute *bodies* by *materials*, *body* parameter by *mat-index* and set the material index where apply VR as a integer value. For example, to apply this IF on material 2 the configuration should look like code 40.

```
1 VR/IForcing/materials/VR1/mat-index 2
2 VR/IForcing/materials/VR1/particle "electron"
3 VR/IForcing/materials/VR1/interaction 2
4 VR/IForcing/materials/VR1/factor 400
5 VR/IForcing/materials/VR1/min-weight 0.1
6 VR/IForcing/materials/VR1/max-weight 2
```

Code 40: Interaction forcing configuration for materials

Next technique, bremsstrahlung splitting, requires one or two parameters depending on whether the VR is specified for bodies or materials respectively. Both patterns are shown

in the code 41, where *body-alias* must be substituted by the body alias where we want to apply the splitting, *splitting-factor* specify the number of times bremsstrahlung photons will be cloned and *imat* is the material index where apply splitting.

```
1 VR/bremss/bodies/body-alias/splitting splitting-factor
2
3 VR/bremss/materials/mat-index imat
4 VR/bremss/materials/splitting splitting-factor
```

Code 41: Bremsstrahlung splitting configuration for bodies and materials

5.5.2 Specific VR techniques

A specific VR techniques can be used only on a specific particle state type. All of them, when used in the PenRed provided main program, follows the configuration pattern code 42.

```
1 VR/type/vr-name/parameter/path value
```

Code 42: Specific VR technique parameter configuration pattern

Actually, PenRed only provides specific VR techniques for photons. Thus, the only possible value for the specific VR type is *photon*.

```
1 VR/photon/vr-name/parameter/path value
```

Code 43: Photons VR technique parameter configuration pattern

The parameter *vr-name* specify a name for this VR instance and is only used for identification purposes.

5.5.2.1 X-Ray splitting

X-Ray splitting uses a pattern similar to bremsstrahlung case, an example can be found in the example *1-disc-vr* and all the available parameters are summarised in the code 44.

```
1 VR/photon/vr-name/type "XRAY_SPLITTING" (mandatory)
2
3 VR/photon/vr-name/bodies/body-alias/splitting splitting-factor (optional)
4
5 VR/photon/vr-name/materials/mat-index imat (optional)
6 VR/photon/vr-name/materials/splitting splitting-factor (optional)
```

Code 44: X-ray splitting configuration for bodies and materials

First, the VR type is specified by the parameter *type*. Then, the other parameters *VR/photon/name/bodies/...* and *VR/photon/name/materials/...* are analogous to the bremsstrahlung case.

5.5.3 Generic VR techniques

Generic VR techniques can be used on all particle state types. All of them, when used in the PenRed provided main program, follows the configuration pattern shown in the code 45.

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

Code 45: Specific VR technique parameter configuration pattern

where *vr-name* specify the VR instance name for identification purposes.

5.5.3.1 Splitting

The splitting VR technique consists of cloning the particle and reducing its weight by a factor equal to the number of clones. Thus, if a particle with a weight $WGHT$ is split in 10 clones, counting itself, the weight of all the resulting particles will be $WGHT/10$. An example of configuration can be found in the example *8-fake-chamber*. Also, all the parameters are shown in the code 46 and explained below,

- **type**: Specify the VR type, must be set to *"SPLITTING"*.
- **minWght**: Specify the minimum particle weight to apply splitting. Particles with lesser weight than the specified will not be split.
- **maxWght**: Specify the maximum particle weight to apply splitting. Particles with greater weight than the specified will not be split.
- **materials/mat-name**: Subsection to specify splitting parameters for materials. Notice that **mat-name** is only used as identifier in the VR instance, and will not be identified with the name used in the geometry configuration. Instead, the material is specified by the integer index **mat-index**. Finally, the number of cloned particles in this material is specified by the **splitting** parameter. This parameter can be repeated for different material indexes.
- **bodies/body-alias/splitting**: There, *body-alias* must identify a valid body in the configured geometry. The specified value of **splitting** will be used in this body. Notice that *bodies* configuration overwrite materials ones. This parameter can be repeated for different bodies.

```
1 VR/generic/vr-name/type "SPLITTING" (mandatory)
2 VR/generic/vr-name/minWght 0.05 (mandatory)
3 VR/generic/vr-name/maxWght 21.0 (mandatory)
4
5 VR/generic/vr-name/materials/mat-name/mat-index 2 (optional)
6 VR/generic/vr-name/materials/mat-name/splitting 20 (optional)
7
8 VR/generic/name/bodies/body-alias/splitting 20 (optional)
```

Code 46: Splitting configuration for bodies and materials

Notice that the splitting is only applied when a particle enter to the material/body after crossing an interface. Therefore, is not applied to secondary particles generated inside the material and is not triggered in bodies if the geometry has not an interface in their boundaries. To force a interface, the user can create a detector or use different materials.

5.5.3.2 Russian roulette

Russian roulette technique kills the particles with a probability specified in the configuration (**prob**). If the particle survive, its weight is multiplied by a factor $1/prob$. All the parameters of the Russian roulette technique are equivalent to the *splitting* class parameters (section 5.5.3.1). The only difference is that the **splitting** parameter is substituted by the **prob** parameter, which specify the particle survival probability. The parameters are summarised in the code 47. In addition, the **type** parameter must be set to *"RUSSIAN_ROULETTE"* instead of *"SPLITTING"*. An example of configuration can be found in the example *8-fake-chamber*.

```

1 VR/generic/vr-name/type "RUSSIAN.ROULETTE" (mandatory)
2 VR/generic/vr-name/minWght 0.05 (mandatory)
3 VR/generic/vr-name/maxWght 21.0 (mandatory)
4
5 VR/generic/vr-name/materials/mat-name/mat-index 2 (optional)
6 VR/generic/vr-name/materials/mat-name/prob 20 (optional)
7
8 VR/generic/name/bodies/body-alias/prob 20 (optional)

```

Code 47: Russian roulette configuration for bodies and materials

Like in the *splitting* case, Russian roulette only triggers when a particle cross an interface.

5.6 Simulation parameters

Simulation parameters are main specific, unlike source, tally, geometry or material configurations. This section will explain the available parameters for the “pen.main” program. All parameters follows the pattern code 48.

```

1 simulation/parameter/path value

```

Code 48: Simulation parameters configuration pattern

Our main provides the capability to dump the current state of the whole simulation. This dump can be used to resume a crashed simulation. To configure this feature, the user can use a set of configuration parameters. The first one, *dump-interval* specify the time, in seconds, between successive dumps. Next, *dump2read* expects a string with the name of a dump file to read. This file will be read before the simulation beginning to continue a previous simulation. Another parameter is *dump2write* which expect a string and allow the user to change the default dump filename (dump.dat). Finally, *dump2ascii* tells to the program that the simulation should not be resumed. Instead, the program will load the state stored in the specified dump file (specified with *dump2read*) and extract the tally contents using the usual data reports. Notice that to use this option is necessary to specify the *dump2read* parameter. An example of dump configuration can be found in the code 49, 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.

```

1 simulation/dump-interval 3600
2 simulation/dump2read "dump.dat"
3 simulation/dump2write "dump2.dat"

```

Code 49: Dump configuration parameters

When multiple threads are used during the simulation, the program will create a independent dump file for each one. To avoid override the dump files, each thread appends an identifier using its thread ID number, same as used for tally reports. Also, when we use multi-threading to read a dump file, the program expects one dump file for each thread with the name specified by the *dump2read* parameter and the thread ID prefix.

Another type of simulation parameters is the one that allows us to control the multi-threading capabilities. The first parameter to configure multi-threading is *nthreads*, which expect an integer value to specify the number of threads to use within the simulation. The number of histories to simulate on each source will be distributed among all specified threads. By default the number of threads is set to one.

```

1 simulation/threads 2

```

Code 50: 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, this is specified via the *mpirun* parameters (see section 2). So, when both kinds of parallelism are combined, each MPI process will spawn a number of threads equal to the 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.

Actually is not possible to specify a different number of threads for each MPI process, but this feature is intended to be implemented in future PenRed versions.

Following with multi-threading parameters, *thread-affinity* parameter expect a boolean to enable or disable CPU affinity. Actually, this feature can be used only if threads are implemented via *pthread*s package, i.e., in most Unix environments.

```
1 simulation/thread-affinity true
```

Code 51: Threads affinity

Finally, the user can specify the initial random generator seeds using *seed1* and *seed2* parameters. Another option consists of selecting a seed pair provided by *rand0* [11] function via the *seedPair* parameter. Notice that on multi-threading and/or MPI simulations, only the parameter *seedPair* can be used to set initial seeds. This restriction is necessary to ensure that each random number chain is truly independent.

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

Code 52: Initial seeds

5.7 Load balance parameters

As simulation parameters, load balance parameters are main dependent. This optional feature can be enabled during the cmake configuration with the option *WITH_LB*. Actually, the only mandatory parameter to configure the load balance system is *balance-interval* (code 53). 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 cmake 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.

```

1 loadBalance/balance-interval 500      (mandatory)
2 loadBalance/host "server-hostname"    (optional)
3 loadBalance/port 5555                  (mandatory if enabled "host")
4 loadBalance/worker 0                   (mandatory if enabled "host")
5 loadBalance/CA-cert                    (optional)
6 loadBalance/cert-file                  (mandatory if enabled "CA-cert")
7 loadBalance/key-file                   (mandatory if enabled "CA-cert")
8 loadBalance/key-password                (optional)
9 loadBalance/hostname                    (optional)

```

Code 53: Load balance parameters

6 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
```

6.1 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.

6.1.1 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.

6.1.2 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.

6.2 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.

6.3 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.

6.4 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.

6.5 5-accelerator

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

6.5.1 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.

6.5.2 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.

6.5.3 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

6.6 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.

6.7 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.

6.8 8-fake-chamber

This example describes an ionization chamber geometry inside a water phantom of $30 \times 30 \times 30 \text{ cm}^3$. 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.

6.8.1 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.

6.8.2 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.

7 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/
```

7.1 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**).

7.2 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 find `iaeaPSF` folder which includes the 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 command line to execute it is

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

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

7.3 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.

where *granularity* and the number of bins are expected to be integers, and the remaining parameters doubles. 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 54, 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.

```

1 geometry/type "PEN_QUADRIC"
2 geometry/input-file "disc.geo"
3 geometry/processed-geo-file "report.geo"
4
5 voxelized/nx 300
6 voxelized/ny 300
7 voxelized/nz 120
8
9 voxelized/dx 0.1
10 voxelized/dy 0.1
11 voxelized/dz 0.2
12
13 voxelized/ox -15.0
14 voxelized/oy -15.0
15 voxelized/oz -12.0
16
17 voxelized/granularity 4

```

Code 54: Complete `geo2voxel` configuration example to convert a quadric geometry.

7.4 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.

7.5 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.

7.6 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.

7.7 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 8. These ones are executed with no arguments via the command line.

Executable	Registered information
<code>regGenericVR</code>	Generic VR types
<code>regPhotonVR</code>	Specific photon VR types
<code>regGeometries</code>	Geometry types
<code>regSamplers</code>	Particle sampler types
<code>regTallies</code>	Tally types

Table 8: Available register programs with the corresponding registered information.

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 9: 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	Ferroboration
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 10: 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 Agency (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] 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>
- [10] 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>
- [11] 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>