

PenRed: User Manual

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

¹Departament de Física Atòmica Molecular i Nuclear, Universitat de València, Dr. Moliner, 50, 46100, Burjassot, València, Spain, gimenez.alventosa.vicent@gmail.com

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

²Departament de Física Teòrica and IFIC, Universitat de València-CSIC, Dr. Moliner, 50, 46100, Burjassot, València, Spain,

September 3, 2025

Abstract


The present document is a manual for the penRed code system, whose content is focused on the description of the usage of the framework. penRed is a general-purpose, multiplatform, standalone, 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 techniques. In addition, penRed offers, among other features, the capability to read and process DICOM images to perform simulations. These features facilitate its usage in medical applications. To enhance simplicity, a Python interface has been implemented to run penRed simulations and utilities within the Python environment. This package, named pyPenred, can be installed via pip with no compilation requirements. Our framework has been successfully tested against the original PENELOPE Fortran code and other benchmarks.


1 Introduction

The present document contains all the information necessary to use the penRed [1] framework. That is, all modules and programs already included in the package. 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 implementation guide included within the penRed package.

1.1 What is penRed?

PenRed is a fully parallel, modular, and customisable framework for Monte Carlo simulations of the passage of radiation through matter. It is based on the PENELOPE [2] code system, inheriting its physics models and tracking algorithms for simulated particles. For

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>

further details on the interactions models, the reader is referred to the PENELOPE manual [3]. PenRed has been coded in C++ following an object-oriented programming paradigm restricted to the C++14 standard, allowing it to run different platforms and OS. It has been tested in Linux, Windows and MacOS environments. 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 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. Section 3 provides some information of the internal data structure format used to configure the simulations. The required files to run a simulation are summarized in the Section 4. In Section 5 a briefly description of important definitions, constants, parameters and units is shown. Following, Section 6, describes all the implemented tallies, sources, geometries, variance reduction techniques and other parameters to configure the simulation. Then, the Section 7 describes *pyPenred*, the python binding developed to use penRed through python programming. Finally, the Section 8 describe the examples located in the **examples** folder, which are ready to be simulated. Finally, the offered utilities in the penRed package and how to use them are explained in the section 9.

Notice that the section 5 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	6
2.1	Pre-compiled (pyPenred)	6
2.2	Manual compilation	6
2.2.1	Automatic scripts	7
2.2.2	CMake configuration	7
2.2.3	Compilation options	10
2.2.4	Basic usage	11
2.3	Examples	11
3	Internal data format	11
4	Simulation files	12
5	Constants, parameters and definitions	12
5.1	Important definitions	12
5.2	Units	14
5.3	Particle indexes	15
5.4	Particle interaction indexes	15
5.5	Atomic electron shells	15
6	Simulation configuration	16
6.1	Materials	16
6.1.1	Common configuration	17
6.1.2	Composition	19
6.1.3	Composition databases	20
6.2	Particle sources	21
6.2.1	Generic Sampling Sources	22
6.2.2	Spatial source samplers	23
6.2.3	Direction samplers	29
6.2.4	Energy samplers	30
6.2.5	Time samplers	32
6.2.6	Specific samplers	33
6.3	Geometries	39
6.3.1	Geometry visualization	40
6.3.2	Quadric	40
6.3.3	Triangular Mesh Surfaces	41
6.3.4	Voxel	45
6.3.5	DICOM	48
6.3.6	Combo	52
6.4	Tallies	54
6.4.1	Output files	54
6.4.2	Intervals	55
6.4.3	Output data	55
6.4.4	Cylindrical Dose Distribution	55

6.4.5	Emerging Particles Distribution	57
6.4.6	Energy Deposition Body	58
6.4.7	Energy Deposition Material	58
6.4.8	Impact Detector	59
6.4.9	Spatial Dose Distribution	61
6.4.10	Angular Detector	62
6.4.11	Particle Generation	63
6.4.12	Spherical Dose Distribution	63
6.4.13	Phase Space File (PSF)	64
6.4.14	Kerma track length estimator	65
6.4.15	CT tally	67
6.4.16	DICOM kerma tally (Dose Volume Histogram)	69
6.4.17	Tracks	71
6.4.18	Primary and Scatter Separated (PSS)	72
6.4.19	Detection Spatial Distribution	74
6.4.20	Emerging Spherical Distribution	75
6.5	Variance Reduction	77
6.5.1	Context specific	77
6.5.2	Generic VR techniques	79
6.5.3	Specific VR techniques	81
6.6	Simulation parameters	82
6.6.1	Dump	82
6.6.2	Threading	84
6.6.3	Seeds	84
6.6.4	Simulation time	85
6.6.5	Logs	85
6.6.6	Image output	85
6.7	Load balance parameters	86
6.7.1	Balance server	86
7	pyPenred	87
7.1	Installation	87
7.2	Configuration Format	87
7.3	Usage	89
7.4	API	92
8	Examples	93
8.1	1-disc	93
8.1.1	1-disc-no-vr	93
8.1.2	1-disc-vr	93
8.2	2-plane	93
8.3	3-detector	94
8.4	4-x-ray-tube	94
8.5	5-accelerator	94
8.5.1	5-accelerator-1	94
8.5.2	5-accelerator-2	94
8.5.3	5-accelerator-3	95
8.6	6-polarisation	95
8.7	7-aba	95
8.8	8-fake-chamber	95
8.8.1	8-fake-chamber-novr	96

8.8.2	8-fake-chamber-vr	96
9	Utilities	96
9.1	Blender Plugin	96
9.2	Mutren	97
9.3	iaeaPSF	97
9.4	geo2voxel	97
9.5	geo2mesh	99
9.6	range	100
9.7	PSF spectrum	101
9.8	PSF to ASCII	101
9.9	Registers	101
9.10	CompositionsDB	101
9.11	Material creation	102
9.12	internalData2YAML	103
A	Predefined materials	104

2 Installation

There are two main options to install and use penRed. The simplest is to use pre-compiled packages, which require no compilation or manual dependency installation. The second option is to compile the code locally. While this approach demands extra effort, penRed provides automatic scripts and minimize the necessary programs to simplify the compilation process. Additionally, local compilation allows the compiler to leverage hardware-specific optimizations, improving computational efficiency.

This section will explain how to install penRed. The description of the necessary configuration files to run a simulation can be found in the Section 4

2.1 Pre-compiled (pyPenred)

Regarding the pre-compiled options, pyPenred can be installed as any other pip module:

```
pip install pyPenred
```

or, if pip is not installed as a stand-alone package in the system, you can use it through python:

```
python -m pip install pyPenred
```

The pyPenred module is available for Linux, Windows and MacOS systems. Once the installation is finished, a configured simulation can be executed providing the corresponding configuration file, as is shown in the Code 1. The details about the necessary configuration files can be found in Section 4.

```
1 # Import the pyPenred package
2 import pyPenred
3 # Run the simulation configured in the "config.in" file
4 pyPenred.runFromFile("config.in")
```

Code 1: How to run a configuration file with pyPenred.

pyPenred includes other functionalities and more advanced simulation procedures. Further details can be found in the Section 7.

2.2 Manual compilation

PenRed requires a minimal set of tools to compile successfully. The following software must be installed beforehand:

- **Compiler:** A C++14-compatible compiler (e.g., GCC, Clang, MSVC, AppleClang, or Intel ICC).
- **CMake**¹ version 3.11 or grater.
- **Git**² for fetching penRed and other libraries source code.

These tools are cross-platform and available for Linux, Windows, and macOS. Once the prerequisites are installed, download the source code from the official GitHub repository³.

¹<https://cmake.org/download/>

²<https://git-scm.com/>

³<https://github.com/PenRed/PenRed>

This can be done either manually via the repository page or using *git* with the following command:

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

Compilation can be performed using the provided scripts or configured manually via the *CMake* tools. Most modern IDEs with *CMake* integration (e.g., Microsoft Visual Studio, VS Code) can also handle the compilation. Note that even if the IDE bundles *CMake* or *Git*, the standalone versions must still be installed in the system for full functionality.

2.2.1 Automatic scripts

After downloading the source code (and unzipping it if needed), navigate to the `src` directory. This folder contains the main `CMakeLists.txt` file along with automated compilation scripts. The script to use depends on the operative system:

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

```
bash compile.sh
```
- **Windows:** Execute the batch script *compile.bat*

The scripts will generate a `build` directory for temporary compilation files. Upon completion, a `compiled` folder will appear in `src`, containing all executables. The main penRed executable for running simulations is located at:

```
src/compiled/mains/pen_main
```

To customize the build, modify the script to enable/disable optional features:

- **DICOMs:** If enabled, PenRed's capabilities to read and simulate DICOM images will be activated. This option requires the DICOM toolkit (dcmtoolkit)[4]. The build system will use the version installed on the system, if available. Otherwise, the `dcmtoolkit` library and its required dependencies will be automatically downloaded and compiled. Please note that this process may take several minutes, depending on the performance of the computer.
- **MPI:** This option enables MPI simulations. It requires a library with an implementation of the MPI standard, such as OpenMPI[5] or MPICH[6].
- **Load Balance:** Enables load balancing between threads and MPI processes.

The *CMakeLists* file provide many other compilation options. Those options can be added to the script or configured via the *CMake* GUI or other IDEs.

2.2.2 CMake configuration

To access all available compilation options, users can either modify the compilation script by adding definitions manually or use *CMake* to select the desired options. Both the command line interface and the *CMake* graphical user interface (GUI) can be used for this purpose.

2.2.2.1 GUI

To configure penRed using the *CMake* GUI, open the application and set the source folder to the PenRed **src** directory. For the build folder, specify a path where the intermediate files will be stored (Figure 1). We recommend creating a dedicated directory for this purpose, such as a **build** folder within the **src** directory, to keep generated files separate from the source code.

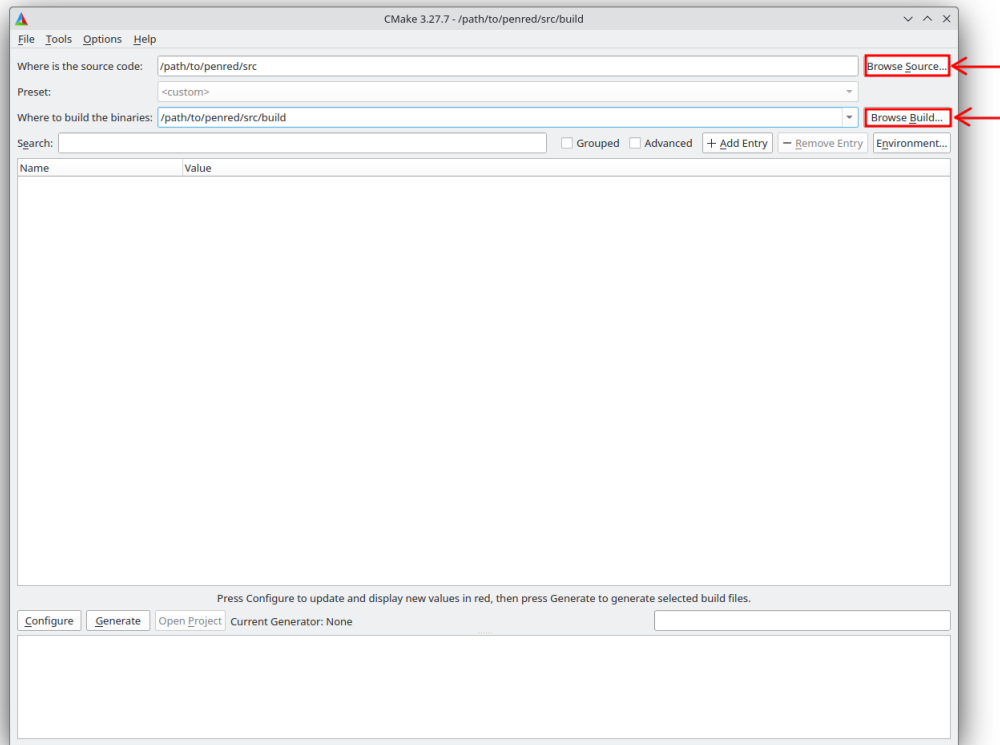


Figure 1: Set source and build paths with cmake user interface.

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

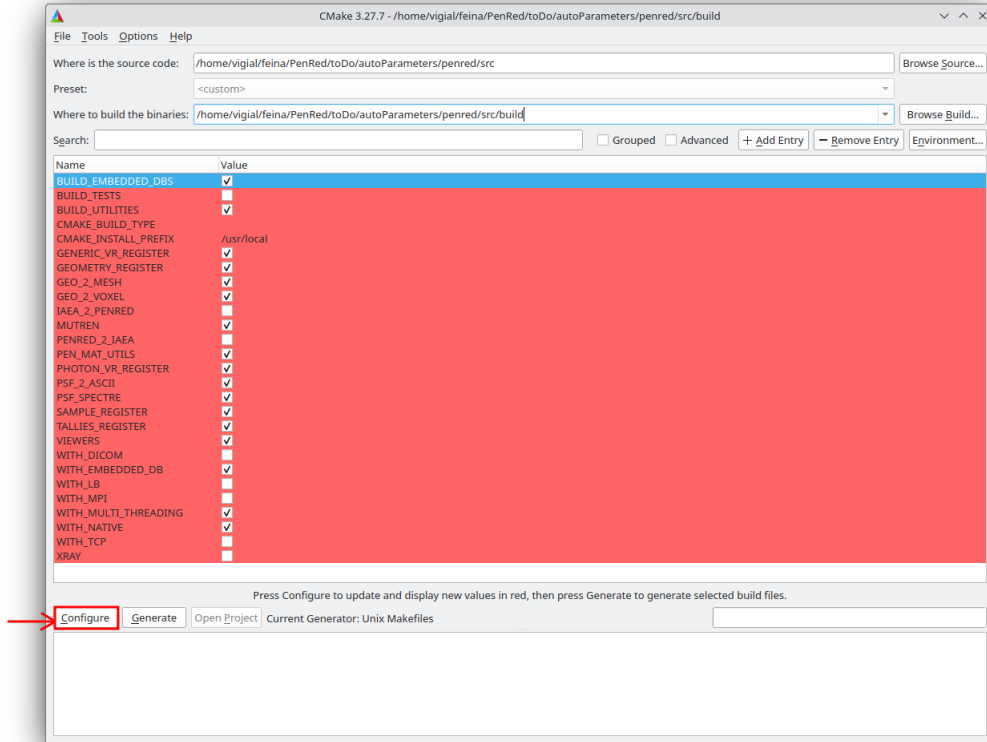


Figure 2: Configure project with cmake user interface, generating a list of compilation options.

After selecting all desired options, click the *Generate* button to create all necessary build files in the specified *build* folder. For IDE users (e.g., MSVC), the *Open Project* button will become active, allowing direct project loading into the development environment where the code can be built and installed.

If not, and the *Open Project* button remains disabled, the build must be done by command line. First, open a terminal (Command Prompt or Windows PowerShell) in the build folder. Then, run the *CMake* build and installation command:

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

Windows users should note the executable extension:

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

Upon successful completion, all generated executables will be located in the *compiled* subdirectory of PenRed's *src* folder. The main program is stored in:

```
src/compiled/mains/pen_main
```

2.2.2.2 Command line

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

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

```
mkdir build
```

```
cd build
```

Then, run the *ccmake* command within the build folder

```
ccmake ../
```

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

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

Windows users should note the executable extension:

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

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

```
src/compiled/mains/pen_main
```

2.2.3 Compilation options

This section summarizes some of the compilation options available in the penRed's CMake-List file:

- **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_MPI**: Handles MPI capabilities. Require an MPI library installed in the system.
- **WITH_NATIVE**: Enable or disable the compilation with native architecture optimizations.
- **WITH_DICOM**: Handles DICOM capabilities. Requires the *dcmTk* library. If *dcmTk* is not installed, it will be downloaded and compiled automatically.
- **FORCE_DICOM_DOWNLOAD**: Forces the build system to download and compile the *dcmTk* library, even if it is already installed in the system. This option must be used with **WITH_DICOM** enabled.
- **WITH_LB**: Enables or disables load balance capabilities. If enabled, some optional features related with the balance system can be configured:

- **WITH_SSL**: Enables or disables SSL secure connections for TCP or HTTPS communications. Requires OpenSSL libraries and its dependences.
- **WITH_HTTP**: Enables or disables connections to a HTTP/HTTPS rest api as balance server (Experimental feature). Requires the libCurl library.
- **BUILD_TESTS**: Enables or disable the compilation of test codes.

2.2.4 Basic usage

To run a pre-configured simulation (see Section 4), the `pen_main` executable must be placed in the same directory as the configuration file. For a configuration file named `config.in` (the default), simply double-click the executable to launch the simulation.

For custom configuration filenames, open a terminal (Command Prompt or Windows PowerShell) in the configuration directory and execute:

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

The simulation generates two log files by default:

- `configuration.log`: Records initialization and setup details.
- `simulation.log`: Tracks runtime progress.

Both filenames can be modified in the configuration file.

For MPI-enabled builds, execute with standard MPI launcher syntax:

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

where *Nprocesses* specifies the number of MPI processes. Additional `mpirun` options like `--hostfile` may be used as needed.

2.3 Examples

The penRed distribution includes an `examples` folder containing various pre-configured simulations. These demonstration cases, described in Section 8, are executable with either the pre-compiled package or manually compiled versions of the code.

For beginners, the simplest examples are located in the `examples/quadrics` subdirectory, which utilize quadric-based geometries. These provide a suitable starting point for understanding penRed's core functionality before progressing to more complex simulations. To understand the configurations in each example, the Section 4 describes the required files in a generic simulation.

3 Internal data format

To provide a unified format for input and configuration, PenRed uses *key/value* pairs where the *key* format is based on unix or web path 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, boolean (True or False), character, text (string) or an array of numbers, booleans 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 2 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 2: Internal data example

4 Simulation files

In this section, the required files to run a simulation are reviewed. A generic simulation uses the following files:

- **Executable:** The penRed executable is not necessary if pyPenred is used. See Section 2 for instructions on how to generate it.
- **Configuration:** This file contains all the configuration parameters. Those parameters include source, tallies, geometry and other definitions (Section 6). The format used to define each parameter is the one described in Section 3.
- **Geometry:** A definition file is required to describe the simulation's geometry system. The format of this file depends on the selected geometry type (see Section 6.3).
- **Materials:** These files contain the physical properties for all materials involved in the simulation. If not provided, penRed can generate them automatically by providing the corresponding composition and density, or by selecting a material from one of the available databases.

The paths to the **materials** and **geometry** files, along with other possible files depending on the configuration, are specified within the configuration file. These paths can be declared relative to the execution directory or as absolute paths.

A comprehensive description of all available configuration parameters can be found in Section 6. Furthermore, the *examples* directory contains several ready-to-run examples, each comprising a configuration file alongside its corresponding material and geometry files (Section 8). To execute any of these provided examples, please follow the instructions in Section 2.1 for pyPenred or Section 2.2.4 for the regular penRed executable.

5 Constants, parameters and definitions

This section summarises some important constants and values used within penRed.

5.1 Important definitions

- **Void region/volume:** Geometry region with material index 0. It contains no material, i.e. the particles crosses this materials without interacting.

- **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 geometry 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*. This basic state includes:
 - **Position (X,Y,Z):** Particle’s position in cm.
 - **Direction (U,V,W):** Direction cosines in each axis.
 - **Energy (E):** Particle’s energy in eV.
 - **Age (PAGE):** Particle’s age, including sampled age and time of flight, in seconds.
 - **Weight (WGHT):** Statistically weight. It defaults to 1.0, but can be changed due variance reduction techniques.
 - **Geometry:** Stores the body (**IBODY**) and material (**MAT**) indexes where the particle is located.
 - **ILBs:** Particle generation information. This parameter is described in the next item.
- **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 5.4) 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

- **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.

5.2 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.

5.3 Particle indexes

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

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

Table 3: Particle indexes and names.

5.4 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.

5.5 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

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.

6 Simulation configuration

This section addresses how to configure a penRed simulation by specifying the corresponding parameters within the configuration file.

All configuration files must follow the penRed data format, which is explained in Section 3. The parameter keys are classified into sections based on a prefix to specify the simulation characteristics. These sections are: sources, geometry, materials, tallies, variance reduction, and non-physics (simulation) parameters. The following sections describe each of these in detail. Table 8 summarizes the configuration sections and their corresponding prefixes.

Section	Prefix	Description
Sources	sources/...	Defines particle sources
Tallies	tallies/...	Extracts simulation results
Geometry	geometry/...	Defines the geometry system
Materials	materials/...	Specify properties for involved materials
Simulation	simulation/...	Defines non-physics parameters (optional)
Variance Reduction (VR)	VR/...	Apply VR techniques (optional)

Table 8: Configuration sections and their prefixes.

6.1 Materials

Material properties, such as cross sections, density, and composition, are defined in material files. These files can be generated at runtime by specifying the composition directly within the configuration file, as will be discussed in this section. Alternatively, they can be pre-generated using the provided utilities (Section 9.11) or pyPenred functions (Section 7)⁴.

Since the material files adhere to the same structure as those used in the original PENELOPE FORTRAN code, they can also be generated using the tools and databases from the original PENELOPE package, which are included in the penRed distribution. The material database is located in the folder:

`dataBases/penmaterials`

⁴<https://penred.github.io/PenRed/pyPenred/simulation.materials.html>

6.1.1 Common configuration

Regardless of whether the material file has been pre-created or is defined to be created at runtime, a set of common parameters can be defined to initialize the physical properties. First, to define a material in the configuration file, the user must adhere to the pattern shown in Code 3. Here, *materials* is the materials section prefix (a constant text), *material-name* serves as a user-selected text identifier, and *parameter/path* and *value* represent the parameters and values to be specified for that material.

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

Code 3: Material configuration pattern

For instance, Code 4 shows the copper material configuration used in the first quadric geometry based example provided in the package (`examples/quadric/1-disc`). In this code, the *material-name* is set to *cu*.

```
1 materials/cu/number 1
2
3 materials/cu/eabs/electron 1.0e3
4 materials/cu/eabs/positron 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 shown, the parameters required for each material specification are analogous to those needed by the PENELOPE FORTRAN version. The mandatory parameters are the following:

- **number:** The material *number* parameter is a numeric index employed to identify the material within the geometry system. This number must be greater than 0, as material 0 represents void, and less than or equal to the total number of materials.
- **filename:** Specifies the filename containing the material's physical properties. If the material file is generated at runtime, the resulting data is stored in this file. The path to the material file must be enclosed in quotes to be identified as text.

Other parameters are optional and will be set automatically to penRed default values. However, it is advisable to select these parameters based on the simulation characteristics. The following sections describe all available configuration parameters for materials.

6.1.1.1 Absorption energy

The absorption energy determines when a particle's simulation is halted. Any particle with an energy below this threshold is considered to be absorbed locally, finishing its transport simulation. Similarly, any secondary particles generated with an energy below this threshold will be considered locally absorbed and will not be simulated. This value can be provided on both a global and a per-material basis for each particle type. To set a default absorption energy value for all materials, use the following parameter path:

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

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

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

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

6.1.1.2 Limiting range

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

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

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

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

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

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

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

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

Notice that default range values can be overwritten by local material ranges. For example, the configuration shown in the Code 5 sets the the default range for electron and gamma to 0.1 cm for all materials. Then, a *infinite* absorption energy is set to positrons to avoid their transport. Finally, in the *water* material, the default range value is overwritten for gammas to 0.05 cm. Because the electron transport parameters are not specified, all materials use the default ones.

```
1
2 material-ranges/electrons 0.1
3 material-ranges/gamma 0.1
4
5 materials/cu/number 1
6 materials/cu/eabs/positron 1.0e35
7 materials/cu/filename "Cu.mat"
```

```

8
9 materials/Fe/number 2
10 materials/Fe/eabs/positron 1.0e35
11 materials/Fe/filename "Fe.mat"
12
13 materials/water/number 3
14 materials/water/range/gamma 0.05
15 materials/water/eabs/positron 1.0e35
16 materials/water/filename "H2O.mat"

```

Code 5: Configuration example with different absorption limits

6.1.1.3 Electron and positron transport parameters

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

- **C1:** Average angular deflection, $C1 \approx 1 - \langle \cos\theta \rangle$, 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.

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

6.1.2 Composition

A material can be created at runtime specifying the composition using the *elements* section. In this section, each path defines an element’s atomic mass (Z), and the corresponding value specifies the fraction by mass of this element in the material composition. This pattern is illustrated in Code 6.

```

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

```

Code 6: Material composition configuration pattern

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

```

1 materials/chamberTop/number 3
2
3 materials/chamberTop/elements/1 0.080538
4 materials/chamberTop/elements/6 0.599848
5 materials/chamberTop/elements/8 0.319614
6 materials/chamberTop/density 1.19500000E+00
7 materials/chamberTop/force-creation true
8
9 materials/chamberTop/eabs/electron 1.0e3
10 materials/chamberTop/eabs/positron 1.0e3
11 materials/chamberTop/eabs/gamma 1.0e3
12
13 materials/chamberTop/C1 0.00
14 materials/chamberTop/C2 0.00
15
16 materials/chamberTop/WCC 0.0
17 materials/chamberTop/WCR 0.0
18
19 materials/chamberTop/filename "pmma.mat"

```

Code 7: Material composition configuration pattern

In Code 7, the parameters for material file creation, in addition to the already discussed *elements* subsection, are:

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

6.1.3 Composition databases

Instead of manually specifying the composition of a material, users can load a material composition from PENELOPE's internal database or from other included external databases. To select a material from a database, the following parameters must be specified:

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

Notice that both values are text strings and must be enclosed in quotation marks. The first database contains all predefined materials from the PENELOPE library, which are listed in `dataBases/penmaterials/material-list.txt` and also in Appendix A of this document. For example, Code 8 shows a configuration where the material *Water, liquid* is selected from the *PENELOPE* material DB.

```

1 materials/water/number 1
2
3 materials/water/DB "PENELOPE"
4 materials/water/DB-material "Water, liquid"
5 materials/water/filename "water.mat"

```

Code 8: PENELOPE Material DB selection

To list the available external databases, and their materials, use the *compositionsDB* utility (Section 9.10).

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

```
1 materials/humeri/number 3
2
3 materials/humeri/DB "ICRP_AF"
4 materials/humeri/DB-material "Humeri_upper_cortical"
5 materials/humeri/filename "humeri.mat"
```

Code 9: Material composition DB configuration pattern

6.2 Particle sources

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

To configure a particle source parameter, users must adhere to the syntax structure shown in Code 10:

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

Code 10: Source configuration parameters

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

- *generic*: For sampling generic particle states
- *polarized*: For sampling gamma rays with explicit polarization states. An example can be found in `examples/quadrics/6-polarisation/config.in`.

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

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

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

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

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

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

6.2.1 Generic Sampling Sources

In addition to spatial, direction, energy and time sampling configurations, some generic-sampling source parameters have to be discussed. The Code 11 illustrates those parameters, both mandatory and optional, for a generic samplers-based source named *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 11: Generic sampling based source configuration parameters

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

- *nhist*: Number of histories to generate and simulate.
- *kpar*: Particle type ("gamma", "electron", or "positron") following the Table 3.
- *record-time*: Boolean flag to enable/disable time recording.

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

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

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

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

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

Code 12: Mono-energetic sampler selection

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

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

Code 13: Energy value specification

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

```

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 aperture
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 14: Complete source configuration example

The user can find many other pre-configured sources in the *examples* folder. A good start point are the *examples/quadric* folder, which includes quadric-based geometry examples. Available generic samplers are listed by type in the following sections.

6.2.2 Spatial source samplers

Spatial samplers determine the initial particle position. To configure the corresponding parameters, the key prefix to be used must fit the following pattern:

`sources/source-type/source-name/spatial/... value`

where *source-type* can be set to *generic* or *polarized* (for polarized gammas) and *source-name* is the name identifier for the configured source. Following in this section, all available spatial sampling sources are described.

6.2.2.1 Point

This spatial sampler sets the initial position (x, y, z) of generated particles to the specified 3D point, in cm. Following is a configuration example of this kind of spatial sampling corresponding to the 1-disc quadric 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 15: Spatial source POINT sampler

6.2.2.2 Box

This spatial sampler defines a 3D box in which particles are generated with a uniform probability. To define the box, the user must provide the following parameters:

- **type**: Must be set to "BOX".
- **position/x,y,z**: Specify the 3D center, in cm, of the sampling box.
- **size/dx,dy,dz**: Sets the box size in each axis, in cm.

An example of configuration file is shown below.

```
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 16: Spatial source BOX sampler

6.2.2.3 Cylindrical shell

This spatial sampler generates particles within a cylindrical shell. It is defined by subtracting a smaller cylinder from a bigger one where it is included. The sampling region corresponds to the shaded area in the Figure 3.

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

- **type**: Must be set to "CYLINDER".

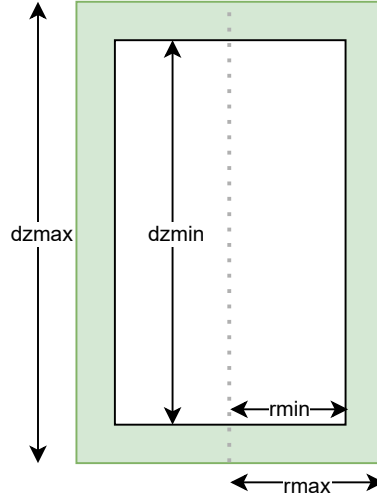


Figure 3: Cylindrical shell sampling region and values to define.

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

For example, the Code 17 shows the configuration to sample particles in a thin layer of radioactive paint on a brachytherapy seed located at the origin.

```

1 # Spatial sampling
2 #####
3
4 sources/generic/source1/spatial/type "CYLINDER"
5
6 # Set cyl size
7 sources/generic/source1/spatial/size/rmin 0.025
8 sources/generic/source1/spatial/size/rmax 0.025175
9 sources/generic/source1/spatial/size/dzmin 0.28
10 sources/generic/source1/spatial/size/dzmax 0.295
11
12 # Set position
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 0.0

```

Code 17: Spatial source CYLINDER sampler

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

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

Spatial Distribution tally (Section 6.4.20). Those files contains a ND regular mesh, defining the sampling probability for each bin or segment(1D)/pixel(2D)/voxel(3D) within a space region.

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

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

The dimension-specific parameters are as follows:

- **3D:**
 - **Type:** The tally type must be set to “3D_MEASURE”.
- **2D:**
 - **Type:** The tally type must be set to “2D_MEASURE”.
 - **plane:** Specifies the plane where the particles are sampled, using one of the following values: “xy”, “xz”, “yz”.
 - **constant-coordinate:** Specifies the value of the constant coordinate, in cm, according to the selected plane. For example, if the “xy” plane is selected, this parameter fixes the coordinate along the Z axis.
- **1D:**
 - **Type:** The tally type must be set to 1D_MEASURE”.
 - **axis:** Specifies which axis will be sampled. The allowed values are “x”, “y”, and “z”.
 - **x:** Sets the constant value along the X axis. Defaults to 0.
 - **y:** Sets the constant value along the Y axis. Defaults to 0.
 - **z:** Sets the constant value along the Z axis. Defaults to 0.

An example configuration is shown in code 18 for the 2D case.

```

1 sources/generic/source1/spatial/type "2D_MEASURE"
2
3 sources/generic/source1/spatial/filename "source.dat"
4
5
6 sources/generic/source1/spatial/constant-coordinate 0.0
7 sources/generic/source1/spatial/plane "xy"
8
9 sources/generic/source1/spatial/dx 0.0
10 sources/generic/source1/spatial/dy 0.0
11 sources/generic/source1/spatial/dz 0.0

```

Code 18: 2D Measure sampler configuration example

6.2.2.5 Image

The image spatial sampler determines the particle position according to the intensity values of a DICOM image, such as those from PET or SPECT modalities. Therefore, DICOM capabilities must be enabled during compilation (Section 2.2.3) to use this sampling method. This sampler is typically used in simulations involving two DICOM images: a geometry or structural image (e.g., a CT) for geometry definition and a source or functional image (e.g., SPECT or PET) for the spatial sampling distribution (Figure 4). In such cases, parameters must be provided to spatially align these two images.

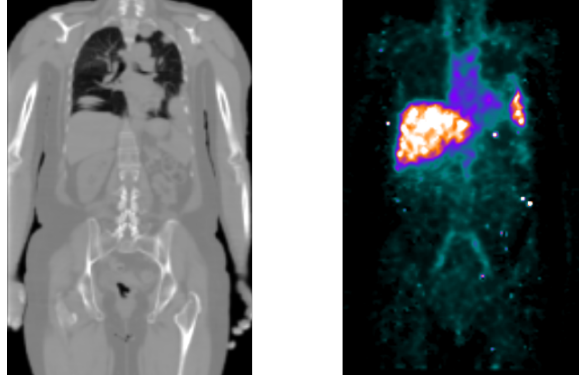


Figure 4: Structural (left) and functional (right) image pair from the *internal rt* example.

The available configuration parameters are:

- **type**: Must be set to "IMAGE".
- **directory**: Path to the folder containing the source DICOM files (functional image).
- **isocenter/x,y,z**: Optional parameters to specify the isocenter coordinates for the source DICOM, in cm. If not provided, these values are set to (0,0,0).
- **phantom-isocenter/x,y,z**: Optional parameters to specify the phantom (structural image) isocenter coordinates for the geometry DICOM, in cm. These parameters are used to align both images, and are incompatible with the *position* parameters. If not provided, these values are set to (0,0,0).
- **position/x,y,z**: Optional values to set a global translation applied to the sampled positions, in cm. If provided, all phantom-isocenter parameters are ignored and the final translation is calculated as:

$$T^i = position^i - isocenter^i \quad (2)$$

where i denotes the axis (x, y, z), and T is the final translation applied to sampled positions. If not provided, these values defaults to (0,0,0).

- **euler/omega,theta,phi**: Optional values to rotate the sampled positions. The rotation is applied **before** the translation, using the ZYZ axis convention:

$$R = R_z(\omega)R_y(\theta)R_z(\phi) \quad (3)$$

- **ActivityDistribution/save:** Specifies a filename to save the processed source image, enabling the loading of this pre-processed information in subsequent simulations. This parameter is optional.
- **ActivityDistribution/read:** Specifies a filename containing a pre-processed source to load the corresponding parameters. This file must have been produced by a previous simulation with *ActivityDistribution/save* enabled. This parameter is optional.
- **SpatialSamplingInfo:** A boolean flag to enable or disable saving the 3D distribution of generated particles. This option is used for debugging and defaults to false.
- **SpatialSamplingInfoSaveTo:** Specifies the filename to save the sampled particle information.

Isocenter information is used to calculate the translation required to align the functional and structural images. Therefore, phantom isocenter values are only used if the geometry type is a DICOM image. In that case, the translation applied along each axis is calculated as:

$$T^i = O_f^i - O_s^i - (I_f^i - I_s^i) + n_{vox}^i d_{vox}^i / 2 \quad (4)$$

where i denotes the axis (x, y, z); T^i is the translation applied along axis i ; O_f^i and O_s^i are the functional and structural image origins, respectively; I_f^i and I_s^i are the functional and structural image isocenters, respectively; n_{vox}^i is the number of voxels in the functional image along axis i ; and d_{vox}^i is the voxel size in the functional image along axis i . To specify a global translation directly, the user can set the *position* values, which forces the calculation to ignore the phantom isocenter and origin information from both images. In that case, the final translation is:

$$T^i = position^i - isocenter^i \quad (5)$$

A configuration sample for this spatial source can be found in the *internal rt* example (Code 19). In that case, as both isocenters match, the corresponding values have been set to (0,0,0). Therefore, the effective translation applied to sampled positions is:

$$T^i = O_f^i - O_s^i + n_{vox}^i d_{vox}^i / 2 \quad (6)$$

```

1
2 # Spatial sampling
3 #####
4
5 # Set image size
6 sources/generic/source1/spatial/type "IMAGE"
7 sources/generic/source1/spatial/directory "data/Dicom/SPECT"
8
9 # Set image center
10 #sources/generic/source1/spatial/position/x 24.918781
11 #sources/generic/source1/spatial/position/y 10.793281
12 #sources/generic/source1/spatial/position/z 37.955624
13
14 sources/generic/source1/spatial/isocenter/x 0.0
15 sources/generic/source1/spatial/isocenter/y 0.0
16 sources/generic/source1/spatial/isocenter/z 0.0
17
18 sources/generic/source1/spatial/phantom-isocenter/x 0.0
19 sources/generic/source1/spatial/phantom-isocenter/y 0.0

```

```

20 sources/generic/source1/spatial/phantom-isocenter/z 0.0
21
22 sources/generic/source1/spatial/ActivityDistribution/read "data/WalkerAliasing
    /ActivityInitialization.dat"
23
24 #sources/generic/source1/spatial/ActivityDistribution/save "data/
    WalkerAliasing/ActivityInitialization.dat"

```

Code 19: Image sampler configuration example

6.2.3 Direction samplers

Direction samplers determine the initial particle direction. To configure the corresponding parameters, the key prefix to be used must fit the following pattern:

`sources/source-type/source-name/direction/... value`

where *source-type* can be set to *generic* or *polarized* (for polarized gammas) and *source-name* is the name identifier for the configured source. This section defines available direction samplings and their parameters.

6.2.3.1 Cone

Particles generated by the source are sampled uniformly within a conical beam, which direction is specified in spherical coordinates. The required parameters are the polar angle (*theta*), the azimuthal angle (*phi*), and the semi-aperture angle (*alpha*), which together define the direction and extent of the cone. Note that all angles are expected to be provided in degrees.

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

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

```

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 20: Direction CONE sampler

6.2.3.2 Solid angle

In this case, the direction of the generated particles is sampled within a solid angle defined by configurable polar and azimuthal apertures. The required parameters for this directional sampler are:

- **type**: Must be set to **SOLID_ANGLE**.
- The **direction** components along each axis (u,v,w). This vector defines the central axis of the solid angle.
- The polar aperture, defined by the minimum (*theta0*) and maximum (*theta1*) values of the polar angle, in degrees. Both values must be contained in the interval [0,180].
- The azimuthal aperture, defined by the minimum value of the azimuthal angle (*phi0*) and its angular increment (*dphi*).

The polar and azimuthal apertures are defined relative to the specified central direction. Therefore, the sampling process is as follows:

- A random polar angle and a random azimuthal angle are sampled uniformly within the specified apertures, defining a point on the unit sphere.
- This sampled direction vector is then rotated from the reference Z-axis to align with the user-defined **direction** parameter.

The code snippet below (Code 21) demonstrates an example of a sphere section pointing in the +Z direction with a semi-aperture of 30 degrees across the entire azimuthal interval:

```

1
2 # Directional sampling
3 #####
4
5 sources/generic/source1/direction/type "SOLID_ANGLE"
6
7
8 # Set direction vector
9 sources/generic/source1/direction/u 0.0
10 sources/generic/source1/direction/v 0.0
11 sources/generic/source1/direction/w 1.0
12
13 # Set polar overture
14
15 sources/generic/source1/direction/theta0 0.0
16 sources/generic/source1/direction/theta1 30.0
17
18
19 # Set azimuthal overture
20 sources/generic/source1/direction/phi0 0.0
21 sources/generic/source1/direction/dphi 360.0

```

Code 21: Direction SOLID_ANGLE sampler

6.2.4 Energy samplers

Energy samplers determine the initial particles' kinetic energy. To configure the corresponding parameters, the key prefix to be used must fit the following pattern:

`sources/source-type/source-name/energy/... value`

where *source-type* can be set to *generic* or *polarized* (for polarized gammas) and *source-name* is the name identifier for the configured source. This section describes all available energy samplers and their parameters.

6.2.4.1 Monoenergetic

This energy sampler generates a monoenergetic beam of particles with the specified energy in eV. The only required parameter is the *energy* value, which must be a positive number. An example of its usage can be found in the configuration file shown below. These lines correspond to the example quadrics/4-x-ray-tube configuration 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 22: Energy samplers MONOENERGETIC

6.2.4.2 Intervals

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

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

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

The probabilities are not required to be normalized. The following code shows its usage within the configuration file corresponding to the *quadrics/3-detector* example (`detector1.in`). In this example, each interval is set with the same low and top energy limit values. Therefore, this configuration produce discrete spectrum lines rather than a continuous spectrum. Both, discrete spectrum lines and continuous spectrum bins can be combined in the same configuration.

```
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 23: Energy samplers INTERVALS

6.2.4.3 File spectrum

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

The format consists of a piecewise function where each entry (file row) in the spectrum contains two numbers. The first one specifies the starting energy of a channel, and the second specifies its unnormalized probability. The end energy for that channel is the starting energy defined in the next entry (file row), and so on. The probabilities must be non-negative and do not need 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 Code 24, where the character # is interpreted as a comment. Note that the spectrum file must contain only comments or valid spectrum values until the end of the spectrum. Blank lines are not allowed. However, all text beyond the end of the spectrum (marked by a negative probability) 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 24: File energy spectrum format

Note that the text following the two numerical values in each line is ignored by the program. In the previous code, it is only added to describe the defined spectrum.

This sampler only requires two configuration values. Like other samplers, the first one is the *type*, which must be set to “*FILE_SPECTRUM*”. The other is the spectrum filename which is introduced as a string, as is shown in the code 25.

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

Code 25: File spectrum configuration

6.2.5 Time samplers

Time samplers determine the initial particle age. To configure the corresponding parameters, the key prefix to be used must fit the following pattern:

`sources/source-type/source-name/time/... value`

where *source-type* can be set to *generic* or *polarized* (for polarized gammas) and *source-name* is the name identifier for the configured source.

6.2.5.1 Decay

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

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

The decay follows an exponential distribution, parameterized by the specified half-life.

6.2.6 Specific samplers

The behaviour of specific samplers depends on each type. Certain specific samplers, such as the Phase Space File sampler, completely define the particle state and ignore any configured generic samplers. Others, like the Computed Tomography sampler, work in conjunction with generic samplers (Section 6.2.1) to determine specific state parameters while handling the remaining aspects internally.

Specific samplers exhibit similar variability in handling particle types. The Phase Space File sampler, for example, dynamically determines both particle type and state from an external file during operation. In contrast, other specific samplers maintain fixed particle types that must be explicitly specified through standard configuration, as demonstrated in Section 6.2.1, Code 11.

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

6.2.6.1 Phase Space File (PSF)

The Phase Space File (PSF) source reads particle states from files generated by the PSF tally (Section 6.4.13). Additionally, IAEA pre-generated PSF files can be converted to penRed PSF format using the utility described in Section 9.3. The configuration requires these parameters:

Core Parameters:

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

Variance Reduction:

The **wght-window** parameter defines statistical weight boundaries [w_{\min} , w_{\max}] to apply Russian Roulette (for weights below minimum) and splitting (for weights above maximum),

while `split` sets the splitting multiplier. Therefore, particles whose weight is within the window remain unchanged.

Spatial Transforms:

Allows the user to rotate and translate, in this order, the read particle states via the following parameters:

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

A complete example from the configuration file in *quadrics/5-accelerator-2/* folder is shown in Code 26. This one rotates the read states 90 degrees by the *Y* axis. Then, translates the state 10 cm in the *Z* axis.

```

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/rotation/omega 0
13 sources/generic/source-psf/specific/rotation/theta 90
14 sources/generic/source-psf/specific/rotation/phi 0
15
16 sources/generic/source-psf/specific/translation/dx 0
17 sources/generic/source-psf/specific/translation/dy 0
18 sources/generic/source-psf/specific/translation/dz 10

```

Code 26: Phase Space File Source

The source simulation ends once the number of histories to simulate (*nhist*) is reached or when all PSF states are read.

6.2.6.2 Computed Tomography (CT)

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

- **Specific source**: particles are sampled from a previously generated Phase Space File (PSF). This source internally uses the PSF source (section 6.2.6.1), and inherits its parameters already defined: *filename*, *Emax*, *wght-window* and *nsplit*.

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

- **Generic source**: particles are sampled based on the directional, energy, and spatial distributions described in section 6.2.1. Since CT scans typically emit secondary particles, the configuration file must include the parameter *nSecondaries*. This specifies the number of secondary particles sampled per history before incrementing the history counter.

Regardless the used method (via PSF or generic samplers), the CT source requires three additional parameter groups:

- **Angular information:**

- **phi-ini** and **phi-end** (in degrees): specify the initial and final angular position of the source. Note that **phi-ini** = 0° corresponds to the source located on x-axis, with emission directed to the negative x-axis, as it is shown in Figure 5, and that **phi-end** must be \geq **phi-ini**. Therefore, the source spatial and direction sampling configuration must be set following some requirements, which will be discussed in this section.
- **nProjections**: number of projections during the CT rotation.

- **Position information:**

- [**CTx0**, **CTy0**, **CTz0**]: coordinates, in cm, of the CT rotation center (ISOCENTER).
- **rad**: CT radius, source-isocenter distance (cm).

- **Time information:**

- **tmin**: initial time stamp for the first projection (s).
- **dt**: time interval between projections (s).

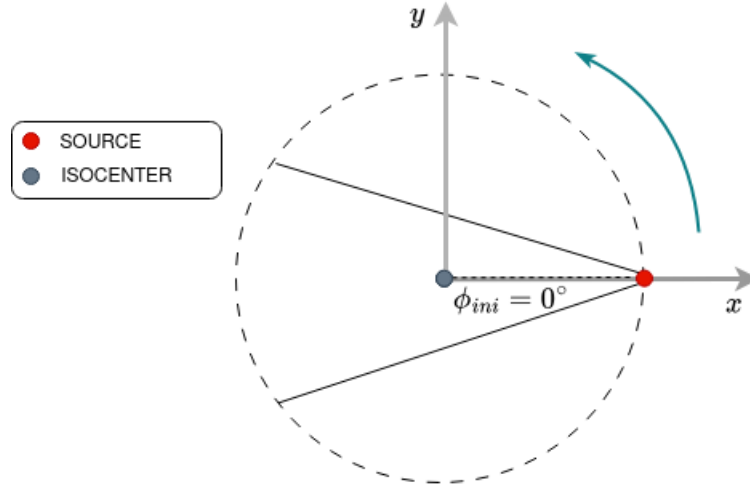


Figure 5: Schematic representation of the CT source emission

To ensure particles are aimed toward the CT center, the sampled beam direction must point along the -X axis. Regarding the spatial position, the CT source will rotate and translate each sampled particle to locate them at each projection position. The transformations applied to each sampled state position P are the following:

- Translate and rotate the initial state 3D position vector P according to the projection angle:

$$P' = R_z^i(P + (r, 0, 0)) \quad (7)$$

where P' is the projection position, R_z^i is the rotation corresponding to the projection i around the Z axis, and r is the CT radius configured by the user.

- Translate the projection position according to the configured CT center position P_{ct} :

$$P'' = P' + (CT_x, CT_y, CT_z) \quad (8)$$

The particle's sampled direction is rotated according to the projection angle, as is done for the position. Therefore, usually, the state's position should be sampled around the origin $(0, 0, 0)$ and the particle beam direction aiming to $-X$.

An example of CT source using PSF generated previously is shown in 27, while an example in which a generic source is used is shown in the code 28.

```

1 # Source 1
2 #####
3
4 sources/generic/source-ct/nhist 1.5e8
5 sources/generic/source-ct/specific/type "CT"
6 sources/generic/source-ct/specific/psf/filename "psf-merged.dat"
7 sources/generic/source-ct/specific/psf/Emax 1.0e5
8
9 sources/generic/source-ct/specific/psf/wght-window [5e-4, 1e-3]
10 sources/generic/source-ct/specific/psf/nsplit 100
11
12 # Angular information
13 #####
14 sources/generic/source-ct/specific/phi-ini 0
15 sources/generic/source-ct/specific/phi-end 360
16 sources/generic/source-ct/specific/nProjections 360
17
18
19 # Position information
20 #####
21 sources/generic/source-ct/specific/CTx0 0
22 sources/generic/source-ct/specific/CTy0 0
23 sources/generic/source-ct/specific/CTz0 0
24 sources/generic/source-ct/specific/rad 52.0
25
26 # Time information
27 #####
28 sources/generic/source-ct/specific/tmin 0.0
29 sources/generic/source-ct/specific/dt 120.0

```

Code 27: PSF Computed Tomography Source

```

1 # Source 1
2 #####
3
4 sources/generic/source-ct/nhist 1e8
5 sources/generic/source-ct/specific/type "CT"
6
7 sources/generic/source-ct/specific/nSecondaries 80
8
9 sources/generic/source-ct/kpar "gamma"
10
11
12 # Directional sampling

```

```

13 #####
14 sources/generic/source-ct/direction/type "CONE"
15
16 # Set theta
17 sources/generic/source-ct/direction/theta 90.0
18
19 # Set phi
20 sources/generic/source-ct/direction/phi 180.0
21
22 # Set oberture (alpha)
23 sources/generic/source-ct/direction/alpha 11
24
25
26 # Energy sampling
27 #####
28
29 sources/generic/source-ct/energy/type "MONOENERGETIC"
30
31 # Set energy
32 sources/generic/source-ct/energy/energy 1.20e5
33
34 # Spatial sampling
35 #####
36
37 sources/generic/source-ct/spatial/type "POINT"
38
39 # Set particle origin
40 sources/generic/source-ct/spatial/position/x 0.0
41 sources/generic/source-ct/spatial/position/y 0.0
42 sources/generic/source-ct/spatial/position/z 0.0
43
44
45 # Angular information to do
46 #####
47 sources/generic/source-ct/specific/phi-ini 0
48 sources/generic/source-ct/specific/phi-end 360
49 sources/generic/source-ct/specific/nProjections 360
50
51
52
53 # Position information
54 #####
55 sources/generic/source-ct/specific/CTx0 19.15
56 sources/generic/source-ct/specific/CTy0 19.15
57 sources/generic/source-ct/specific/CTz0 14.8
58 sources/generic/source-ct/specific/rad 106.0
59
60
61
62 # Time information
63 #####
64
65 sources/generic/source-ct/specific/tmin 0.0
66 sources/generic/source-ct/specific/dt 120.0

```

Code 28: Generic Computed Tomography Source

6.2.6.3 PENNUC

The PENNUC source has been adapted from the original FORTRAN code published in [9] and developed by CIEMAT, Laboratoire National Henri Becquerel, and Universitat de

Barcelona. This tool enables simulation of radioactive nuclide decay and subsequent use of the resulting particles in generic simulations. The decay characteristics for each nuclide, provided in the PENNUC format, can be obtained from a public database⁵.

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

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

For spatial sampling, the PENNUC source requires a generic spatial sampler, such as the box sampler (Section 6.2.2). Additionally, the source must be configured with the parameters shown in Code 29 and explained below:

- **type**: The source type must be set to PENNUC.
- **nucleide_filename**: Specifies the file containing decay data downloaded from the nucleide database⁶.
- **age**: Optional parameter to enable or disable age recording (default: `false`)
- **source-material**: Optional parameter to restricts particle generation to a specific material, resampling the position if necessary.
- **pennuclog_filename**: Optional parameter to specify the log file (default: `pennuc.dat`).

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

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

```

1 # PENNUC sampling
2 #####
3
4
5 sources/generic/source1/specific/type "PENNUC" (mandatory)
6 sources/generic/source1/specific/nucleide_filename "Co-60.nuc" (mandatory)
7 sources/generic/source1/specific/age true (optional)
8 sources/generic/source1/specific/source-material 1 (optional)

```

Code 29: PENNUC source configuration

6.2.6.4 Brachytherapy seeds source

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

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

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

⁵<http://www.lnhb.fr/home/nuclear-data/nuclear-data-table/>

⁶<http://www.lnhb.fr/home/nuclear-data/nuclear-data-table/>

- Seed center positioned at origin (0,0,0). This is usually considered as the geometrical centroid of the active material.
- Seed orientation aligned with the +Z axis

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

Configuration Parameters:

- **type**: Must be set to "BRACHY".
- **psf** subsection: Configures the internal PSF source (see Section 6.2.6.1).
- **seed-rotation**: Enables/disables per-seed rotation (default: `true`). When enabled, an extra rotation will be applied to the generated particles according to each seed orientation.

An example of a complete configuration is shown in the code 30.

```

1
2 #-----
3 # Source 1
4 #####
5
6 sources/generic/source-psf/nhist 5.0e7
7 sources/generic/source-psf/specific/type "BRACHY"
8 sources/generic/source-psf/specific/psf/filename "psf-merged.dat"
9 sources/generic/source-psf/specific/psf/Emax 1.40e6
10
11 sources/generic/source-psf/specific/psf/wght-window [5e-4, 1e-3]
12 sources/generic/source-psf/specific/psf/nsplit 10
13
14 sources/generic/source-psf/specific/seed-rotation false

```

Code 30: Brachytherapy source

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

6.3 Geometries

Geometry modules define the space in which particle transport is simulated. This involves defining volumes, known as bodies, which are filled with different materials, each possessing its own physical properties. Additionally, these modules handle the ray-tracing operations required to move particles through the geometry system, detecting when a particle crosses an interface, changes material, etc. This kind of modules usually require external files where the geometry is defined.

Each simulation is limited to a single geometry, whose configuration parameters are specified with the prefix *geometry* for correct identification. Like sources, geometries require their type to be specified, along with other type-specific parameters. All available geometries and their parameters are described in this section.

6.3.1 Geometry visualization

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

<https://github.com/PenRed/GeometryViewer/releases>

which includes the documentation to install and use the viewer.

6.3.2 Quadric

This geometry type inherits from the PENELOPE package and uses mathematical quadric surfaces to define volumetric regions through boolean operations of such elements.

This is the geometry type used in all the PENELOPE-based examples that have been translated to be reproduced with penRed (see the folder *examples/quadrics*). For instance, Code 31 shows the geometry configuration for the first simulation example (1-disc).

The geometries for this type are defined using an external file following the PENELOPE quadric format. This format is explained in the PENELOPE manual [3] in Chapter 6, **Constructive quadric geometry**. Moreover, PenRed is compatible with PENELOPE geometry files, allowing them to be used directly from PENELOPE simulations. To simplify its definition, penRed provides a Blender plug-in to construct this and other kinds of geometries using the Blender GUI. The plug-in description can be found in Section 9.1.

```
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 31: Complete geometry configuration example

The available parameters are described following:

- **type**: The geometry type must be set to **PEN_QUADRIC**.
- **input-file**: Specifies the relative or absolute path to the file where the geometry has been defined.
- **processed-geo-file**: specifies a file to generate a geometry report.
- **dsmax**: This optional parameter, specifies the maximum distance, in cm, allowed for electrons and positrons to jump when class II transport is active. This is used in thin regions to ensure a minimum number of steps in the specified body. Note that the body where the *dsmax* is applied is specified via its name/alias in the parameter path, **geometry/dsmax/alias**. For instance, in the previous code, the *dsmax* is applied to the body with the alias "1". It can be defined for each geometry body.
- **kdet**: Specifies the detector identifier to which the body belongs. Again, the body is specified using its alias in the parameter path, **geometry/kdet/alias**. It can be defined for each geometry body. Additionally, many bodies can be assigned with the same detector index, all of them conforming a single detector.

Optionally, it is possible to specify local absorption energies for any body and particle. In Code 32 we show an example where the local energy absorption for each particle is set to 10 keV for the body with alias "2". Note that the most restrictive absorption energy is

applied, whether the one 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 32: Configuration of geometry absorption energy example

6.3.3 Triangular Mesh Surfaces

This type of geometry uses triangular mesh surfaces to define the geometry system bodies. For this purpose, PenRed defines its own format for input files. However, with the provided Blender plug-in (Section 9.1), several mesh formats can be easily transformed to the PenRed one. The plug-in allows also to build this kind of geometries directly using the Blender GUI.

6.3.3.1 Input file format

In the examples folder, several triangular mesh based geometry files can be found along with the corresponding configuration file.

`examples/triangular_surfaces`

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

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

An example is shown in the Code 33. Notice that lines beginning with `#` are considered as comments and are ignored when the input is processed. In all geometries, one body acts as the **world**, i.e. it has no parent and all other bodies are included inside it. To specify which body is considered as the world, its parent name must be set to **void**. Therefore, **void** can't be used as a body name.

```
1
2 #MAT #N FACES #N VERTEX #NAME #PARENT NAME #N VERTEX GROUPS
3 002 40796 20400 Capsule Phantom 5
```

Code 33: Body metadata for triangular mesh geometries.

- **Vertex Groups:** Each vertex group consists of a first line with the group name followed by the number of vertices included in this group, along with a list of the vertex indexes, one per line. An example is illustrated in Code 34, where three vertex groups are defined.

```

1 # VERTEX GROUPS
2 #NAME #NVERTEX
3   back    0004
4   0002
5   0003
6   0006
7   0007
8 #NAME #NVERTEX
9   front_down  0002
10  0001
11  0005
12 #NAME #NVERTEX
13   front_up    0002
14   0000
15   0004

```

Code 34: Vertex group format.

Notice that this section is absent for bodies with zero vertex groups. An example with vertex groups can be found at the following path:

`src/lib/x-ray/source/baseAnode.geo`

Here, the vertex groups are utilized to construct an x-ray anode at runtime.

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

`vertex_number X Y Z`

An example is shown in the Code 35.

```

1 # VERTEX LIST
2 # Index  (X Y Z)
3 0000 +0.000000E+00 -8.000000E-02 +7.566000E-02
4 0001 +0.000000E+00 -8.000000E-02 -1.731400E-01
5 0002 +2.513000E-03 -7.996000E-02 +7.566000E-02
6 0003 +2.513000E-03 -7.996000E-02 -1.731400E-01
7 0004 +5.023000E-03 -7.984200E-02 +7.566000E-02
8 .
9 .
10 .

```

Code 35: Body vertex list in triangular mesh geometries.

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

- **Triangle list:** A list with all mesh triangles. As vertex, each one is specified in a single line, which contains the three vertex indexes that defines the triangle.

Notice that the order of the vertex is used to calculate the direction of the triangle normal vector. The normals are assumed to point outward from the body. Most 3D design softwares uses this schema by default and provide tools both to check the geometry normals and to recalculate them.

An input example is shown in the Code 36.

```

1 # FACES(triangles)
2 499 001 003
3 499 003 599
4 599 003 005
5 599 005 699
6 699 005 007

```

```

7 699 007 799
8 .
9 .
10 .

```

Code 36: Body triangle list in triangular mesh geometries.

6.3.3.2 Configuration

The mandatory configuration parameters to use this type of geometry are shown in the code 37 and are listed following,

- **type:** Must be set to **MESH_BODY**.
- **input-file:** Path to the input file with the geometry description. This must follow the format previously described (Section 6.3.3.1).

Optionally, the **report-file** parameter can be specified to print a geometry report. Regarding specific parameters for each body, the user can specify the **dsmax** and **kdet** parameters as is done for quadric geometries.

In this kind of geometries each body is split in several regions in order to accelerate the ray-tracing calculation. Each region contains a subset of the body triangles, allowing to check only the triangles belonging to crossed regions. In turn, this regions are grouped in super-regions, following the same approach and creating an octree structure. That clustering avoids checking many triangles and saves computational time.

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

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

These parameters can be different for each body. For example, in the code 37, the body named **Capsule_cyl** has an objective number of triangles per region of 50 and will try to group 20 regions in each super-region. Notice that the real number of elements in each cluster could be different depending on the body structure. For example, regions with a small number of triangles are merged with boundary regions.

```

1
2 geometry/type "MESHBODY" (mandatory)
3 geometry/input-file "seedMesh.geo" (mandatory)
4 geometry/report-file "report.geo" (optional)
5
6 geometry/RegionElements/Capsule_cyl 50 (optional)
7
8 geometry/SuperRegions/Capsule_cyl 20 (optional)

```

Code 37: Triangular mesh geometry configuration example.

Tinkering this parameters can significantly increase the simulation speed depending on the geometry structure and complexity. It could be worth it to perform some short simulations with different parameters in order to maximize the efficiency.

6.3.3.3 Vertex transforms

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

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

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

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

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

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

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

An example of transformations applied to a body named *anode* is shown at Code 38.

```

1 transforms/anode/backEnlarge/index 0
2 transforms/anode/backEnlarge/vertex-group "back"
3 transforms/anode/backEnlarge/transforms/enlargeY/type "TRANSLATION_Y"
4 transforms/anode/backEnlarge/transforms/enlargeY/index 0
5 transforms/anode/backEnlarge/transforms/enlargeY/ds -1.0

```

Code 38: Vertex group transformation configuration example.

6.3.3.4 Limitations

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

6.3.4 Voxel

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

This type of geometry accepts the following parameters:

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

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

- **enclosure-margin:** This parameter defines the margin, in cm, of a homogeneous enclosure containing the voxelized geometry (Figure 6). It is intended to take into account border effects, such as backscatter.

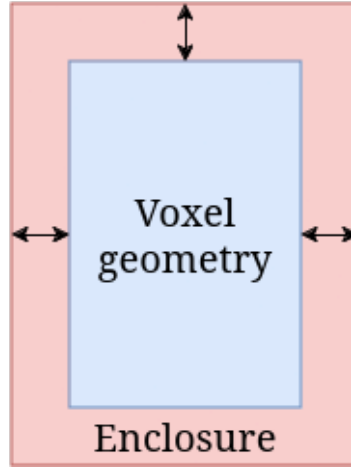


Figure 6: Voxelized geometry enclosure schema

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

An example of configuration file for voxelized geometries is shown in the code 39.

```

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/enclosure-margin 20
11 geometry/enclosure-material 5
12
13 geometry/dsmax/1 0.05
14 geometry/dsmax/2 0.02
15 geometry/dsmax/3 0.02
16 geometry/dsmax/4 0.50

```

Code 39: Configuration of voxelized geometry example

6.3.4.1 Binary format

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

src/tests/geometry/voxels/read_Dump.cpp

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

src/utilities/geometry/geo2voxel.cpp

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

As shown in the mentioned codes, the function to fill a voxel geometry class is named **setVoxels**, illustrated in code 40. The initialization of voxel materials and densities is not shown. The required parameters are:

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

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

```

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 40: Set voxels function and geometry file creation

6.3.4.2 ASCII format

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

- Three positive, non-zero integer values representing the number of voxels in the x, y, and z axes.
- Three positive, non-zero numeric values specifying the voxel size in the x, y, and z axes.

- Two values for each voxel: first, an integer specifying the voxel material index; second, a numeric value corresponding to the density factor. Both values must be positive and greater than zero, as void material is not allowed inside a voxelized geometry.

The first lines of an example file in ASCII format for a voxelized geometry are shown in Code 41. This example represents a voxelized matrix of $(100 \times 200 \times 120)$ voxels with a voxel size of $(0.1 \times 0.1 \times 0.2)$ cm.

```

1
2 100 200 120
3 0.1 0.1 0.2
4 1 1.0
5 1 1.0
6 1 1.1
7 2 0.8
8 2 0.9
9 .
10 .
11 .

```

Code 41: ASCII voxelized geometry example.

6.3.5 DICOM

Medical images are usually stored using the international standard of Digital Imaging and Communications in Medicine (DICOM). PenRed implements a DICOM geometry module to convert the DICOM file to a voxel geometry automatically, allowing simulating directly on DICOM images. This geometry type is only available if DICOM support is enabled during compilation (Section 2).

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 absolute or relative path to the folder containing the DICOM images. 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. It consists of an array defining a polynomial calibration curve used 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 will be discussed in this section. To define this parameter, the following pattern must be used:

geometry/calibration [a0,a1,a2,...,an]

where the a_i value correspond to x^i polynomial coefficient. Therefore, the transform applied to voxel intensities, once transformed by the slope and intercept parameters in the DICOM file, is:

$$\rho = \sum_{i=0}^n a_i x^i \quad (9)$$

where ρ is the final density and x is the voxel intensity value.

- **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 other 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 42, 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 42: DICOM intensity ranges configuration

- **ranges**: Subsection analogous to *intensity-ranges* but using density ranges instead of pixel values. Thus it could be used to specify the voxels material indexes via density ranges. This method requires a calibration curve to previously convert HU units to densities. The subsection consists of the following parameters,
 - **material**: Material index to assign.
 - **density-low**: Minimum density value for this range.
 - **density-top**: Maximum density value for this range.
- All voxels with a density value between $(density - low, density - top]$ will be assigned with the material index *material*.
- **contours**: This subsection allows to the user to assign materials and densities according to contours stored inside the DICOM image. Each subsection of this type defines a single contour which name is specified in the corresponding parameters paths. Notice that the contour name must coincide with the contour name stored in the DICOM file. This subsection contains the following parameters,
 - **material**: Default material to assign to this contour. This is an optional field. If it is not provided, the existing material assign will not be overwritten.
 - **density**: Default density to assign to this contour (g/cm^3). This is an optional field. If it is not provided, the voxel density will be not overwritten.
 - **priority**: A priority value to control which contours are overwritten by other ones. Contours with lower priority values will be overwritten by contours with higher priority values.

- **intensity-ranges**: Material and density can also be assigned defining intensity ranges specific for each contour. This method overwrites the specified default material and density for the contour. To define the ranges, use the same pattern explained for global **intensity-ranges**.
- **ranges**: Materials can also be assigned defining density ranges specific for each contour. This method overwrites the specified default material for the contour. To define the ranges, use the same pattern explained for global **ranges**.

For example, to configure a contour named *target*, with air in some regions, the configuration file should contain something like the lines shown in code 43.

```

1 geometry/contours/target/material 1
2 geometry/contours/target/density 1.05
3 geometry/contours/target/priority 1.0
4
5 geometry/contours/target/intensity-ranges/Air/material 2
6 geometry/contours/target/intensity-ranges/Air/low -2000
7 geometry/contours/target/intensity-ranges/Air/top -500
8 geometry/contours/target/intensity-ranges/Air/density 0.001290

```

Code 43: DICOM contour configuration

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

- **enclosure-margin**: Like the voxel based geometry case, this parameter specifies the distance between the DICOM mesh corner and the limit of an external enclosure, **in cm**.
- **enclosure-material**: Specify the enclosure material index.
- **print-ASCII**: This optional boolean configuration parameter can be set to *true* to export the processed DICOM data in ASCII format. The exported data include detailed information about voxel materials and densities, contours, and seeds. This information is stored in the following files:
 - **dicomASCII.rep**: Contains the voxelized geometry data, including the material ID and density assigned to each voxel.
 - **dicomContourMask.dat**: Contains a mask where each voxel is assigned the index of the contour it belongs to. Voxels not within any contour are assigned a value of -1.
 - **dicomContours.dat**: Contains the 3D coordinates of the points defining each contour, with all penRed-specific transformations applied.
 - **roi_contour.dat**: One file is created for each DICOM contour (Region of Interest). Each file contains a binary mask, assigning a value of 1 to voxels inside the specific contour and 0 to voxels outside of it. Notice that this masks don't take into account other contours.
 - **dicomSeeds.dat**: Stores the positions, trajectories, and relative weights of brachytherapy seeds, as defined in the RTPLAN DICOM file (including time weights). The seed positions have all penRed-specific transformations applied.

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 7.

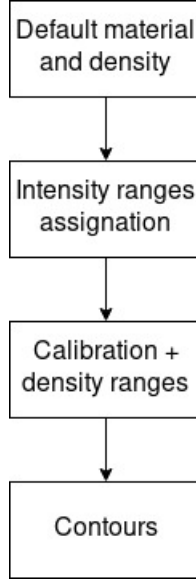


Figure 7: DICOM voxels material and density assign methods hierarchy.

The allowed image modalities for the DICOM geometry type are, by the moment, Computed Tomography (CT), Ultrasound (US), Radiotherapy Structure Set (RTSTRUCT) and Radio-therapy Plan (RTPLAN). Positron emission tomography (PT) and SPECT are also accepted, but not for geometry construction purposes. Instead, PT and SPECT images can be used to define a spatial sampling (Section 6.2.2.5).

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

6.3.5.1 Constraints

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

The constraints are specified with the following pattern:

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

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

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

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

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

6.3.6 Combo

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

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

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

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

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

`geoName_bodyName`

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

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

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

`geometries/accelerator/path/to/parameter`

The parameters to be specified for each geometry are:

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

An example of configuration is shown in the Code 44, where three quadric geometries, named 1, 2, 3, are combined.

```

1 geometry/type "COMBO"
2
3 geometry/geometries/1/priority 2
4 geometry/geometries/2/priority 1
5 geometry/geometries/3/priority 0
6
7 geometry/geometries/1/config/type "PEN_QUADRIC"
8 geometry/geometries/1/config/input-file "1.geo"
9 geometry/geometries/1/config/processed-geo-file "report1.geo"
10
11 geometry/geometries/2/config/type "PEN_QUADRIC"
12 geometry/geometries/2/config/input-file "2.geo"
13 geometry/geometries/2/config/processed-geo-file "report2.geo"
14
15 geometry/geometries/3/config/type "PEN_QUADRIC"
16 geometry/geometries/3/config/input-file "3.geo"
17 geometry/geometries/3/config/processed-geo-file "report2.geo"

```

Code 44: Geometry “COMBO” type configuration example.

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

A more interesting usage case is the combination of different types of geometries. For example, the Code 45 shows the combination of a brachytherapy seed geometry constructed using the quadric geometry type with a DICOM based geometry, where the materials are assigned via contours. In this example, the seed overwrites the DICOM.

```

1
2 geometry/type "COMBO"
3
4 geometry/geometries/seed/priority 0
5 geometry/geometries/DICOM/priority 1
6
7 geometry/geometries/seed/config/type "PEN_QUADRIC"
8 geometry/geometries/seed/config/input-file "seed.geo"
9 geometry/geometries/seed/config/processed-geo-file "report.geo"
10
11 geometry/geometries/DICOM/config/type "DICOM"
12 geometry/geometries/DICOM/config/directory "test_mamaHDR"
13
14 #Contour assign
15 geometry/geometries/DICOM/config/enclosure-margin 20
16 geometry/geometries/DICOM/config/enclosure-material 4
17
18 geometry/geometries/DICOM/config/default/material 4
19 geometry/geometries/DICOM/config/default/density 1.2000E-03
20
21 geometry/geometries/DICOM/config/contours/body/material 1
22 geometry/geometries/DICOM/config/contours/body/density 1.0
23 geometry/geometries/DICOM/config/contours/body/priority 2.2
24
25 geometry/geometries/DICOM/config/contours/heart/material 2
26 geometry/geometries/DICOM/config/contours/heart/density 1.5
27 geometry/geometries/DICOM/config/contours/heart/priority 2.6
28
29 geometry/geometries/DICOM/config/contours/skin/material 3
30 geometry/geometries/DICOM/config/contours/skin/density 1.1
31 geometry/geometries/DICOM/config/contours/skin/priority 2.3
32
33 geometry/geometries/DICOM/contours/left_lung/material 4

```

```

34 geometry/geometries/DICOM/contours/left lung/density 1.2E-03
35 geometry/geometries/DICOM/contours/left lung/priority 2.3
36
37 geometry/geometries/DICOM/contours/right lung/material 4
38 geometry/geometries/DICOM/contours/right lung/density 1.2E-03
39 geometry/geometries/DICOM/contours/right lung/priority 2.3
40
41 geometry/geometries/DICOM/contours/left ribs/material 5
42 geometry/geometries/DICOM/contours/left ribs/density 1.92
43 geometry/geometries/DICOM/contours/left ribs/priority 2.3
44
45 geometry/geometries/DICOM/contours/right ribs/material 5
46 geometry/geometries/DICOM/contours/right ribs/density 1.92
47 geometry/geometries/DICOM/contours/right ribs/priority 2.3
48
49 geometry/geometries/DICOM/contours/ptv/material 6
50 geometry/geometries/DICOM/contours/ptv/density 1.0
51 geometry/geometries/DICOM/contours/ptv/priority 2.5

```

Code 45: Geometry “COMBO” with quadric and DICOM geometries.

6.4 Tallies

Tallies are the modules used to retrieve information from the simulation, such as absorbed energy or dose, particle fluence, etc. Therefore, at least one tally should be defined in each simulation. Configuring a tally is similar to configuring sources and other modules. The pattern to configure tallies is exemplified in code 46, where `tallies` is a constant text, *tally-name* is a user-defined unique name for the tally, and *parameter/path* and *value* set the tally parameter keys and values, which depend on the tally type.

```

1 tallies/tally -name/parameter/path value

```

Code 46: Tally configuration pattern

As with other modules, penRed includes several tally types, so the user must specify the parameter *type* for each created tally. In this section, the available tallies and their parameters will be described. Configuration examples are also provided for each tally.

6.4.1 Output files

Usually, each tally produces one or more files to store the results extracted from the simulation. Where these files are stored can be optionally specified for each tally by setting the parameter *outputdir*. Both the type and *outputdir* parameters are common to all tallies and can be set in the configuration file as shown in Code 47.

```

1 tallies/tally -name/type "TYPE.STRING"
2 tallies/tally -name/outputdir "path/to/results/dir/"

```

Code 47: Tally common parameters

When defining the *outputdir* value, the remaining path after the last slash (/” or \” for Unix and Windows based systems respectively) will be appended as a prefix to the results filename instead of being interpreted as a directory. Table 9 shows some cases to exemplify this behaviour applied to a file named *filename*, where the left column represents the *outputdir* value specified in the configuration file, and the right column shows the final file path in the OS file system. If the specified path does not exist, i.e., one or more directories have not been created by the user, penRed will append the prefix “*not-found*” to the filename and create the corresponding file in the execution directory to avoid losing the simulation progress.

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

Table 9: Tally *outputdir* behaviour examples.

The final path for each tally results file is constructed using the following pattern:

outputdir + *tally-name-thi-filename*

where *outputdir* is the parameter discussed before, *tally-name* is the specified name for this tally in the configuration file, *thi* flags the thread number for multithreading executions, and *filename* is the specific tally results file name. In multithreading executions, the file corresponding to thread 0 is the one storing the added results of all threads. Similarly, in MPI executions, the file corresponding to thread 0 within rank 0 stores the final added results.

6.4.2 Intervals

Usually, tallies need to define spatial, energy, angular, or time intervals to discretize the recorded data into several bins or meshes. The convention used in penRed for these bin intervals is $[min, max)$, including the lower interval boundary but excluding the upper bound value. For example, Code 48 shows the configuration corresponding to a cylindrical dose distribution tally. This one requires limits for radial distance (*rmin* to *rmax*), number of radial bins *nbinsr*, limits for the *z* axis (*zmin* to *zmax*) and the number of *z* bins (*nbinsz*). In that example, *rmax* and *zmax* are outside of their respective intervals, but *rmin* and *zmin* are included.

```

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 1
7 tallies/cylDoseDistrib/zmax 30.0
8 tallies/cylDoseDistrib/nbinsz 60

```

Code 48: Tally configuration pattern

6.4.3 Output data

In addition to output files, many tallies allow extracting the results data directly from the simulation class using a unified format. This feature avoids the need to read the data from the disk when creating custom programs or when simulating within the Python environment using pyPenred. As each tally extracts different types and numbers of results, the structure of the returned data will be specified for each described tally in the following sections.

6.4.4 Cylindrical Dose Distribution

This tally measures the absorbed dose in eV/g per history for each volumetric element in a cylindrical mesh. The available parameters for this tally are described below:

- **type “CYLINDRICAL_DOSE_DISTRIB”**: Type name of the tally.
- **print-xyz**: When enabled (true), two extra values per bin coordinate are printed in the results file, corresponding to the lower boundary and average value of each coordinate. For the Z and azimuthal (ϕ) coordinates, the average value is taken at the bin midpoint. For the radial coordinate, the average is weighted using the expression:

$$r_{average} = \frac{2}{3} \frac{(r_{low} + dr)^3 - r_{low}^3}{dr(dr + 2r_{low})} \quad (10)$$

where $r_{average}$ is the resulting bin weighted radius, r_{low} is the low boundary for this bin, and dr is the radial bin size.

- **rmin**: Minimum radial coordinate value (in cm). Must be greater than zero and less than $rmax$.
- **rmax**: Maximum radial coordinate value (in cm).
- **nbinsr**: Number of radial bins.
- **zmin**: Minimum Z -axis coordinate value (in cm). Must be less than $zmax$.
- **zmax**: Maximum Z -axis coordinate value (in cm).
- **nbinsz**: Number of bins along the Z -axis.
- **nbinsPhi**: Number of angular bins (optional). Defaults to 1 if not specified.

An example configuration for this tally is shown in Code 49, extracted from the configuration file *disc.in* located in the **example/quadrics/1-disc** folder.

```

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 49: Cylindrical dose distribution tally configuration

6.4.4.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Dose**: Dose distribution in eV/g per history for the cylindrical mesh. The mesh dimensions are, in order radius (cm), azimuthal angle (ϕ) in degrees and Z in cm.
- **Energy**: Absorbed energy distribution in eV/history for the cylindrical mesh. The mesh dimensions are, in order radius (cm), azimuthal angle (ϕ) in degrees and Z in cm.

6.4.5 Emerging Particles Distribution

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

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

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

The configuration parameters for this tally are the following:

- **type “EMERGING_PART_DISTRIB”:** Type name of the tally.
- **emin:** Minimum energy value in eV. Must be lower than *emax*.
- **emax:** Maximum energy value in eV.
- **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.

Code 50 shows a configuration example of this tally, which has been extracted from the configuration file *detector1.in* located within the folder *examples/quadrics/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 50: Emerging particles tally configuration

6.4.5.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Upbound emerging particles:** Energy spectrum for particles leaving the geometry system with a positive Z direction ($W > 0$). Results are normalized to the energy bin width, reported in units of 1/eV per history. The results are split by particle type, reporting a total of three spectra for, in order, electrons, photons, and positrons.
- **Downbound emerging particles:** Energy spectrum for particles leaving the geometry system with a negative or zero Z direction ($W \leq 0$). Results are normalized to the energy bin width, reported in units of 1/eV per history. The results are split by particle type, reporting a total of three spectra for, in order, electrons, photons, and positrons.

- **Angular emerging particles:** Spherical angular distribution for particles leaving the geometry system. Results are normalized to the bin steradian, reported in units of 1/sr per history. The used dimensions are, in order, azimuthal and polar angle, both in degrees. The results are split by particle type, reporting a total of three spectra for, in order, electrons, photons, and positrons.

6.4.6 Energy Deposition Body

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

- **type “EDEP_BODY”:** Type name of the tally.

A configuration example of this tally is show in the code 51, which belongs to the file *plane.in* located in the folder **examples/quadrics/2-plane**.

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

Code 51: Energy deposition in body tally configuration

6.4.6.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Energy deposition:** Absorbed energy in each geometry body. Results are reported in eV per history.

6.4.7 Energy Deposition Material

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

- **type “EDEP_MAT”:** Type name of the tally.

The code 52 shows a configuration example obtained from the configuration file *detector1.in* belonging to the example 3-detector-1.

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

Code 52: Energy deposition in material tally configuration

6.4.7.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Energy deposition:** Absorbed energy in each material. Results are reported in eV per history.

6.4.8 Impact Detector

This tally can measure different magnitudes related with a defined geometry detector, including the fluence spectrum, particle energy spectrum, energy deposition spectrum, and particle age spectrum. Each one is described below:

- **Fluence:** Measures the spectral fluence integrated over the detector volume, expressed in cm/eV per history. The output file provides the fluence for each particle type as well as the total fluence within the specified detector.
- **Energy spectrum:** Reports the energy spectrum of particles entering the specified detector volume. Particles generated inside the detector, such as secondary particles, are not included in this tally. The units are given in 1/(eV·history). The output file contains the energy spectrum for each particle type.
- **Age:** Records the age distribution of particles impacting the designated detector, with units in 1/(seconds·history). The output file contains the probability distribution across different time intervals for all simulated particles.
- **Energy deposition:** Reports the history energy deposition spectrum measured in the specified detector. The units are given in 1/(eV·history). The output file contains the probability distribution across energy intervals, considering all simulated histories.

The configuration parameters are described following:

- **type “IMPACT_DET”:** Type name of the tally.
- **detector:** Detector index where the measurements of this tally are taken.
- **fluence:** Enable/disable fluence measurements. Defaults to **false**.
- **spectrum:** Enable/disable spectrum measurements. Defaults to **false**.
- **energy-dep:** Enable/disable energy deposition measurements. Defaults to **false**.
- **emin:** Minimum energy, in eV, value for tallied particles in fluence, energy spectrum, energy deposition and **age spectrum** measurements.
- **emax:** Maximum energy, in eV, value for tallied particles in fluence, energy spectrum, energy deposition and **age spectrum** measurements. Must be greater than *emin*.
- **nbin-energy:** Number of energy bins, which is used for fluence, energy spectrum and energy deposition. Must be at least 1.
- **linearScale-fln:** Determines if fluence measurements will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). Defaults to linear scale (**true**).
- **linearScale-spc:** Determines if energy spectrum measurements will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). Defaults to linear scale (**true**).
- **linearScale-edep:** Determines if energy deposition measurements will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). Defaults to linear scale (**true**).
- **age:** Enable/disable age spectrum measurements. Defaults to **false**.
- **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, in seconds.
- **age-max**: Maximum value of the age interval, in seconds. Must be greater than *age-min*.
- **linearScale-age**: Determines if age spectrum measurements will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). Defaults to linear scale (**true**).

To exemplify the tally usage, the next two examples show different configurations. First, Code 53 shows the configuration corresponding to the example located at `quadrics/2-plane`, where 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 (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 53: Impact detector tally configuration example 1

Secondly, Code 54 shows the configuration for the example located at `quadrics/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 54: Impact detector tally configuration example 2

6.4.8.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Fluence**: Fluence energy spectrum integrated over the detector volume. Results are normalized to the energy bin width, reported in units of cm/eV per history. The results are split by particle type, reporting a total of three spectra for, in order, electrons, photons, and positrons. These results are only generated if fluence is enabled in linear scale mode. Otherwise an empty object is returned.
- **Spectrum**: Energy spectrum of particles reaching the detector volume. Results are normalized to the energy bin width, reported in units of 1/eV per history. The results are split by particle type, reporting a total of three spectra for, in order, electrons, photons, and positrons. These results are only generated if spectrum is enabled in linear scale mode. Otherwise an empty object is returned.

- **Energy deposition:** History deposited energy spectrum in the detector volume. Results are normalized to the energy bin width, reported in units of 1/eV per history. This result is only generated if energy deposition is enabled in linear scale mode. Otherwise an empty object is returned.
- **Age:** Age spectrum of particles reaching the detector volume. Results are normalized to the time bin width, reported in units of 1/s per history. This result is only generated if age spectrum is enabled in linear scale mode. Otherwise an empty object is returned.

6.4.9 Spatial Dose Distribution

This tally measures the 3D absorbed dose distribution along the intervals $[xmin, xmax)$, $[ymin, ymax)$, $[zmin, zmax)$ in cm, in a voxelized regular mesh. The units of dose values are eV/g per history. In addition, if enabled, the tally reports the depth dose distribution along the z coordinate in eV/(g/cm²) units. The configuration parameters are described following:

- **type “SPATIAL_DOSE_DISTRIB”:** Type name of the tally.
- **xmin:** Minimum value of coordinate x , in cm.
- **xmax:** Maximum value of coordinate x , in cm. Must be greater than $xmin$.
- **nx:** Number of x bins. Must be, at least, 1.
- **ymin:** Minimum value of coordinate y , in cm.
- **ymax:** Maximum value of coordinate y , in cm. Must be greater than $ymin$.
- **ny:** Number of y bins. Must be at least 1.
- **zmin:** Minimum value of coordinate z , in cm.
- **zmax:** Maximum value of coordinate z , in cm. Must be greater than $zmin$.
- **nz:** Number of z bins. Must be at least 1.
- **print-depthDose:** Enables/disables printing depth dose information. Notice that this depth dose is discretized only in the Z axis, i.e. scores all energy depositions regardless the X and Y component. Therefore, in voxelized geometries, the enclosure is considered in the calculus. It is disabled by default.

Code 55 shows an example extracted from the configuration file *disc.in*, located at *examples/quadratics/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)
12 tallies/SpatialDoseDistrib/print-depthDose false   (optional)

```

Code 55: Spatial Dose Distribution tally configuration

6.4.9.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Dose:** Dose distribution in eV/g per history for the mesh element (voxel). The mesh dimensions are, in order X (cm), Y (cm) and Z (cm).

6.4.10 Angular Detector

This tally reports the energy spectrum of detected particles, filtered by their direction. Only particles whose direction falls within the specified azimuthal and polar angle intervals will be detected. Note that neither the particle direction nor the filtering angles **are related to the detector's orientation**; instead, they are defined relative to the Z-axis (0, 0, 1).

The energy spectrum is normalized to both the energy bin width and the solid angle, reported in units of $1/(\text{eV}\cdot\text{sr}\cdot\text{history})$. The parameters used to configure this tally are described below:

- **type “ANGULAR_DET ”:** Tally type name.
- **detector:** Detector index where the angular detector will be calculated.
- **emin:** Minimum energy value, in eV.
- **emax:** Maximum energy value, in eV. Must be greater than *emin*.
- **theta1:** Polar angle minimum value, in degrees.
- **theta2:** Polar angle maximum value, in degrees. The polar interval must be in the range (0, 180) and *theta2* must be greater than *theta1*.
- **phi1:** Azimuthal angle minimum value, in degrees.
- **phi2:** Azimuthal angle maximum value, in degrees. 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 will be tallied using a liner scale (**true**) or a logarithmic scale (**false**). Defaults to linear scale (**true**).

An example is shown in Code 56 belonging to the configuration file used by the example `examples/quadrics/1-disc-novr` (*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 56: Angular detector tally configuration

6.4.10.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Energy spectrum:** Energy spectrum of incident particles which direction falls within the specified azimuthal and polar intervals. Results are normalized to the energy bin width and the solid angle steradian, reported in units of $1/(eV \cdot sr)$ per history. The results are split by particle type, reporting a total of three spectra for, in order, electrons, photons, and positrons. These results are only generated for linear scale mode. Otherwise an empty object is returned.

6.4.11 Particle Generation

This tally reports information about the number of primary and secondary particles simulated. It reports the number of primary particles escaping the geometry up-bound and down-bound, according to their direction, and the number of particles absorbed in the geometry system. It also calculates the probabilities for secondary particles to escape up-bound, down-bound, or be absorbed.

This tally does not require any parameters to be specified in the configuration file other than the tally type.

- **type “SECONDARY_GEN”:** Tally type name.

An example configuration for this tally is shown in Code 57.

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

Code 57: Particle generation tally configuration

6.4.11.1 Output format

This tally does not provide any results when requested through pyPenred or the C++ simulation class. The results must be read from the generated files.

6.4.12 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. Tally configuration parameters are listed following:

- **type “SPHERICAL_DOSE_DISTRIB ”:** Type name of the tally.
- **print-xyz:** Can be enabled **true** to add the low and average coordinate values for each bin. For angular bins, the average value is set to the midpoint. For the radial coordinate, the average is weighted using the expression:

$$r_{average} = \frac{3(r_{low} + dr)^4 - r_{low}^4}{4(r_{low} + dr)^3 - r_{low}^3} \quad (11)$$

where $r_{average}$ is the resulting bin weighted radius, r_{low} is the low boundary for this bin, and dr is the radial bin size.

- **rmin:** Minimum value, in cm, of radial coordinate. Must be greater than zero.

- **rmax**: Maximum value, in cm, 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.

A configuration example is shown in Code 58.

```

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

```

Code 58: Spherical dose distribution tally configuration

6.4.12.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Dose distribution**: Dose distribution in eV/g per history for the spherical mesh. The mesh dimensions are, in order, radius (cm), polar angle (θ) and azimuthal angle (ϕ), both in degrees.

6.4.13 Phase Space File (PSF)

This tally generates a particle phase space file (PSF) storing all particles impacting the specified detector. Additionally, the impacting particles can be filtered by energy range and particle type. The available tally parameters are listed below:

- **type "PSF "**: Type name of the tally.
- **detector**: Detector index where this tally will be calculated.
- **emin**: Minimum energy value, in eV.
- **emax**: Maximum energy value, in eV. Must be greater than *emin*.
- **particles**: Section to enable or disable particle types recording. In this section, the user can specify the following parameters:
 - **default**: This optional boolean value enables (true) or disables (false) the particle recording for all particle types. Defaults to true.
 - **particle-name**: This optional boolean parameter, where *particle-name* must be replaced by a valid particle name (Table 3), disables (false) or enables (true) recording of the specified particle type. If defined, it overrides the *default* parameter.

For example, Code 59 shows a configuration where only gamma particles with a energy below 6.1 MeV are recorded.


```

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)
5 tallies/psf/particles/default false (optional)
6 tallies/psf/particles/gamma true (optional)

```

Code 59: Example of phase space file tally configuration where only gamma particles are recorded

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

6.4.13.1 Output format

This tally does not provide any results when requested through pyPenred or the C++ simulation class. The results must be read from the generated files.

6.4.14 Kerma track length estimator

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

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

Each filename is specified following this pattern:

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

The data files format consists of two columns. The first one stores the energy in eV and the second the corresponding μ_{en}/ρ value in (cm^2/g). The provided points must contain the energy range $[emin, emax]$. However, is not required a constant energy distance between points. Furthermore, there are no limit to the number of points in each data file. Notice that materials with no μ_{en}/ρ data filename provided will be ignored by the tally. To obtain the μ_{en}/ρ coefficients manually, the utility **mutren** can be used. This one is included in the penRed package and described in the section 9.2.

- **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**: Initial 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**: Initial 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**: Initial 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 many mesh types can be activated in the same tally. In these cases, the tally will generate one report for each mesh type. A configuration example is shown in Code 60.

```

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 60: 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` does not exist, any of the cartesian parameters are required, such as *cartesian/nx* or *cartesian/xmin*.

6.4.14.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Cartesian Distribution:** Kerma distribution in eV/g per history for the cartesian mesh. The mesh dimensions are, in order X (cm), Y (cm) and Z (cm). This result is only generated if the cartesian mesh is defined. Otherwise an empty object is returned.
- **Cylindrical Distribution:** Kerma distribution in eV/g per history for the cylindrical mesh. The mesh dimensions are, in order radius (cm), azimuthal angle (ϕ) in degrees and Z in cm. This result is only generated if the cylindrical mesh is defined. Otherwise an empty object is returned.
- **Spherical:** Kerma distribution in eV/g per history for the spherical mesh. The mesh dimensions are, in order, radius (cm), polar angle (θ) and azimuthal angle (ϕ), both in degrees. This result is only generated if the spherical mesh is defined. Otherwise an empty object is returned.

6.4.15 CT tally

This tally estimates the product of the linear attenuation coefficient (μ) and the path length (x) through the geometry system ($\mu \cdot x$) in a virtual CT-like scanner. The value is derived from the Beer-Lambert law: $\mu \cdot x = \ln(N_0/N)$, where N_0 is the fluence at the source and N is the detected fluence.

The detector geometry is simplified as an arc, with angular positions for each bin reported in degrees. This detector rotates around the Z-axis in a coplanar CT-like movement.

The tally collects the fluence of both particles generated at the source and aimed at the detectors, and particles arriving at the virtual detector. These data are reported along with the sinogram generated from both measures. Additionally, the energy deposited in each material for each projection is also reported.

This tally is intended to be used with the CT source (Section 6.2.6.2). The configuration parameters are described below and schematized in Figure 8:

- **type “CT_SINOGRAM”:** Type name of the tally.
- **emin:** Photon minimum energy to be collect (eV).
- **emax:** Photon maximum energy to be collect (eV).
- **npixels:** Number of pixels of the virtual detector.

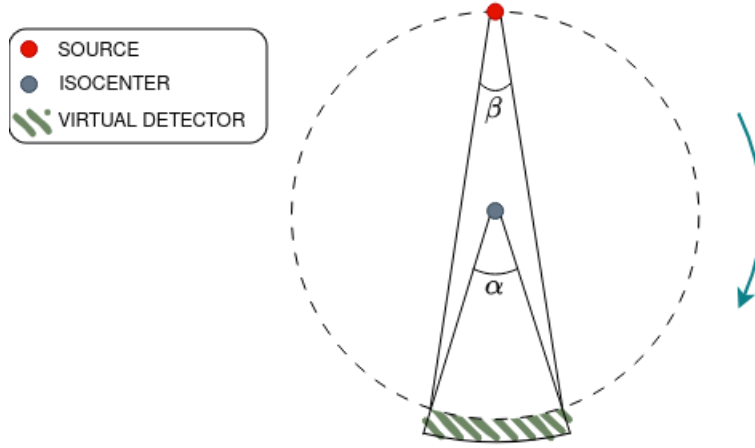


Figure 8: Schematic representation for the CT tally parameters

- **pixel-depth**: Depth size of the pixel along the radius, in cm.
- **r-inner**: Radius of the virtual detector. Corresponds to the isocenter-detector distance (cm).
- **xOrigin, yOrigin, zOrigin**: x, y, and z coordinates of the CT origin. Corresponds to the position of the CT isocenter (cm).
- **phi-ini**: Initial angular position of the CT detector, in degrees. Must be set in front of the CT source with an angular increment of 180 degrees.
- **phi-end**: Final angular position of the CT detector, in degrees.
- **nProjections**: Number of CT projections. Usually the same as the CT source.
- **aperture**: Angle subtended by the detector with respect to the isocenter, in degrees.

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

- **particle**: particle type that will be collect by the detector.

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

```

1 tallies/CTsinogram/type "CT.SINOGRAM" (mandatory)
2 tallies/CTsinogram/emin 0.0 (mandatory)
3 tallies/CTsinogram/emax 1.0e5 (mandatory)
4 tallies/CTsinogram/npixels 700 (mandatory)
5 tallies/CTsinogram/pixel-depth 4 (mandatory)
6 tallies/CTsinogram/r-inner 52.0 (mandatory)
7 tallies/CTsinogram/xOrigin 0.0 (mandatory)
8 tallies/CTsinogram/yOrigin 0.0 (mandatory)
9 tallies/CTsinogram/zOrigin 0.0 (mandatory)
10 tallies/CTsinogram/phi-ini 180 (mandatory)
11 tallies/CTsinogram/phi-end 540 (mandatory)
12 tallies/CTsinogram/angularStep 0.5 (mandatory)
13 tallies/CTsinogram/aperture 44.0 (mandatory)
14 tallies/CTsinogram/particle "gamma" (mandatory)
15 tallies/CTsinogram/tmin 0.0 (mandatory)
16 tallies/CTsinogram/timeInterval 120.0 (mandatory)
17 tallies/CTsinogram/scattered true (mandatory)

```

Code 61: CT tally configuration

6.4.15.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Sinogram**: Generated 2D sinogram. The reported magnitude is the product $\mu \cdot ds$ which units have no dimensions. The sinogram dimensions are, in order, projection number and pixel index.

6.4.16 DICOM kerma tally (Dose Volume Histogram)

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

To obtain the absorbed dose in Gy for each volume, this tally calculates the kerma, using the kerma track length estimator described in the section 6.4.14. The configuration tally parameters are described following:

- **type "DICOM_KERMA_TRACK_LENGTH"**: Type name of the tally.
- **kerma/emin**: Photon minimum energy to be considered (eV).
- **kerma/emax**: Photon maximum energy to be considered (eV).
- **kerma/dataFiles**: One data file with the μ_{en}/ρ (cm^2/g) coefficients per tallied material is required. This parameter is detailed in the Kerma track length estimator tally, section 6.4.14.

- **prescribedDose:** Treatment prescribed dose, in Gy. If this parameter has not been specified, it is set to 1 Gy.
- **MCkerma2Dose:** Conversion factor from kerma MC simulation units (eV/g per history) to absolute dose units, which must be set in Gy/(eV/g per history). The *MCkerma2Dose* factor is calculated using the equation 12.

$$MCkerma2Dose = \frac{1}{100} \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \quad (12)$$

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

$$S_{k,MC} = K_{MC}(d)d^2 \left[\frac{eV \cdot cm^2}{g \cdot history} \right] \quad (13)$$

where K_{MC} is the kerma in air in a thin air cell, such as a ring with 0.1cm width and 0.1cm depth, located on the seed transversal axis at a reference distance d , which is usually set to $d = 10$ cm or $d = 100$ cm. Also, the source must be located in a vacuum media. This configuration example is shown in the figure 9.

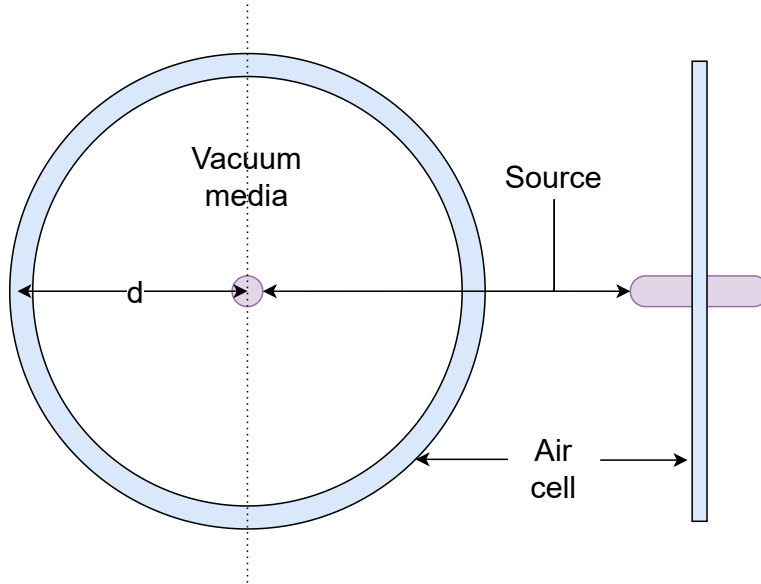


Figure 9: Geometry schema to calculate the $S_{k,MC}$ factor using a ring air cell situated at a reference distance d . The left image shows the central Z plane. The right image shows the central X or Y plane.

Thus, the resulting units for the S_k quotient are,

$$\begin{aligned} & \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[\frac{g \cdot history}{eV \cdot cm^2} \right] \left[\frac{cm^2 \cdot cGy}{h} \right] [h] \\ & \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[\frac{cGy \cdot history}{(eV/g)} \right] \end{aligned} \quad (14)$$

Converting to Gy, we obtain the units of the equation 12.

$$MCkerma2Dose = \frac{1}{100} \frac{S_{k,PLAN}}{S_{k,MC}} \cdot t \left[\frac{Gy \cdot history}{(eV/g)} \right] \quad (15)$$

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

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

```

1 tallies /DICOMkerma/type "DICOMKERMA_TRACKLENGTH" (mandatory)
2
3
4 tallies /DICOMkerma/kerma/emin 2.5e3 (mandatory)
5 tallies /DICOMkerma/kerma/emax 1.39950E+06 (mandatory)
6 tallies /DICOMkerma/kerma/dataFiles/1 "prostate-mutren.dat" (mandatory)
7 tallies /DICOMkerma/kerma/dataFiles/2 "uretra-mutren.dat" (mandatory)
8 tallies /DICOMkerma/kerma/dataFiles/3 "rectum-mutren.dat" (mandatory)
9 tallies /DICOMkerma/kerma/dataFiles/4 "bladder-mutren.dat" (mandatory)
10 tallies /DICOMkerma/kerma/dataFiles/5 "body-mutren.dat" (mandatory)
11 tallies /DICOMkerma/kerma/dataFiles/6 "bone-mutren.dat" (mandatory)
12
13 tallies /DICOMkerma/prescribedDose 15 (optional)
14 tallies /DICOMkerma/MCkerma2Dose 0.06831 (optional)
15 tallies /DICOMkerma/DVH-maxDose 40 (optional)
16 tallies /DICOMkerma/DVH-bins 1000 (optional)

```

Code 62: DICOM kerma tally configuration

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

6.4.16.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Kerma:** Kerma distribution in eV/g per history calculated within the DICOM voxelized mesh. The mesh dimensions are, in order X (cm), Y (cm) and Z (cm).
- **DVH:** The Dose-Volume Histogram (DVH) reports the volume percentage (for each configured DICOM contour) that receives at least a given absorbed dose (in Gy). The result is a 2D matrix with dimensions of absorbed dose and contour index, where each value represents the volume percentage for that dose and contour.

6.4.17 Tracks

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

- **type:** Must be set to **TRACK**.

- **nhists:** Specify the number of histories to be tracked.

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



Figure 10: Example of electron tracks in a water medium. The track color shows changes in the electron energy.

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

6.4.17.1 Output format

This tally does not provide any results when requested through pyPenred or the C++ simulation class. The results must be read from the generated files.

6.4.18 Primary and Scatter Separated (PSS)

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

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

- Primary photons (γ_1) start with $bkp = 0$. Any photon interaction (e.g., Compton) occurring outside the source generates $bkp+ = 1$ (e.g., $bkp_{\gamma_1} = 1$)
- Any secondary particle (e.g., Compton electrons), e_1 , inherits the generation of the incident photon (e.g., $bkp_{e_1} = 0$).
- Any photon produced by these secondary particles (e.g., bremsstrahlung photon), γ_2 , needs $bkp+ = 1$ (e.g., $bkp_{\gamma_2} = 1$).
- The source materials (e.g., active core and encapsulation) don't play any role in the *bkp* of primary photons. However, once the primary particle is scattered once in the phantom (e.g., water), the source starts acting like any other material, adding +1 to the scattered photons' *bkp*.

The configuration of this tally contains the following parameters,

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

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

and the non compatible types are,

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

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

In addition to the **PSS** parameters, the specific parameters of the selected tally must be specified to configure them, as is shown in the Code 63. In this example, a cylindrical dose tally is used and materials 1 and 2 are considered as source materials.

```

1 tallies/cylDoseDistribPSS/type "PSS"
2 tallies/cylDoseDistribPSS/PSS-source-mats [1,2]
3 tallies/cylDoseDistribPSS/PSS-tally "CYLINDRICAL_DOSE_DISTRIB"
4 tallies/cylDoseDistribPSS/print-xyz true
5 tallies/cylDoseDistribPSS/rmin 0.0
6 tallies/cylDoseDistribPSS/rmax 0.01
7 tallies/cylDoseDistribPSS/nbinsr 50
8 tallies/cylDoseDistribPSS/zmin 0.0
9 tallies/cylDoseDistribPSS/zmax 0.005
10 tallies/cylDoseDistribPSS/nbinsz 100

```

Code 63: PSS tally configuration example combined with a cylindrical dose distribution

6.4.18.1 Output format

This tally does not provide any results when requested through pyPenred or the C++ simulation class. The results must be read from the generated files.

6.4.19 Detection Spatial Distribution

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

- **type**: Must be set to “DETECTION_SPATIAL_DISTRIB”
- **spatial/nx**: Specifies the number of spatial bins to use along the X axis. This value is optional and defaults to 1.
- **spatial/xmin**: Sets the origin of the spatial grid along the X axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/xmax**: Sets the limit of the spatial grid along the X axis, in cm. This is required only if *spatial/nx* is specified.
- **spatial/ny**: Specifies the number of spatial bins to use along the Y axis. This value is optional and defaults to 1.
- **spatial/ymin**: Sets the origin of the spatial grid along the Y axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/ymax**: Sets the limit of the spatial grid along the Y axis, in cm. This is required only if *spatial/ny* is specified.
- **spatial/nz**: Specifies the number of spatial bins to use along the Z axis. This value is optional and defaults to 1.
- **spatial/zmin**: Sets the origin of the spatial grid along the Z axis, in cm. This is required only if *spatial/nz* is specified.
- **spatial/zmax**: Sets the limit of the spatial grid along the Z axis, in cm. This is required only if *spatial/nz* is specified.
- **detector**: Specifies the detector index where the data will be collected. The detector must be previously defined in the geometry configuration.
- **energy/nbins**: Specifies the number of energy bins to register the spectrum. This value is optional and defaults to 1.
- **energy/emin**: Sets the lower limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **energy/emax**: Sets the upper limit of the energy spectrum, in eV. This is required only if *energy/nbins* is specified.
- **particle**: Specifies the particle to be registered by name (text identifier).
- **printBins**: This optional boolean value enables/disables printing the spatial and energy bin indexes in the results file. The default is disabled.

- **printCord:** This optional boolean value enables/disables printing the bin coordinates, both spatial and energetic, in the results file. The default is enabled.

A configuration sample is shown in the Code 64, corresponding to the example *quadrics/9-measure-source/fromSource*.

```

1 tallies/SpatialDetector1/type "DETECTION_SPATIAL_DISTRIB"
2 tallies/SpatialDetector1/spatial/nx 100
3 tallies/SpatialDetector1/spatial/xmin -8.0
4 tallies/SpatialDetector1/spatial/xmax 8.0
5 tallies/SpatialDetector1/spatial/ny 50
6 tallies/SpatialDetector1/spatial/ymin -4.0
7 tallies/SpatialDetector1/spatial/ymax 4.0
8 tallies/SpatialDetector1/detector 1
9 tallies/SpatialDetector1/particle "gamma"
10 tallies/SpatialDetector1/energy/nbins 100
11 tallies/SpatialDetector1/energy/emin 10e3
12 tallies/SpatialDetector1/energy/emax 1.5e5

```

Code 64: Detection spatial distribution tally configuration

6.4.19.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Distribution:** The energy spectrum detected in each voxel of the 3D grid is returned in units of 1/history. The results dimensions are, in order, energy (eV), X (cm), Y (cm) and Z (cm), resulting in a 4D matrix.

6.4.20 Emerging Spherical Distribution

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

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

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

By default, all particles are disabled.

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

A configuration sample is shown in the Code 65.

```

1 tallies/angularEmerging/type "EMERGING_SPHERICAL_DISTRIB"
2 tallies/angularEmerging/energy/min 5.0e3
3 tallies/angularEmerging/energy/max 50e3
4 tallies/angularEmerging/energy/nbins 90
5 tallies/angularEmerging/theta/nbins 720
6
7 tallies/angularEmerging/spatial/nx 1
8 tallies/angularEmerging/spatial/xmin -10.0
9 tallies/angularEmerging/spatial/xmax 10.0
10 tallies/angularEmerging/spatial/ny 1

```

```

11 tallies/angularEmerging/spatial/ymin -10.0
12 tallies/angularEmerging/spatial/ymax 10.0
13 tallies/angularEmerging/spatial/nz 200
14 tallies/angularEmerging/spatial/zmin 0.0
15 tallies/angularEmerging/spatial/zmax 100.0
16
17 tallies/angularEmerging/particle/gamma true

```

Code 65: Emerging spherical distribution tally configuration

6.4.20.1 Output format

This tally provides the following results, in order, when requested through pyPenred or the C++ simulation class:

- **Emerging:** The energy spectrum of emerging particles is returned in units of 1/history. This spectrum is recorded for each spherical angular bin, using as dimensions, in order, the energy, in eV, and the polar and azimuthal angles, both in radians. Therefore, a 3D matrix is returned.
- **Last Interaction:** A regular 3D grid scoring the last interaction positions of emerging particles is returned also. Their units are 1/history, and the dimensions for the 3D matrix are, in order, X , Y and Z , all in cm.

6.5 Variance Reduction

In some cases, simulations may take an excessive amount of computational time depending on the statistical nature of the quantity being measured. To artificially increase the probability of desired events occurring, variance reduction techniques (VR) are often applied. To ensure the correctness when this techniques are used, the statistical weight of affected particles is modified depending on the technique.

PenRed implements the very same variance reduction (VR) techniques as the original PENELOPE FORTRAN code along with several additional methods. For generic simulations, these include interaction forcing (*IF*), x-ray splitting, and bremsstrahlung splitting. For phase space file based simulations, penRed also adds particle splitting and the Russian Roulette technique. Notice that variance reduction for phase space files is configured via the corresponding particle source parameters, as outlined in section 6.2. Additionally, penRed implements generic splitting and Russian Roulette as VR modules and provides the capability to add custom VR techniques.

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

In this section the available VR modules, both context specific and generic, and their configuration parameters will be described.

6.5.1 Context specific

To configure context specific VR techniques, the involved parameters are specified in the configuration file using the prefix *VR*, as shown in Code 66. In this code, the field *vr-technique* defines the specific VR technique to apply, such as interaction forcing (*IForcing*) or bremsstrahlung splitting (*bremss*). The *name* field serves as a user-defined identifier for this VR technique. Finally, the fields *parameter/path* and *value* vary based on the selected VR type, with detailed descriptions provided in the following sections for each technique.

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

Code 66: Variance reduction configuration pattern

6.5.1.1 Interaction forcing (IF) configuration

This technique consists of modifying the probability (cross section) of a specific interaction type. Particles interacting because this probability modifier will reduce their statistical weight, along with secondary produced particles. For example, it is used in the example *4-x-ray*, located at `examples/quadrics/4-x-ray`, to increase the production of gamma particles in a x-ray tube, forcing the bremsstrahlung and inner shell interactions for electrons. The corresponding configuration is shown in Code 67. The parameters are described following:

- **particle:** Specifies the particle name (Table 3) to which the technique will be applied.
- **interaction:** Sets the numerical identifier for the interaction type, as explained in section 5.4.
- **factor:** Defines the amplification factor for interaction forcing. A negative value indicates that a particle with the maximum source energy is expected to interact an average of $+factor$ times as it decelerates to rest (for electrons and positrons) or across a mean free path (for photons), similar to the handling in the PENELOPE main program.
- **min-weight:** Sets the minimum particle weight required to apply the IF technique; particles below this weight threshold will not be forced.
- **max-weight:** Sets the maximum particle weight for applying the IF technique; particles above this weight threshold will not be forced.
- **bodies/body-alias:** Defines the bodies in which interaction forcing will be enabled or disabled. To specify a body, add its alias under the *bodies* section and set its value to *true* to *enable* or *false* to *disable* IF in that body. By default, IF is *false* for all bodies. Multiple body aliases can be specified in this section to control IF for various bodies, as shown in Code 67, where bodies with aliases *1*, *2*, and *3* are enabled.
- **materials/material-index:** Similar to the *bodies* section, the *materials* section allows users to enable or disable IF for specific materials. Here, material indices are used instead of body aliases to identify the materials.

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

Code 67: Interaction forcing configuration for bodies

To apply interaction forcing to materials, replace *bodies* with *materials* and specify the desired material index as the *value* parameter. For instance, to apply the previously described IF technique to material with index 2, the configuration would be as shown in Code 68.

```

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

```

Code 68: Interaction forcing configuration for materials

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

6.5.1.2 Bremsstrahlung splitting configuration

This technique clones the photons produced by bremsstrahlung, reducing the statistical weight of cloned particles by a factor $1/N$, where N is the number of cloned particles. Additionally, the bodies (see Code 69) or materials (see Code 70) where the splitting is applied must be specified. The parameters are described following:

- **splitting**: Sets the *splitting-factor*, which determines the number of times bremsstrahlung photons will be cloned.
- **bodies/body-alias**: Defines the bodies where splitting will be enabled or disabled. To specify a body, add its alias to the *bodies* section and set its value to *true* to *enable* or *false* to *disable* splitting in that body. By default, splitting is set to *false* for all bodies. Multiple body aliases can be specified in this section to control splitting in various bodies. For example, Code 69 shows an example where bodies with aliases *1*, *2*, and *3* are enabled.
- **materials/material-index**: Similar to the *bodies* section, the *materials* section allows the user to enable or disable splitting for specific materials. In this case, the material index is used to specify the material, rather than the body alias, as shown in code block 70.

```

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

```

Code 69: Bremsstrahlung splitting configuration for bodies and materials

```

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

```

Code 70: Bremsstrahlung splitting configuration for bodies and materials

6.5.2 Generic VR techniques

Generic VR techniques can be applied to all particle types. Each technique follows the configuration pattern shown in Code 71.

```

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

```

Code 71: Generic VR technique parameter configuration pattern

where ‘vr-name’ specifies the name of the VR instance for identification purposes. The available generic VR techniques, with their parameters, are described following.

6.5.2.1 Splitting

The splitting VR technique works by cloning a particle and reducing its weight by a factor equal to the number of clones created. For example, if a particle with weight $WGHT$ is split into 10 clones (including the original particle), each resulting particle will have a weight of $WGHT/10$. All relevant parameters are listed in Code 72 and are detailed below.

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

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

Code 72: Splitting configuration for bodies and materials

A sample configuration is provided in the *8-fake-chamber-vr* example, as it is shown in Code 73

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

Code 73: Splitting configuration for bodies and materials

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

6.5.2.2 Russian roulette

The Russian roulette technique selectively eliminates particles based on a probability defined by the **prob** parameter in the configuration. If a particle survives, its weight is adjusted

by multiplying it by a factor of $1/\text{prob}$. The parameters for the Russian roulette technique are detailed below and summarized in Code 74. An example configuration can be found in the *8-fake-chamber-vr* example.

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

```

1 VR/generic/RR/type "RUSSIAN_ROULETTE"
2 VR/generic/RR/minWght 0.01
3 VR/generic/RR/maxWght 19.9
4 VR/generic/RR/materials/activeVolume/mat-index 1
5 VR/generic/RR/materials/activeVolume/prob 0.05

```

Code 74: Russian roulette configuration for bodies and materials

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

6.5.3 Specific VR techniques

A specific VR technique is applied to a particular particle type. Each technique follows the configuration pattern shown in code 75.

```

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

```

Code 75: Specific VR technique parameter configuration pattern

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

```

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

```

Code 76: Photons VR technique parameter configuration pattern

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

6.5.3.1 X-Ray splitting

X-ray splitting follows a configuration pattern similar to that used for bremsstrahlung. An example with all available parameters is shown in Code 77 and described below:

- **type**: Specifies the VR technique type, which must be set to *"XRAY_SPLITTING"*.

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

```

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

```

Code 77: X-ray splitting configuration for bodies and materials

A part of the configuration file corresponding to the *1-disc-vr* example, is shown in Code 78. This one enables x-ray splitting for the body with alias *1* cloning each x-ray produced twice.

```

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

```

Code 78: Example of X-ray splitting configuration for bodies

6.6 Simulation parameters

Simulation parameters does not define the physical properties nor the geometry system for our simulation. Instead, these parameters define how the simulation behaves. For example, specifying the number of threads to use, the maximum simulation time, etc.

This section describes the available parameters such parameters, allowing the user to control the behaviour and execution of the simulation. All of them follows the pattern shown in Code 79.

```

1 simulation/parameter/path value

```

Code 79: Simulation parameters configuration pattern

where **simulation** is a constant prefix.

6.6.1 Dump

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

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

To configure this feature, the following parameters are available:

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

An example of dump configuration can be found in the code 80, where the simulation will resume the stored state at dump file *dump.dat* and store the new generated dumps to *dump2.dat*. New dumps will be generated every 3600 s and the final dump is disabled.

```

1 simulation/dump-interval 3600
2 simulation/dump2read "dump.dat"
3 simulation/dump2write "dump2.dat"
4 simulation/finalDump false
5 simulation/dump2ascii false
6 simulation/ascii-results true

```

Code 80: Dump configuration parameters

6.6.1.1 Adding dumps

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

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

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

6.6.2 Threading

The following parameters control the multithreading capabilities of the simulation:

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

```
std::thread::hardware_concurrency()
```

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

- **thread-affinity:** This parameter expects a boolean value to enable or disable CPU affinity. This feature is only available in Unix environments and if threads are implemented via the pthreads package. Is disabled by default.
- **simulation/rank/N/threads:** This parameter specifies the number of threads to be used in the MPI rank number N , where N is the desired rank. This parameter overrides the *threads* parameter in that rank. An example is shown in the code 82.

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

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

Code 81: Number of threads specification

On MPI executions, the number of MPI processes is not specified in the configuration file. Instead, it is specified via the *mpirun* parameters, as regular MPI executions (see section 2). So, when both kinds of parallelism are combined, the number of threads spawned in each MPI rank is calculated according to the parameters specified in the configuration file. To maximize the efficiency in such executions, create a single MPI process for each node in a distributed memory infrastructure, and use threads instead of more MPI processes in the same node. 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.

For example, the configuration in the code 82 specify two threads for each MPI rank except for the ranks 1 and 2, which will use 4 and 8 threads respectively.

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

Code 82: Number of threads specification

6.6.3 Seeds

The user can specify the initial random generator seeds using the parameters *seed1* and *seed2*. Alternatively, a seed pair provided by the function *rand0* [13] can be selected via the *seedPair* parameter. Note that in multi-threading and/or MPI simulations, only the *seedPair* parameter can be used to set the initial seeds. This restriction ensures that each random number chain remains statistically independent.

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

Code 83: Initial seeds

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

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

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

6.6.4 Simulation time

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

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

For example, the Code 84 shows a configuration line specifying a maximum simulation time of 5 minutes.

```
1 simulation/max-time 300
```

Code 84: Maximum simulation time

6.6.5 Logs

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

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

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

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

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

6.6.6 Image output

PenRed allows to export some of the simulation outputs as images. Which outputs can be exported as images depends on each component implementation, for example, many tallies uses this capability. To enable the image exporter, the field *images* must be included in the *simulation* section followed by the format to enable/disable, as is shown in the code 85. Actually, the two available image formats are *MHD* [14], which is supported by many

viewers and can be read with the ITK library, and *GNU*, which consists of a format ready to be plotted as a matrix with the Gnuplot [15] tool. Each format can be enabled or disabled independently of the others. For instance, the code 85 shows how to enable both formats

```
1 simulation/images/mhd true
2 simulation/images/gnu true
```

Code 85: Image export

6.7 Load balance parameters

Load balance system is intended to distribute the workload among all available threads and MPI ranks, if enabled, on runtime. This online distribution maximize the simulation efficiency on distributed heterogeneous systems, where each node have different hardware and simulation speeds.

This optional feature can be enabled during the cmake configuration with the option *WITH_LB*. The only mandatory parameter to configure the load balance system is *balance-interval* (Code 86). This one specify the minimum balance time interval in seconds. Once a balance interval finish, the number of histories to do in each thread and MPI rank is recalculated according to their relative simulation speed.

6.7.1 Balance server

In addition to regular load balance, penRed simulations can be balanced using a centralised server, providing a method to balance simulations across the internet. To start a balance server, the penRed package a utility, which code is located at `src/utilities/LB/LBserver.cpp`.

In addition, in the simulation configuration, the user must specify the balance server hostname or IP as a 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 86: Load balance parameters

Note: Simulations balanced through a external balance server does not sum tally results automatically, as is done for regular load balance executions. Instead, the user should produce a final dump and add them (Section 80).

7 pyPenred

The *pyPenred*⁷ module provides Python bindings for penRed, enabling direct integration of radiation transport simulations into Python workflows. These bindings support scripted simulation control, allowing parameter optimization executions, and embedding within larger applications like the Blender plugin (Section 9.1). The implementation uses pybind11 for efficient C++/Python interoperability.

The API documentation⁸ includes some examples and can be found online or be built within the package. Read the corresponding readme at `src/bindings/python/pyPenred` for more information.

7.1 Installation

The simplest installation method is via pip:

```
pip install pyPenred
```

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

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

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

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

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

7.2 Configuration Format

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

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

For example, consider the equivalent configurations in Code 87 (native format) and Code 88 (YAML representation).

⁷<https://pypi.org/project/pyPenred/>

⁸<https://penred.github.io/PenRed/pyPenred/>

```

1 # Source #
2 sources/generic/source1/nhist 1.0e9
3 sources/generic/source1/kpar "gamma"
4 sources/generic/source1/record-time true
5
6 # Directional sampling
7 sources/generic/source1/direction/type "CONE"
8 sources/generic/source1/direction/theta 0.0
9 sources/generic/source1/direction/phi 0.0
10 sources/generic/source1/direction/alpha 5.0
11
12 # Energy sampling
13 sources/generic/source1/energy/type "MONOENERGETIC"
14 sources/generic/source1/energy/energy 3.0e7
15
16 # Spatial sampling
17 sources/generic/source1/spatial/type "POINT"
18 sources/generic/source1/spatial/position/x 0.0
19 sources/generic/source1/spatial/position/y 0.0
20 sources/generic/source1/spatial/position/z -25.0
21
22 # GEOMETRY PARAMETERS #
23
24 geometry/type "PEN_QUADRIC"
25 geometry/input-file "plane.geo"
26 geometry/processed-geo-file "report.geo"
27 geometry/kdet/1 1
28
29 # MATERIALS #
30
31 materials/water/number 1
32 materials/water/eabs/electron 1.0e5
33 materials/water/eabs/positron 1.0e5
34 materials/water/eabs/gamma 1.0e4
35 materials/water/C1 0.05
36 materials/water/C2 0.05
37 materials/water/WCC 2.0e3
38 materials/water/WCR 2.0e3
39 materials/water/filename "water.mat"
40
41 # SIMULATION PARAMETERS #
42
43 simulation/threads 2
44 simulation/max-time 600

```

Code 87: Configuration with the usual penRed format for the example *2-plane*.

```

1 geometry:
2   input-file: "plane.geo"
3   kdet:
4     1: 1
5   processed-geo-file: "report.geo"
6   type: "PEN_QUADRIC"
7 materials:
8   water:
9     C1: 5.00000E-02
10    C2: 5.00000E-02
11    WCC: 2.00000E+03
12    WCR: 2.00000E+03
13    eabs:
14      electron: 1.00000E+05
15      gamma: 1.00000E+04
16      positron: 1.00000E+05

```



```

17     filename: "water.mat"
18     number: 1
19 simulation:
20     max-time: 600
21     threads: 2
22 sources:
23     generic:
24         source1:
25             direction:
26                 alpha: 5.00000E+00
27                 phi: 0.00000E+00
28                 theta: 0.00000E+00
29                 type: "CONE"
30             energy:
31                 energy: 3.00000E+07
32                 type: "MONOENERGETIC"
33             kpar: "gamma"
34             nhist: 1.00000E+09
35             record-time: true
36             spatial:
37                 position:
38                     x: 0.00000E+00
39                     y: 0.00000E+00
40                     z: -2.50000E+01
41                 type: "POINT"

```

Code 88: Configuration file in YAML format for the example textit2-plane.

The conversion utilities support:

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

As shown in Code 89, users can load an existing configuration, modify parameters through dictionary assignment, and export back to standard PenRed format while maintaining full compatibility with standalone simulations. Additionally, the dictionaries can be used directly to configure simulations, as will be shown in the next section.

```

1 import pyPenred
2
3 # Load PenRed internal or YAML file formats and store it in a dictionary
4 d = pyPenred.readConfigFile("config.in")
5
6 # Modify the dictionary...
7
8 # Export the modified dictionary back to PenRed format as string
9 configText = pyPenred.simulation.dict2SectionString(d)
10
11 # Write final configuration to a file
12 with open("finalConf.in", 'w') as f:
13     f.write(configText)

```

Code 89: Load configuration to dictionary.

7.3 Usage

Currently, the *pyPenred* wrapper does not implement all the features of the C++ *penRed* library. For example, it lacks support for MPI and the capability to sum dumps. However, all

core simulation components, such as tallies, geometries, and samplers, are fully functional. As a result, simulations configured for penRed can also be executed using *pyPenred*.

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

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

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

```
1 import pyPenred
2 pyPenred.runFromFile("config.in")
```

Code 90: How to run a simulation from pyPenred and a penRed configuration file.

In addition to the usual format used by penRed for configurations, this function accepts also *YAML* files. These *YAML* files must follow the structure shown at the Section 7.2.

The previous example is quite simplistic, as it does not offer much control to the user. However, the definition of the *runFromFile* function (Code 91) can serve as a foundation for building more complex workflows.

```
1 def runFromFile(configFile = "config.in",
2                 statusTime = 20,
3                 configLog = "config.log",
4                 simulationLog = "simulation.log"):
5
6     #Set logs
7     simulation.setConfigurationLog(str(configLog))
8     simulation.setSimulationLog(str(simulationLog))
9
10    #Create simulation object
11    simu = simulation.create()
12
13    #Try to get the configuration from yaml or penred internal format
14    try:
15        f = open(configFile)
16        d = yaml.load(f, Loader=yaml.SafeLoader)
17        confSetErr = simu.configure(d)
18    except:
19        confSetErr = simu.configFromFile(configFile)
20
21    if(confSetErr != 0):
22        print("Invalid configuration file format. See config log\n")
23        return 1
24
25    print("Configuration set\n")
26
27    #Start the simulation asynchronously
28    err = simu.simulate(True)
29    if(err != 0):
```

```

30     print("Error on simulation configuration. See config.log\n")
31     return 2
32
33     #Simulation started, check status every 30 seconds
34     print("Simulation started\n")
35     while simu.isSimulating():
36         try:
37             time.sleep(statusTime)
38         except:
39             time.sleep(120)
40         status = simu.stringifyStatus()
41         for e in status:
42             print(e)
43
44     print("Simulation finished")

```

Code 91: Function *runFromFile*.

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

```

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

```

Code 92: Function *runFromFile* setting logs.

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

```

1     #Create simulation object
2     simu = simulation.create()

```

Code 93: Function *runFromFile* creating simulation object.

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

```

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

```

Code 94: Function *runFromFile* parsing and storing configuration.

Once the configuration is parsed and saved in the simulation object, the simulation process can be initiated. The *pyPenred* wrapper allows for running the simulation in both blocking and asynchronous modes through the *simulate* method of the *simulation* object. This method takes one optional boolean parameter to enable (true) or disable (false) the asynchronous simulation mode. In Code 96, the simulation is executed in asynchronous mode to retrieve its status periodically.

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

```

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

```

Code 95: Function *runFromFile* starting simulation.

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

```

1 #Simulation started , check status every 30 seconds
2 print("Simulation started\n")
3 while simu.isSimulating():
4     try:
5         time.sleep(statusTime)
6     except:
7         time.sleep(120)
8         status = simu.stringifyStatus()
9         for e in status:
10             print(e)
11
12 print("Simulation finished")

```

Code 96: Function *runFromFile* creating simulation object.

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

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

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

7.4 API

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

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

8 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
```

8.1 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.

8.1.1 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.

8.1.2 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.

8.2 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.

8.3 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.

8.4 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.

8.5 5-accelerator

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

8.5.1 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.

8.5.2 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.

8.5.3 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

8.6 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.

8.7 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.

8.8 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.

8.8.1 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.

8.8.2 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.

9 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/
```

9.1 Blender Plugin

This utility consists of a Python plugin designed for use within the Blender⁹ software. The plugin utilizes Blender's UI and 3D design tools to streamline the creation of both Quadric (see Section 6.3.2) and Triangular Mesh Surface (see Section 6.3.3) geometries. Furthermore, the plugin includes functionality to configure a complete simulation directly within the Blender UI, enabling users to export both the geometry file and the configuration file. The plugin is located in the following directory:

```
src/utilities/Blender
```

The directory contains the latest version of the plugin, which is compatible with Blender LTS version 4.2.3¹⁰. Within the plugin directory, the folder corresponding to the specific Blender version includes the plugin source code, a zip file named *penred.zip* (ready for installation via Blender's add-on system), and the plugin documentation source. The documentation is also available online at <https://penred.github.io/PenRed/Blender/>. When constructing a geometry in Blender for export to PenRed, it is important to note that Blender's default units are already interpreted as centimeters. As a result, no additional unit transformations are required. Specific considerations for each geometry type will be discussed in the following sections. However, this manual does not provide an introduction to Blender usage, as it is extensively documented elsewhere, and numerous tutorials are available online.

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

⁹<https://www.blender.org/>

¹⁰<https://www.blender.org/download/releases/4-2/>

9.2 Mutren

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

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

Each parameter is described following,

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

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

9.3 iaeaPSF

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

9.4 geo2voxel

This utility converts any geometry type in a voxelized one. As program arguments, the `geo2voxel` requires a configuration file which format is the same as `pen_main`, i.e. the

format described in the section 3. This configuration file, must include all the configuration parameters to load a geometry of any type, following the descriptions provided in the section 6.3, and the parameters listed below,

- **voxelized/nx**: Number of voxels in the X axis.
- **voxelized/ny**: Number of voxels in the Y axis.
- **voxelized/nz**: Number of voxels in the Z axis.
- **voxelized/dx**: Voxel size in the X axis, in cm.
- **voxelized/dy**: Voxel size in the Y axis, in cm.
- **voxelized/dz**: Voxel size in the Z axis, in cm.
- **voxelized/ox**: Voxelized geometry origin in the X axis, in cm.
- **voxelized/oy**: Voxelized geometry origin in the Y axis, in cm.
- **voxelized/oz**: Voxelized geometry origin in the Z axis, in cm.
- **voxelized/granularity**: Granularity used to determine the material and density of each voxel.
- **materials/material-name/ID**: Identification number of the material. There the *material-name* must be specified by the user and it is used only to distinguish different materials in the configuration file, i.e. has no influence in the converted geometry. This parameter must be repeated for different materials.
- **materials/material-name/density**: Density value of each material of the geometry, in g/cm^3 . This parameter must be repeated for different materials. This density is assigned to the material number with the same *material-name*.

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

An example of configuration to generate a voxelized geometry from a quadric geometry file is shown in the code 97, 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
18
19 materials/water/ID 1
20 materials/water/density 0.99821

```

Code 97: Complete `geo2voxel` configuration example to convert a quadric geometry.

9.5 geo2mesh

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

- **geometry:** This section contains information to configure the geometry, and its contents vary depending on the geometry type. It is structured as the used for a simulation.
- **nx:** The number of voxels used in the X-axis.
- **ny:** The number of voxels used in the Y-axis.
- **nz:** The number of voxels used in the Z-axis.
- **dx:** The voxel size in the X-direction, measured in centimeters.
- **dy:** The voxel size in the Y-direction, measured in centimeters.
- **dz:** The voxel size in the Z-direction, measured in centimeters.
- **ox:** The X-coordinate of the mesh origin, measured in centimeters.
- **oy:** The Y-coordinate of the mesh origin, measured in centimeters.
- **oz:** The Z-coordinate of the mesh origin, measured in centimeters.
- **granularity:** The number of points created within each voxel along each axis to assign the material number. It is advisable to use lower values to minimize computational costs. Typically, a value of 2 or 3 is sufficient.
- **smooth/steps:** The number of smoothing steps to apply to the mesh. This parameter is optional and is set to 50 by default.
- **smooth/factor:** The strength of the smoothing filter. This is also optional and is set to 0.2 by default.

Notice that the mesh origin is located at the first corner in the direction of increasing coordinates of the first mesh voxel. To use the configuration file, run the program as follows:

```
./geo2mesh config-file
```

the program will generate two files: one containing vertex information named 'geo.obj' and another with material information named 'geoMats.mtl'. The code 98 shows an example of configuration file to convert the 'tube.geo' geometry of the quadric example to a mesh.

```
1 #####
2 #####
3 # GEOMETRY PARAMETERS #
4 #####
5
6 geometry/type "PEN_QUADRIC"
7 geometry/input-file "tube.geo"
8 geometry/processed-geo-file "report.geo"
9
10 #####
11 # GEO2MESH PARAMETERS #
12 #####
13
14 ox -10
15 oy -10
16 oz -10
17
18 dx 0.1
19 dy 0.1
20 dz 0.1
21
22 nx 200
23 ny 200
24 nz 200
25
26 granularity 3
27
28 smooth/steps 20
29 smooth/factor 0.2
```

Code 98: Complete `geo2mesh` configuration example to convert a quadric geometry.

9.6 range

The **range** utility prints the electron, positron and photon ranges for the specified material and energies. The corresponding code is located in the folder `penMats`. The program must be executed via the command line as follows,

```
./range material-file E1 E2 E3
```

where *range* is the executable name, *material-file* is the material file with the material information, the same as the used for simulations, and the following arguments ($E1, E2, E3$) are a list of energies to calculate the corresponding ranges. Notice that in the execution example we have used 3 energies. However, any number of energies can be used.

The interpretation of the returned range depends on the particle type:

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

9.7 PSF spectrum

The `psf_spectre` utility extracts the energy spectrum distribution of the particles registered in a PSF file, and the corresponding source code can be found in the `psf/psf_spectre.cpp` file. This utility is used via the command line as follows,

```
./psf_spectre PSFfilename emin emax nbins
```

where *psf_spectre* is the utility executable, *PSFfilename* the path to the PSF file to be processed, *emin* the minimum energy, in eV, of the extracted spectrum, *emax* the corresponding spectrum maximum energy and *nbins* the number of spectrum bins.

9.8 PSF to ASCII

The `psf2ascii` utility converts a binary PSF file into a ASCII file format. The source code can be found in `psf/psf2ascii.cpp` and must be used via the command line as follows,

```
./psf2ascii inputFile outputFile
```

where *psf2ascii* is the executable, *inputFile* the binary PSF file path and *outputFile* the filename of the generated ASCII file.

9.9 Registers

The `registers` folder contains a set of sources which compiled programs show all the available types of a specific component. The shown types correspond to the ones used in the configuration files to specify the component *type* parameter. All the available registers are summarised in the Table 10. This 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 10: Available register programs with the corresponding registered information.

9.10 CompositionsDB

The *compositionsDB* utility provide access to the composition databases (DB). If no arguments are provided, the names of all available databases are listed (Code 99):

```
1 ./compositionsDB
2 usage: ./compositionsDB [composition-DB] [material-name]
3 Registered data bases:
4 + ICRP_AF
5 + ICRP_AM
```

Code 99: Composition utility DB listing.

For now, the two available databases are the ICRP compositions for the adult female and male phantoms, *ICRP_AF* and *ICRP_AM*, respectively [16]. If a database is specified, all materials belonging to it are listed. (Code 100):

```

1 ./compositionsDB ICRP_AF
2 usage: ./compositionsDB [composition-DB] [material-name]
3 Registered compositions in database 'ICRP_AF':
4 + Adrenal_left
5 + Adrenal_right
6 + ET1(0-8)
7 + ET1(8-40)
8 + ET1(40-50)
9 + ET1(50-surface)
10 + ET2(-15-0)
11 + ET2(0-40)
12 + ET2(40-50)
13 + ET2(50-55)
14 + ET2(55-65)
15 + ET2(65-surface)
16 + Oral_mucosa_tongue
17 + Oral_mucosa_mouth_floor
18 + Oral_mucosa_lips_and_cheeks
19 + Trachea
20 + BB(-11-6)
21 + BB(-6-0)
22 + BB(0-10)
23 .
24 .
25 .

```

Code 100: Composition utility: Listing ICRP_AF materials.

Finally, if both the database and material name are provided, the density and composition of the specified material are shown. The composition is specified by pairs (atomic number, fraction by weight), as is shown in the Code 101:

```

1 ./compositionsDB ICRP_AF Humeri_upper_cortical
2 usage: ./compositionsDB [composition-DB] [material-name]
3
4 + Material 'Humeri_upper_cortical' ( 1.90400E+00 g/cm**3):
5 - 1 0.036000
6 - 6 0.159000
7 - 7 0.042000
8 - 8 0.448000
9 - 11 0.003000
10 - 12 0.002000
11 - 15 0.094000
12 - 16 0.003000
13 - 20 0.213000

```

Code 101: Composition utility DB: Listing material composition.

The user can specify multiple material names in a single execution, obtaining the corresponding composition for each one.

9.11 Material creation

Two utilities are available to create material files, in addition to the runtime option and pyPenred functions. These utilities are *createMat* and *createListedMat*.

The first utility, *createMat*, behaves like the FORTRAN material creation utility included in the PENELOPE package. It creates a generic material by specifying the composition, density, and other optional parameters. Once executed, follow the program instructions to create the material. As in the FORTRAN code, a set of predefined materials can be built using numerical codes, which are specified in Appendix A, in Tables 11 and 12.

The second utility, *createListedMat*, creates a pre-defined material from an existing database. To create one of the PENELOPE pre-defined materials (those included in Appendix A, in Tables 11 and 12), specify, in order, the material numeric identifier and the output filename:

```
./createListedMat mat-id output
```

To create a material from another database, those listed by the utility *compositionsDB* (Section 9.10), specify, in order, the database name, material name, and output filename:

```
./createListedMat database material output
```

9.12 internalData2YAML

This utility converts a file with the penRed internal data format to a YAML format file. In the process, each slash is converted to a new level in the YAML structure.

```
1 usage: ./internalData2YAML internalData-filename outputYAML
```

Code 102: internalData2YAML usage.

A Predefined materials

ID	Name	ID	Name	ID	Name
1	Hydrogen	34	Selenium	67	Holmium
2	Helium	35	Bromine	68	Erbium
3	Lithium	36	Krypton	69	Thulium
4	Beryllium	37	Rubidium	70	Ytterbium
5	Boron	38	Strontium	71	Lutetium
6	Amorphous carbon	39	Yttrium	72	Hafnium
7	Nitrogen	40	Zirconium	73	Tantalum
8	Oxygen	41	Niobium	74	Tungsten
9	Fluorine	42	Molybdenum	75	Rhenium
10	Neon	43	Technetium	76	Osmium
11	Sodium	44	Ruthenium	77	Iridium
12	Magnesium	45	Rhodium	78	Platinum
13	Aluminium	46	Palladium	79	Gold
14	Silicon	47	Silver	80	Mercury
15	Phosphorus	48	Cadmium	81	Thallium
16	Sulfur	49	Indium	82	Lead
17	Chlorine	50	Tin	83	Bismuth
18	Argon	51	Antimony	84	Polonium
19	Potassium	52	Tellurium	85	Astatine
20	Calcium	53	Iodine	86	Radon
21	Scandium	54	Xenon	87	Francium
22	Titanium	55	Cesium	88	Radium
23	Vanadium	56	Barium	89	Actinium
24	Chromium	57	Lanthanum	90	Thorium
25	Manganese	58	Cerium	91	Protactinium
26	Iron	59	Praseodymium	92	Uranium
27	Cobalt	60	Neodymium	93	Neptunium
28	Nickel	61	Promethium	94	Plutonium
29	Copper	62	Samarium	95	Americium
30	Zinc	63	Europium	96	Curium
31	Gallium	64	Gadolinium	97	Berkelium
32	Germanium	65	Terbium	98	Californium
33	Arsenic	66	Dysprosium	99	Einsteinium

Table 11: 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 12: 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] E. García-Toraño, V. Peyres, F. Salvat, Pennuc: Monte carlo simulation of the decay of radionuclides, *Computer Physics Communications* 245 (2019) 106849. doi:<https://doi.org/10.1016/j.cpc.2019.08.002>.
URL <https://www.sciencedirect.com/science/article/pii/S0010465519302267>
- [10] V. Giacometti, S. Guatelli, M. Bazalova-Carter, A. Rosenfeld, R. Schulte, Development of a high resolution voxelised head phantom for medical physics applications, *Physica Medica* 33 (2017) 182 – 188. doi:<https://doi.org/10.1016/j.ejmp.2017.01.007>.
URL <http://www.sciencedirect.com/science/article/pii/S1120179717300078>
- [11] J. F. Williamson, Monte carlo evaluation of kerma at a point for photon transport problems, *Medical Physics* 14 (4) (1987) 567–576. arXiv:<https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1118/1.596069>, doi:10.1118/1.596069.
URL <https://aapm.onlinelibrary.wiley.com/doi/abs/10.1118/1.596069>
- [12] N. Ravinder et al, Dosimetry of interstitial brachytherapy sources: recommendations of the AAPM radiation therapy committee task group no 43, *Med. Phys.* 22 (1995) 209–34. doi:10.1118/1.597458.
- [13] A. Badal, J. Sempau, A package of linux scripts for the parallelization of monte carlo simulations, *Computer Physics Communications* 175 (6) (2006) 440 – 450. doi:<https://doi.org/10.1016/j.cpc.2006.05.009>.
URL <http://www.sciencedirect.com/science/article/pii/S001046550600230X>
- [14] Mhd/raw image format, <https://itk.org/Wiki/ITK/MetaIO/Documentation>, accessed: 2022-07-15.
- [15] Gnuplot, <http://www.gnuplot.info/>, accessed: 2022-07-15.
- [16] M. Zankl, Adult male and female reference computational phantoms (icrp publication 110), *Japanese Journal of Health Physics* 45 (4) (2010) 357–369.