TPPTsim description

TPPTsim is a part of the TPPT software framework, developed at the LIP-Coimbra group in the frame of the TPPT collaboration. The framework consists of the TPPTsim (simulator), TPPTbuilder (event builder) and TPPTcoinc (coincidence sorter). The TPPTsim framework has two main purposes. The first one is to provide stimulation-based predictions for the spatial distributions of the activity of the positron emitting species and the dose induced by a proton beam. The second one is to conduct simulations providing data on the energy deposition in the scintillators of the TPPT ToF-PET scanner. These results can be later processed in TPPTbuilder to define the scintillation events and, after export of the event data to TPPTcoinc, to provide a list of the coincidence pairs resulting from annihilation gammas.

Table of contents

1. Installation guide

To compile the framework for the first time use the following steps:

- make a new directory and download the source from the Github repository https://github.com/andrmor/TPPTsim
- open terminal in this directory
- mkdir build
- cd build
- cmake ..
- make

Any changes made in the source files (e.g. modification of the simulation configuration) require the project to be recompiled through the use of “make”. The file name of the generated executable is “sim”.

As an alternative we recommended to use the Qt framework (https://www.qt.io/) to configure, compile and run TPPTsim.To open start working with the framework, open CMakeLists.txt file in Qt.

2. Overview

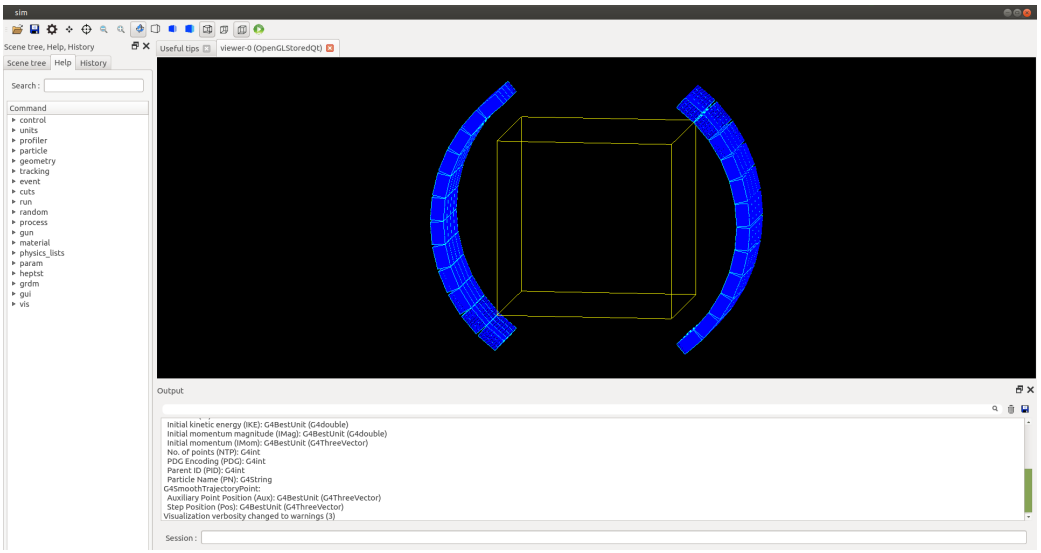
Each simulation in TPPTsim requires configuration of four aspects. They are designated as Phantom, Detector composition, Source and Simulation Mode:

- Phantom is an object (single or a composite one) defined in the geometry which represents the physical phantom
- Detector composition defines the composition of the scanner. The examples are the scintillators, the structural components of the scanner and a logical volume used for particle loggings during simulations.
- Source is the component responsible for generation of the primary particles.
- Simulation Mode defines the type of the simulation to be conducted.

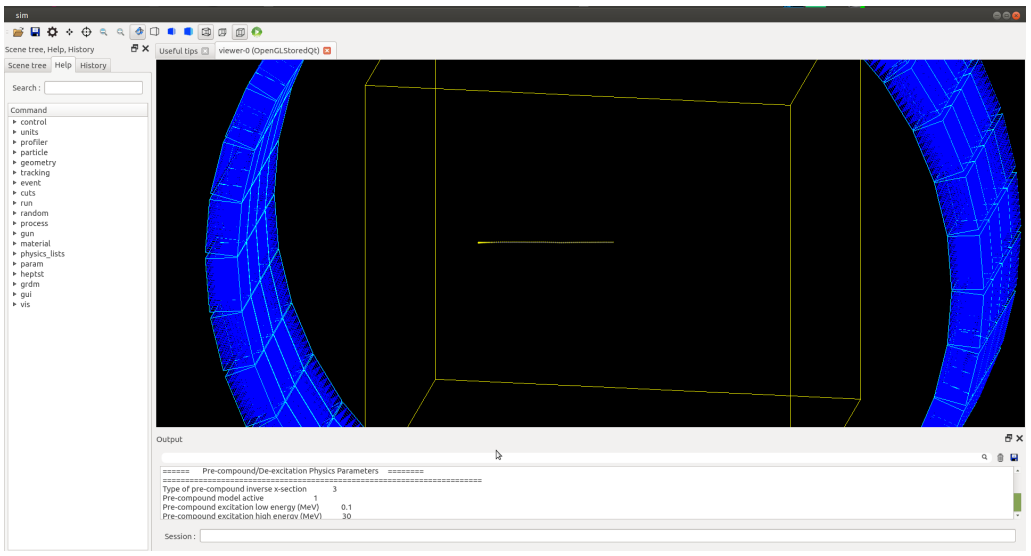
In order for these aspects to be configured, it is possible to directly provide the values for SM.Phantom (Phantom component), SM.DetectorComposition (Detector composition component), SM.SourceMode (Source component) and SM.SimMode (Simulation Mode component) located in the configuration section of the main.cc file. An example of a simplistic configuration, defining a box-shaped phantom of PMMA material, a detector consisting only of scintillators, a primary source generating 100 MeV protons isotropically from a point at (0,0,0) coordinates with the timestamp of 0 ns, and starting the Geant4 GUI as the simulation mode, is the following:

```
SM.Phantom = new PhantomBox(200.0*mm, 200.0*mm, 200.0*mm, EMaterial::PMMA);
SM.DetectorComposition.add(DetComp::Scintillator);
SM.SourceMode = new SourcePoint(new Proton(100.0*MeV), new ConstantTime(0), {0,0,0});
SM.SimMode = new ModeGui();
```

The result of running TPPTsim with this configuration will be opening of the following window:



When the green “play” button situated at the right side of the top toolbar is pressed, the result of a Geant4 simulation of a single proton will be shown:



Important notes:

- All arguments provided with this configuration approach use the default units of Geant4. To avoid doubts, it is strongly recommended to specify the intended units directly using the Geant4's "system of units": for example, in order to provide 1 mm one can use 1*mm or 0.1*cm or 0.001*m.
- To start any simulation, a phantom, source and simulation mode must be provided. The detector composition can be omitted if the detector is not required for the selected simulation mode.
- Note that later in this text we refer to running a simulation using the ModeGui to visualize the geometry as "Visualization Mode".

Besides the four main aspects, two more features should be considered (or the default values could be left unchanged). One is the simulation seed and the other is the working directory. In order to get different results when starting simulations with the same configuration, a different number has to be assigned to *SM.Seed*. In order to start a simulation and get the output files, a destination directory needs to be assigned to *SM.WorkingDirectory*.

Lastly, there are several additional options which can be configured or left unchanged. These options and their function are listed below:

- *SM.SimAcollinearity* = true/false → When set to true, the gammas of the annihilation pairs are generated taking into account acolinearity. When set to false, the default behavior of Geant4 is used (gammas are collinear, unless positron is not fully thermalized)
- *SM.KillNeutrinos* = true/false → When set to true, created neutrinos are disregarded (their tracking is skipped)
- *SM.UseStepLimiter* = true/false → When set to true, a limiter to the steps size considered by Geant4 in its simulations is enabled. The value of this limiter is assigned to SM.PhantomStepLimit
- Production cuts for secondary gammas, electrons and positrons, used by Geant4 simulation, can be specified separately for the phantom and the rest of the geometry.

The cuts give the minimum spatial range which the secondary particle should be able to travel in the current medium. If the energy is not sufficient, the particle is not generated, and it is assumed that the energy is dissipated locally. The default values recommended for TPPT simulations are:
    - *SM.CutPhantomGamma* = 10.0*mm
    - *SM.CutPhantomElectron* = 10.0*mm
    - *SM.CutPhantomPositron* = 0.1*mm
    - *SM.CutScintGamma* = 0.1*mm
    - *SM.CutScintElectron* = 0.1*mm
    - *SM.CtScintPositron* = 0.1*mm
- *SM.Verbose* = true/false → When set to true, Geant4 simulation is configured to give the maximum amount of logging information to the terminal. It activates the following Geant4 options:
    - /hits/verbose 2
    - /tracking/verbose 2
    - /control/saveHistory
- *SM.ShowEventNumber* = true/false → When set to true, the current event number of the running simulation is shown in the terminal. The interval of the shown events is assigned to *SM.EvNumberInterval*. The default value is 10000

Note that each run of the TPPTsim creates a configuration file BuilderConfig.json in the working directory, which lists all the parameter values (JSON format). Such files provide an alternative way to configure TPPTsim: the parameter values could be read from the file. To use this configuration mode, provide the file name in the command line: “sim -f name_of_config_file”. In this case the values given directly in the main.cc have no effect. It is also possible to specify the seed directly as the command line argument (“-s random_seed”). Both -f and -s flags can be present, in this case the seed specified in the command line will be used.

The following sections will describe the different options available in each of the four main aspects previously presented.

3. Phantom

The phantoms currently implemented in the framework are presented in the following subsections. The subsections contain a short description of the phantom (and its purpose if it has a specific one), lists the signature of the method used to add that phantom and explains its arguments.

An example of the method used to configure a phantom is:

*SM.Phantom = new PhantomCylinder(30.0*mm, 100.0*mm, EMaterial::PMMA)*

Using this line, a cylindrical phantom with 30 mm diameter and 100 mm height, made of PMMA, is constructed.

After configuring a phantom, we recommend checking the geometry using the Visualization Mode.

1.1. How to configure a simulation without phantom

As a phantom must be configured to start any simulation, in this case one has to use the “PhantomNone” formal phantom.

Signature: PhantomNone()

There are no arguments.

1.2. Box shaped phantom

This option is used to configure a cuboid (box shaped) phantom.

Two available signatures:

PhantomBox(double sizeX, double sizeY, double sizeZ, string G4_material_name)

PhantomBox(double sizeX, double sizeY, double sizeZ, EMaterial material)

Arguments:

- sizeX - full size along X axis
- sizeY - full size along Y axis
- sizeZ - full size along Z axis
- G4_material_name - name of a standard Geant4 material (e.g. “G4_WATER”)
- material - custom material (e.g. EMaterial::GelWater), see section 1.2.1

1.2.1 Material options

The list of the standard material of Geant4 can be found here:

https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/Appendix/materialNames.html

Currently implemented custom materials (to be used with the prefix EMaterial:: in the method signature, e.g. EMaterial::PMMA):

- PMMA (formula: C5O2H8; density: 1.18 g/cm3)
- HDPE (formula: C2H4; density: 0.95 g/cm3)

- Graphite (formula: C; density: 1.83 g/cm3)
- GelTissue (Weight fraction % per element: O(73.8), C(14.9), H(9.6), N(1.46); density: 1.13 g/cm3)
- GelWater (Weight fraction % per element: O(87.6), H(11.03), C(1.04), N(0.32); density: 1.01 g/cm3)

To add a new custom material please modify MaterialBuilder class (MaterialBuilder.hh and MaterialBuilder.cc)

1.3. Cylindrical phantom

This option is used to configure a cylindrical phantom with the axis aligned along the scanner axis.

Two available signatures:

PhantomCylinder(double diameter, double length, string G4_material_name)

PhantomCylinder(double diameter, double length, EMaterial material)

Arguments:

- diameter - cylinder's diameter
- length - cylinder's height (along the scanner axis)
- G4_material_name - name of a standard Geant4 material (e.g. "G4_WATER")
- material - custom material (e.g. EMaterial::GelWater), see section 1.2.1

1.4. Phantom based on a CT data from a DICOM file

This option is used to configure an anthropomorphic phantom which is generated based on the computed tomography (CT) attenuation data contained in a file using DICOM (Digital Imaging and COmmunications in Medicine) format.

The CT attenuation data are given in the Hounsfield units (HU) in the file. In order to create Geant4 voxels, the following procedure is used: (1) the HU values from each slice file are converted to density values using a calibration curve (provided by the user or a default one) and (2) a material is assigned to each voxel, according to the density value of the voxel.

The conversion from HU to density values is performed according to the information in the ASCII file CT2Density.dat. A file with default conversion data is generated automatically by the framework in the run directory, but its content might need to be modified if another CT scanner was used to record the data. The assignment of a material to each voxel is performed according to the information in the ASCII file Materials.dat. A default file is also generated automatically in the run directory. The formats of both files are given at the end of this subsection.

When a DICOM dataset is used for the first time, an ASCII file with the extension .g4dcm is produced for each DICOM file, representing a slice, and saved in the phantom directory defined by the user. For subsequent simulations, these ASCII files are used as the basis for creating the DICOM phantom. Each ASCII file includes a header as defined below:

3 - Number of materials

```
0 Material1    - A line for each material with its index and name
1 Material2    (the same as defined in the Materials.dat file)
2 Material3
nX nY 1        - Number of voxels in X, Y and Z
X1 X2          - Minimum and maximum extension in X (mm)
Y1 Y2          - Minimum and maximum extension in Y (mm)
Z1 Z2          - Minimum and maximum extension in Z (mm)
```

Then, nX*nY values corresponding to the material indices (one per voxel) are presented. Finally, another nX*nY values corresponding to the material densities (one per voxel) are also defined. Note that the number of voxels and the phantom dimensions are automatically defined according to the information provided in the DICOM files. Note also that these ASCII files can be generated by an external text editor and used in the simulation framework.

To use this phantom, the user must define a directory that contains the files corresponding to each phantom slice. All the phantom voxels are positioned inside an air-filled cylindrical container with a radius defined by the user. The size along the Z axis (along the scanner axis) is automatically defined according to the number and thickness of the phantom slices. The container position is also defined by the user.

The user should also specify a compression factor, which should be a power of 2 (1, 2, 4, 8, 16, …). If the compression factor is 1, this means that each original pixel in the DICOM files corresponds to one voxel in the Geant4 phantom. If the compression value is 2, then 2 pixels are merged in each direction, so that the number of voxels in the X and Y dimensions is reduced by a factor of 2 (consequently, the voxel size in X and Y is increased by a factor of 2). The HUs of the merged voxels are averaged to give the resulting value for the new voxel. Note that this compression does not affect the number of voxels in the Z direction (i.e. the number of slices remains unchanged). Similarly, if a compression factor of 4 is used, the number of voxels in the X and Y dimensions will be reduced by a factor of 4, and so on for higher compression values. For simulation purposes, a compression value of 1 is strongly recommended. If the user just wants to visualize the DICOM phantom, a higher compression value should be used (e.g. 8 or 16, depending on the number of slices of the phantom volume).

Signature: PhantomDICOM(string dataDir, string sliceBaseFileName, int sliceFrom, int sliceTo, int lateralCompression, double containerRadius, array posInWorld, bool makeCointainerVisible)

Arguments:

- dataDir - directory in which the DICOM files with the slice data are stored
- sliceBaseFileName - prefix name of the slices file (without the slice number and the suffix!)
- sliceFrom - first slice that is used to construct the phantom
- sliceTo - last slice that is used to construct the phantom
- lateralCompression - number (power of 2: 1, 2, 4, 8, 16, …) of voxels that will be merged into one in the X and Y dimensions.
- containerRadius - radius of the air-filled cylindrical phantom container

- posInWorld - coordinates of the phantom container in the world (use {0,0,0} for isocenter)
- rotInWorld - Euler angles of the phantom container in the world ({0,0,0} is default, do not forget to indicate the units: e.g. {180.0*deg,0,0} rotates the phantom 180 degrees around the axis)
- makeCointainerVisible - if set to true (default is false), the container is visible in Visualisation Mode

If the user wants to modify the process of converting the HU to Geant4 voxels (e.g. add new materials), the files CT2Density.dat and/or Materials.dat file have to be modified. Note that the presence of the CT2Density.dat file is first checked in the phantom directory. Only if it is not there, the default file CT2Density.dat from the run directory is used.

The format of the CT2Density.dat file:

In the first line, the number of entries is given. Then the pairs of the HU value and the corresponding density, separated by a space, are given, each in a new line. An example of a CT to density calibration file with 3 entries is shown below:

```
3
-1000  0.1
0      1.0
1000   1.8
```

The format of the Materials.dat file:

Each line consists of a string with a material name followed by a density value. An example of a file with 3 materials is shown below:

```
Material1   0.25
Material2   0.5
Material3   1.0
```

According to this material file example, a voxel with density lower than 0.25 is classified as Material1. Then, a voxel with a density between 0.25 and 0.5 is classified as Material2, a voxel with a density between 0.5 and 1.0 is classified as Material3, and so on for all the materials in the list. Each material appearing in the file Materials.dat should be defined with the same name in the PhantomDICOM class (see file DicomPhantom.cc in src folder). Note that the density units are not important, but should be consistent in both files.

1.5. Derenzo Phantom

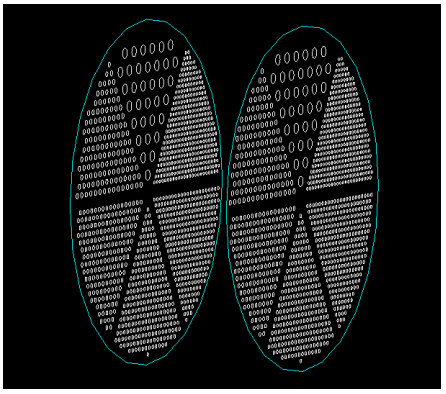
This option is used to generate a custom Derenzo-like phantom. It consists of a cylindrical container and an arbitrary number of sets of cylindrical capillaries each filling a sector of the cylinder. The capillaries are packed in the triangular pattern, and the distance between the axes of the neighboring ones is equal to the capillary diameter. The number of capillaries per sector is the maximum one which can fit into the sector according to the selected diameters

of the cylinder and the capillary, as well as the margins (empty space) configured by the user in the center and periphery of the cylinder.

The material used in the phantom is PMMA whereas the capillaries are filled with water (“G4_WATER” standard material of Geant4). When used in conjunction with the Material Limited Source (see section 3.3.5), and material is limited to G4_WATER, the primary particles (configurable) are generated uniformly over the entire volume of all the capillaries.

Example of Derenzo phantom:

*SM.Phantom = new PhantomDerenzo(200.0*mm, 100.0*mm, {1.8*mm, 2.0*mm, 2.2*mm, 2.5*mm, 3.0*mm, 6.0*mm}, 10.0*mm, 5.0*mm, 60.0)*



Signature: PhantomDerenzo(double diameter, double height, array capillaryDiameters, double radialOffset, double margin, double dPhi)

Arguments:

- diameter - cylinder’s diameter
- height - cylinder’s height
- capillaryDiameter - an array with the diameters of the capillaries for each section. The size of the array defines the number of sections present in the phantom (e.g. 6 in the example above)
- radialOffset - distance between of the container’s axis and the axis of the first capillary in each section
- margin - minimum distance between the container’s side and the axis of the utmost capillaries
- dPhi - rotation angle of the sections around the phantom’s axis

Note: When not specified in the arguments, phantom’s center is positioned at the isocenter (XYZ coordinates of zero).

2. Detector composition

This section describes the detector components (geometric bodies and the corresponding materials) which can be enabled during simulations. All the components of the TPPT scanner are listed below in the corresponding subsections. Note that at the end of this chapter a special component is described which is required for the simulation mode used to log in a text file the particles generated in the phantom which are about to enter the scintillators.

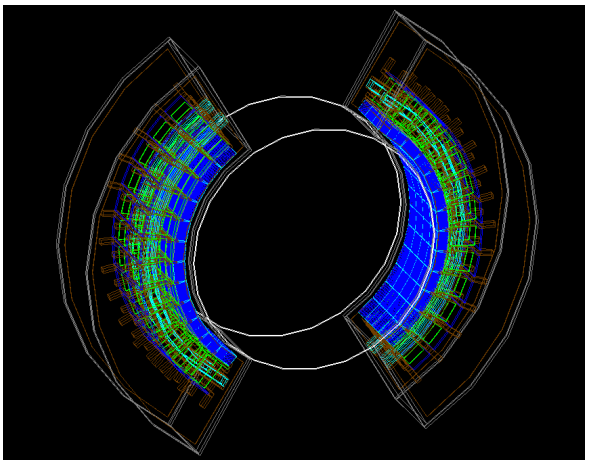
An example of the detector configuration which enables all the components:

SM.DetectorComposition.add(DetComp::Scintillators);

SM.DetectorComposition.add({DetComp::Base, DetComp::ClosedStructures, DetComp::SIPM, DetComp::PCB, DetComp::CopperStructure, DetComp::CoolingAssemblies});

SM.DetectorComposition.add(DetComp::ParticleLogger);

Running Visualization Mode will show the following:



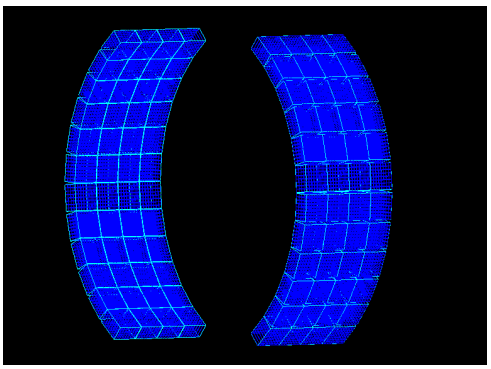
2.1. Scintillators

When this component is activated, the scintillators are added to the geometry. In total, there are 6144 scintillators, which are organized in two detector heads (crescents), each having 4 rows of 12 assemblies of 8 x 8 scintillators.The scintillator size is 3.005 x 3.005 x 15.0 mm^3. The material is $Lu_{49}Y_1Si_{25}O_{125}$:Ce with a density of 7.31 g/cm^3. The scintillators are separated by Teflon layers with 0.195 mm thickness. The same thickness Teflon layer is also present on the outside of the assemblies. Each scintillator has an individual index (copy number).

To activate this component, the following configuration line has to be added in the configuration section of main.cc:

SM.DetectorComposition.add(DetComp::Scintillators);

Using the Visualization Mode the image of the scintillators can be obtained:



After starting any simulation with enabled scintillators, an ASCII file with a look-up table (LUT) containing information on all the scintillators is automatically generated in the working directory with the name “LUT.txt”. The file lists information for all scintillators (index from 0 to 6143) in the following format:

FacePosX FacePosY FacePosZ NormX NormY NormZ HeadNumber Angle AssemblyNumber

where

- FacePosXYZs are the XYZ coordinates of the center of the inner surface (the one oriented towards the isocenter) of the scintillator
- NormXYZs are the XYZ components of the unit vector normal to the inner surface of the scintillator and oriented towards the isocenter
- HeadNumber is the index of the detector head (“crescent”): 0 or 1
- Angle is the rotation angle in degrees around the detector axis, which gives the position of the scintillator in respect to the one with index of 0
- AssemblyNumber is the index of an 8x8 scintillator assembly, containing this scintillator.

Each line is terminated with ‘\n’ (end of line) symbol. Space character is used as the delimiter.

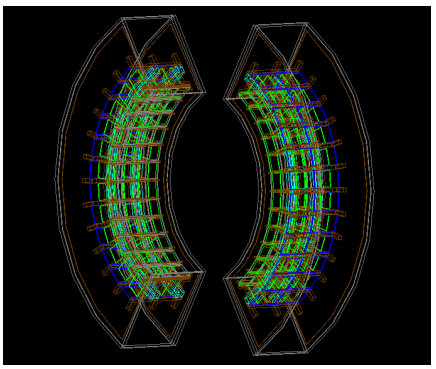
2.1.1. Support structures

There are several components which represent different elements of the TPPT scanner: the upper and lower base, housing, silicon photomultipliers, printed circuit boards, copper “ribs" and cooling assemblies. A detailed description of these components can be found following this link: https://estudogeral.uc.pt/handle/10316/106057.

These components can be add together in the framework by using the following line in the configuration section of main.cc:

SM.DetectorComposition.add({DetComp::Base, DetComp::ClosedStructures, DetComp::SIPM, DetComp::PCB, DetComp::CopperStructure, DetComp::CoolingAssemblies});

Using the Visualization Mode the following image will be obtained:



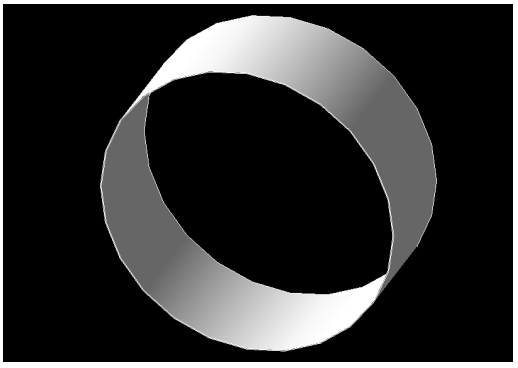
2.2. Particle recorder

The ParticleRecorder is a physical volume (thin-walled tube) made of air which should be added to the geometry in order to use the ModeParticleLogger simulation mode (see section 4.10). This volume is positioned right before the scintillators. The ModeParticleLogger mode allows to divide time-consuming simulations of interactions in the detector in two stages. The first stage covers the particle interactions in the phantom. The particles that leave the phantom and enter the ParticleRecorder are “killed” (the tracking process is stopped) and their information is saved in a file. Second stage simulation can start from generation of these particles as primaries using a special particle source (see section 3.2.1) which reads particle information from such files. With this approach the long simulation phase in the phantom can be performed only once, thus reducing the simulation time for the follow-up simulations of, e.g, interactions of particles in the detector.

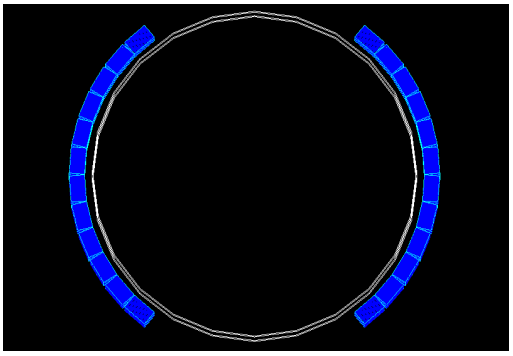
This structure can be enabled by using the following line in the configuration section of main.cc:

```
SM.DetectorComposition.add(DetComp::ParticleLogger);
```

Using the Visualization Mode the image of the ParticleRecorder can be obtained:



In order to better understand its position and orientation, the following image shows this structure together with the scintillators:



3. Source

In Geant4, in order to start a simulation, the primary particle generator has to be configured. The following information has to be provided for each particle to generate: the particle name, its kinetic energy, the xyz coordinates of the origin, the time of generation and the momentum direction.

In this framework the user has to select one of the implemented sources and configure it using the method arguments. The currently available sources are described in the following subsections divided in three groups. First are the beam-related sources, second are the sources based on output files created using the framework and third are the special sources. The description includes the signature of the method used to select the source and explains the arguments.

The following line gives an example of how to construct a source:

*SM.SourceMode = new SourcePoint(new Gamma(0.511*MeV), new ConstantTime(0), {0, 0, 0})*

Using this source, for each event a gamma of 0.511 MeV will be generated at XYZ coordinates of zero in a random direction (isotropic distribution).

Typically, as for the case given above, the particle type and the time generation options are decoupled from the source definition. For example, a point source can be used with any particle, and the time of generation is independent from the spatial characteristics of the source. The available particle types and time generators are described in section 3.4 and 3.5, respectively.

Note that when a direction is not specified in the method arguments it is assumed that the particles are generated isotropically.

3.1. Beam-related sources

3.1.1. Beamlet

This source is used to generate a parallel beam of particles. The user specifies the particle (and its energy), the TimeGenerator, the beam origin and direction. Optionally, it is possible to configure the beam profile in the plane perpendicular to the beam direction. If not provided (default), a pencil beam with zero width is generated. The available options for the profile are discussed at the end of this subsection. Another optional parameter is the number of particles per generation (default is 1): in this case all these particles are generated independently (do not share position or timestamp). The purpose of this option is to reduce the size of the output files for some of the simulation modes which generate small number of output per event (e.g. PES generation). In that case it is preferable to generate, e.g. 1000 particles per event. Note that for that case the total number of generated particles will be the number of events multiplied by the number of particles per generation.

Signature: SourceBeam(ParticleBase* particle, TimeGeneratorBase* timeGenerator, array origin, array direction, int numParticles, ProfileBase* spread)

Arguments:

- particle - the chosen particle object
- timeGenerator - the chosen time generator object

- origin - xyz coordinates of the generation position. If a profile is applied using the spread argument this position will be the center of the profile
- direction - direction of the beam
- numParticles - number of particles to be generated per event.
- spread - profile of the beam. If the argument given is nullptr the beam has zero width. Otherwise a new profile object has to be provided.

The available options for the beam profile are:

➔ Uniform

Signature: UniformProfile(double dullSizeX, double fullSizeY)

Particle positions are generated within a rectangle, perpendicular to the beam direction, with the full sizes dx and dy, and the center at the origin

➔ Round

Signature: RoundProfile(double diameter)

Particle positions are generated within a round area, perpendicular to the beam direction, with the specified diameter and the center at the origin

➔ Gaussian

Signature: GaussProfile(double sigmaX, double sigmaY)

Particle positions are generated on a plane perpendicular to the beam using a Gaussian distribution with the specified sigmaX and sigmaY standard deviations and centered at the origin.

➔ Custom radial

Signature: CustomRadialProfile(string fileName)

Particles positions are generated on a plane perpendicular to the beam using a radial distribution provided by the user assuming axial symmetry. The argument gives the file name which lists the radial distances (in mm) and the corresponding weight factors (format: one pair per line, values are separated with space character). The data should start with the radial distance of 0 and end with a pair having zero weight.

For example, to configure a proton beam with a round profile of 20 mm diameter, the following configuration line can be used:

*SM.SourceMode = new SourceBeam(new Proton(75.8*MeV), new UniformTime(0, 0.1*s), {0*mm, 0*mm, -55.0*mm}, {0,0,1.0}, 1, new RoundProfile(20.0*mm));*

3.1.2. Multi-beamlet

This source is intended for simulations of multi-beamlet treatment plans of MD Anderson. The source operates using the information provided by the user for each beamlet: the nominal proton energy, the mean position at the isocenter, time (start and duration) as well as the statistical weight of the beamlet. The origin of all protons is a point situated 2.52 meters away from the isocenter (Y direction, offsets in X and Z are zero). The position where this proton would hit the iso-plane is then generated based on the given mean beamlet

position and a random offset in X and Z directions (Gaussian distribution, sigma is taken from the calibration dataset provided by MD Anderson). Proton energy is taken from a look-up table (true vs nominal energy). This table was computed also based on the calibration information. In order to avoid passing the protons through air over such a sling distance but keep the divergence information, the protons after generation are “teleported” to a point at their trajectory 150 mm away from the isocenter. The proton generation time is sampled from uniform distribution with the given start time and duration.

The total number of generated protons is defined by the user. Note that the user provides the statistical weights for each beamlet, which is used to compute how many protons will be generated in each particular beamlet:

$N_{beamlet} = N_{total} * StatWeight_{beamlet} / SumStatWeights$

The information for the beamlets are provided in an ASCII file, data for each beamlet in a new line. The file has the following format:

NominalEnergy IsoX IsoZ TimeStart TimeSpan StatWeight

where

- NominalEnergy is the beamlet’s nominal energy
- IsoX and IsoZ are the mean position coordinates of the beamlet at the isoplane
- TimeStart and TimeSpan define the beamlet start time and duration
- StatWeight is the statistical weight of the beamlet.

Each line is terminated with ‘\n’ (end of line) symbol. Space character is the delimiter.

Multi-beamlet calibration data (true energy and beamlet sigma) are read from the ASCII file BeamletCalibration.txt, situated at the source root directory of the framework. Note that this file is automatically copied to the run directory during compilation: this copy of the file is used for calibration! This file has the following format:

NominalEnergy TrueEnergy SpotSigma

where

- NominalEnergy is the nominal beamlet energy in MeV
- TrueEnergy is the energy attributed to the beamlet protons (MeV)
- SpotSigma is the beamlet full width at half maximum (mm) for this energy

Each file line is terminated with ‘\n’ (end of line) symbol. Space character is the delimiter.

Note that linear interpolation of the two closest datapoints is applied for the energy and width data during calibration. If the nominal energy is less than the smallest or larger than the largest value present in the calibration dataset, the data corresponding to the smallest or the largest energy are used.

Source signature:

SourceMultiBeam(ParticleBase particle, string beamletFileName, double totalParticles)*

Arguments:

- particle - the chosen particle object. During normal simulations this argument value should be ***new Proton()***. However, for tracking diagnostic purposes, it is also possible to configure it to ***new Geantino()***.
- beamletFileName - full name of the file (with directory) with the information on the beamlets to generate
- totalParticles - total number of generated particles over all beamlets together. Note that it is generally a good idea to synchronize the number of particles configured in this source and in the simulation mode.

3.2. Sources reading information from files

3.2.1. Particles source reading from list file

Individual particles are generated according to the information provided in the specified file. For each particle, the file lists the particle name (Geant4 system), energy (keV), xyz coordinates of the origin (mm), direction unit vector and time (ns).

Signature: SourceParticleListFile(string FileName, bool BinaryFile)

Arguments:

- FileName - full name (with the directory) of the input file
- BinaryFile - set to true if the file has the binary format (otherwise it has ASCII format)

For the ASCII files the format is the following:

#EventIndex

ParticleName Energy X Y Z Direction[0] Direction[1] Direction[2] Time

Each line is terminated with '\n' (end of line) symbol. The '#' symbol precedes the event index (integer from 0, at least one event has to be present in the file). There could be empty events.

Particle parameters are:

- ➔ ParticleName - particle name using Geant4 system (e.g. proton or e+)
- ➔ Energy - energy in MeV
- ➔ X, Y and Z - origin position coordinates in mm
- ➔ Direction[0], Direction[1], Direction[2] - direction unit vector
- ➔ Time - particle generation time in ns

Space character is the delimiter. More than one particle can be defined in an event.

For the binary files the format is the following:

0xEE EventIndex(int)

0xFF Particlename(zero terminated string) Energy(double) X(double) Y(double) Z(double) Direction[0](double) Direction[1](double) Direction[2](double) Time(double)

Note that 0xEE and 0xFF chars are used to indicate the type of the following record: event index or particle information. There are no other delimiters.

This source is typically used in order to generate positron emission species using the output file generated with the brute-force approach (see section 4.3). It can also be used with the output file of Particle Logger simulation mode (see section 4.4).

3.2.2. PES source reading from histogram files

This source can be used to generate positron emitters according to the spatial distribution histograms of the activity, generated by several simulation modes (e.g. probability-based PES generation, see chapter 4.4) and stored in a specified directory. The histograms have the following naming format:

A_B_C.dat

where *A* is the proton number of the parent isotope, *B* is the nucleon number of the parent isotope and C is the symbol of the daughter isotope (element symbol followed by the nucleon number). For example, the histogram file for ^{15}O generated from ^{16}O would have the name 8_16_O15.txt.

Note that the histogram file has a header section (the first line of the file terminated with '\n' character), starting with '#' character. The header contains binning information in JSON format. This information is given as arrays of three numbers (X, Y and Z direction), and the naming is the following:

- NumBins - number of bins
- BinSize - size of bins in mm
- Origin - the coordinates of the origin corner (not the center!) of the 0,0,0 bin (in mm).

The number of the generated positron emitters is given by the activity value of the bin multiplied by a user-defined scaling factor. As the number has to be an integer, the rounded down result of the multiplication is used.

Generation of the positron emitter positions can be done using two approaches: the bin center only or uniformly distributed over the bin volume. The generated particle is attributed with zero energy, so there is no drift before the radioactive decay is triggered.

Signature: SourcePesHistogramFiles(string directory, double multiplier, bool generateUniformOverBin)

Arguments:

- directory - name of the directory where the histogram files are stored
- multiplier - value of the multiplier
- generateUniformOverBin - when set to true, the positron emitter positions are uniformly distributed over the bin volume. Otherwise, the generation position is always the bin center

3.2.3. Source of gamma pairs using a histogram of the annihilation positions

This source is used to generate back-to-back gamma pairs of 0.511 MeV according to the specified histogram of the spatial distribution of the activity. The number of the generated gamma pairs is given by the activity value of the bin multiplied by a user-defined scaling factor. As the number has to be an integer, the rounded down result of the multiplication is used.

Generation of the positron emitter positions can be done using two approaches: the bin center only or uniformly distributed over the bin volume. Their direction of the first gamma of the pair is sampled assuming isotropic distribution. The direction of the second gamma opposite to the first one.

Signature: SourceAnnihilHistFile(string histogramFileName, double activityMultiplier, bool generateUniformOverBin)

Arguments:

- histogramFileName- full name of the histogram file
- activityMultiplier - value of the multiplier
- generateUniformOverBin - when set to true, the positron emitter positions are uniformly distributed over the bin volume. Otherwise, the generation position is always the bin center

This source is typically used to generate gamma pairs based on the output file of the activity generation simulation mode (see section 4.6).

3.3. Special sources

3.3.1. Point source

The specified particle is generated at a point with the given coordinates. The time is sampled using the specified TimeGenerator. The particle direction is sampled assuming isotropic distribution.

Signature: SourcePoint(ParticleBase* particle, TimeGeneratorBase* timeGenerator, array origin)

Arguments:

- particle - the chosen particle
- timeGenerator - the chosen time generator
- origin - xyz coordinates of the generation position

This source can be useful for generation of datasets for testing reconstruction algorithms. One can use, for example, a GammaPair particle (in this case a back-to-back gamma pair is emitted) to have all lines of response (LOR) in reconstruction crossing exactly the same point. Alternatively, the particle can be a positron-emitting isotope (such as C11). In this case gamma pairs will be emitted only after annihilation of a positron, created during the radioactive decay. Thus the LORs will “originate” from the point-centered region blurred due to the positron drift.

3.3.2. Na22 point source

Due to long half-life (2.6 years), using a Na22 isotope with the “normal” point source (section 3.3.1) is not a good idea as the time delay introduced by the decay would strongly affect the timestamp produced by the time generator. This source allows the user to avoid such a problem.

Na22 decays into an excited state of Ne22 either by electron capture (9.7%) or beta plus decay (90.2%) and when this isotope returns to the ground state a gamma of 1.275 MeV is emitted.

The source operates in the following way: first a gamma with 1.275 MeV is emitted at the specified position assuming isotropic distribution; the emission time is random, it is uniformly distributed in the time interval provided by the user. Then a random number in the range from 0 and 1 is generated and if it is larger than 0.096 a pair of back-to-back 0.511 MeV gammas is also generated at the same time and position (also isotropic distribution).

Signature: SourceNa22Point(double timeFrom, double timeTo, array origin)

Arguments:

- timeFrom - start time
- timeTo - end time
- origin - xyz coordinates of the generation position

3.3.3. Line source

The specified particle is generated along a line defined by the given start and end xyz coordinates. The timestamp is provided by the specified TimeGenerator. The particle direction is random (isotropic distribution).

Signature: SourceLine(ParticleBase* particle, TimeGeneratorBase* timeGenerator, array startPoint, array endPoint)

Arguments:

- particle - the chosen particle
- timeGenerator - the chosen time generator
- startPoint - start xyz coordinates of the line
- endPoint - end xyz coordinates of the line

3.3.4. Cylindrical source

The specified particle is generated inside a cylinder-shaped region of space with the dimensions given by the user. The time is sampled using the specified TimeGenerator. The particle direction is random (isotropic distribution).

Signature: SourceCylinder(ParticleBase* particle, TimeGeneratorBase* timeGenerator, double radius, array startPoint, array endPoint, string fileName)

Arguments:

- particle - the chosen particle
- timeGenerator - the chosen time generator
- radius - radius of the cylinder source

- startPoint - xyz coordinates of the center position of the cylinder's lower base
- endPoint - xyz coordinates of the center position of the cylinder's upper base
- fileName - Name of the output file. When the argument is not provided, the output file is not generated.

If "fileName" argument is provided, an ASCII file is generated listing the XYZ coordinates (mm) of the created particles, each given in a new line with the following format:

X Y Z

The lines are terminated with '\n' (end of line) symbol. The XYZ coordinates are separated by a space character.

3.3.5. Material limited source

This source is used to generate the specified particle, uniformly inside a given spatial region (inside a bounding box) but rejecting those positions which are not occupied by a specified material. For each generation, 10000 attempts to "find" the specified material is made (if not found, the particle is not generated). The time is sampled using the specified TimeGenerator. The particle direction is sampled assuming isotropic distribution.

Signature: SourceMaterialLimited(ParticleBase* particle, TimeGeneratorBase* timeGenerator, array origin, array boundingBoxFullSize, string material, string fileName_EmissionPositions)

Arguments:

- particle - the chosen particle
- timeGenerator - the chosen time generator
- origin - xyz coordinates of the center of the bounding box
- boundingBoxFullSize - xyz size of the bounding box
- material - name of the material to which particle generation is limited to
- fileName_EmissionPositions - Name of the output file. If this argument is not provided, the output file is not generated.

If "fileName_EmissionPositions" argument is provided, an ASCII file is created listing the XYZ coordinates (in mm) of the generated particles, one per line, and separated with the space character. Each line is terminated with '\n' (end of line) symbol.

3.3.6. LYSO natural radioactivity source

This source is used to simulate the natural radioactivity from Lu176 inside the scintillators of the TPPT scanner. Lu176 mainly decays into an excited state of Hf176 (excitation energy of 596.820 keV) emitting an electron (broad energy distribution up to ~600 keV). The excited Hf176 also decays, typically emitting one, two or three gammas.

The decay position is generated randomly (uniform distribution over all volume occupied by the scintillators). Then the electron is generated (isotropic) with the energy sampled from the pre-loaded distribution. Finally, Hf176 in the excited state is generated, which decay is handled by Geant4. The time of all emitted particles is randomly selected in the user-defined interval (uniform distribution).

Signature: SourceLysoNatural(double timeFrom, double timeTo)

Arguments:

- timeFrom - start time
- timeTo - end time

3.3.7. Source mixer

This source is used to combine more than one source in a simulation run. The user provides a list of sources and their respectful statistical weights. For generation, one of the sources is randomly selected based on the statistical weights.

Signature: SourceMixer(array of pairs of {BaseSource* source, double statWeight});

Arguments:

- source - a pointer to any source object
- statWeights - statistical weight (non-negative number)

Example:

```
SM.SourceMode = new SourceMixer ({
        { new SourcePoint ( new GammaPair, new ConstantTime(0), {0, 0, 0}),     0.80},
        { new SourcePoint ( new Proton, new ConstantTime(0), {0, 0, 50*mm}),    0.20}
})
```

3.4. Implemented particles

The particle objects inherit their properties from class ParticleBase.

Available particles:

- Geantino
- Gamma
- Gamma Pair
- Proton
- Isotopes
    - C10
    - C11
    - O15
    - N13
    - N12
    - Hf176exc

Other isotopes can also be created using the following constructors:

Isotope(int z, int a) or Isotope(int z, int a, double excitationEnergy)

where

- z and a are the isotope atomic and mass number
- excitationEnergy is the energy of the excited state of the isotope.

3.5. Time generators

The following time generators are currently implemented:

- Constant Time

  ConstantTime(double time)

  All particles are generated at the same time given timestamp

- Uniform Time

  UniformTime(double timeFrom, double timeTo)

  Timestamps of the generated particles are uniformly distributed in the time period given by the method's arguments.

- Exponential Time

  ExponentialTime(double timeFrom, double halfLife)

  Timestamps of the generated particles are exponentially distributed in the period of time from timeFrom and according to the configured decay half-time.

4. Simulation Mode

Simulation mode defines what task is to be conducted during the simulation. All the names of the simulation modes start from the “Mode” word, for example, ModeDoseExtractor, used to compute the spatial distribution of the dose (or deposited energy) or ModePesGenerator_MC, which is used to generate positron emitting species inside the phantom.

The following sub-sections describe the simulation modes which are currently implemented in the framework. The description includes the signature of the method used to configure the selected mode and explains the arguments.

For example, to select ModeDoseExtractor simulation mode, one can use the following:

*SM.SimMode = new ModeDoseExtractor(1e5, {1.0*mm, 1.0*mm, 1.0*mm}, {100, 100, 100}, {-50.0*mm, -50.0*mm, -50.0*mm}, “Dose.txt”)*

Using this configuration, the simulation will record in a file named “Dose.txt” the 3D spatial distribution of the dose accumulated in the configured phantom during 10^5 events (define by the configured source) in a region split in 1 x 1 x 1 mm^3 voxels, with the origin coordinates of (-50,-50,-50) and 100 voxels along each dimension.

4.1 Energy deposited in scintillators

This mode is used to record energy deposition in the scintillators of the TPPT scanner. The data include the scintillator index, the deposited energy and the timestamp for each deposition. The obtained data can be post-processed in order to identify the PET coincidence pairs using the TPPTbuild and TPPTcoinc sub-frameworks (see the respective manuals).

This mode allows to enable pre-clustering of the deposition records in order to reduce the data size and the processing time. For each scintillator, the clustering algorithm merges the neighboring deposition records if the difference in the timestamps between them is less than a given threshold. In the merging process the timestamp of the combined record is computed as the energy-weighted timestamp of the two records, and the energies are summed. This process ignores the position inside the scintillator assuming that it will not affect the time and energy of the detected events.

Mode signature:

ModeDepositionScint(int numEvents, string fileName, bool binary, size_t maxCapacity, bool doCluster, double maxTimeDif)

Arguments:

- numEvents - number of events in this simulation run
- fileName - name of the output file
- binary - type of output file. The flag is set to true for binary type and false for ASCII
- maxCapacity - size for the memory buffer. The default value is 10000 records per scintillator
- doCluster - if true, the pre-clustering process will be conducted
- maxTimeDif - threshold for the difference in arrival time for pre-clustering. The default value is 0.01 ns

The output file lists the energy deposition records (pairs of time[ns] and energy[MeV]), grouped by the scintillator index.

For the ASCII files the format is the following:

 # ScintIndex
Time_1 Energy_1
Time_2 Energy_2
…
Time_n Energy_n
 # ScintIndex+1
Time_1 Energy_1
Time_2 Energy_2
…
Time_m Energy_m
…

Each line is terminated with '\n' (end of line) symbol. The '#' symbol marks the lines which contain the scintillator index (ScintIndex). Space character separates # symbol and the scintillator index, as well as the time mark and the energy values.

For the binary files the format is the following:

0xEE ScintIndex(int)
0xFF Time_1(double) Energy_1(double)
0xFF Time_2(double) Energy_2(double)
…
0xFF Time_n(double) Energy_n(double)
0xEE ScintIndex+1(int)
0xFF Time_1(double) Energy_1(double)
0xFF Time_2(double) Energy_2(double)
…
0xFF Time_m(double) Energy_m(double)
…

Note that 0xEE and 0xFF chars are used to indicate the type of the following record: scintillator index or time/energy record. There are no other delimiters.

To speed-up the simulation process and have a well-defined requirement on memory usage, the collection of the deposition records is conducted using a memory buffer of a fixed capacity. When the buffer is filled, all records are saved to the output file, ordered by the scintillator index. The buffer is then cleared and simulation continues. Thus the same file can have several entries for the same scintillator.

4.2 Dose distribution

This simulation mode is used to record the dose (or, alternatively, the deposited energy) in a specified region of space. This region is defined by a given number of box-shaped voxels

and their sizes for each direction (in X, Y and Z) and the origin position in the world. Note that the origin is given not for the (#0,#0,#0) voxel center but to the corner of that voxel.

The dose (or the deposited energy) is recorded per voxel. Note that the dose is collected as the ratio of the energy deposited in the voxel (in Joules) divided by the mass of the voxel (in kg). If the voxel is occupying a region of space with different materials, each particular deposition will contribute to the accumulated dose assuming the mass of the voxel based on the material at the current deposition position. The deposited energy is collected in MeV. Note that to avoid artifacts in the distribution related to binning (or a fixed maximum step length), the “interaction position” is computed as a random position between the beginning and the end of the current tracking step (uniform distribution).

Mode signature:

ModeDoseExtractor(int numEvents, array binSize, array numBins, array origin, string fileName, bool energyDepositionMode)

Arguments:

- numEvents - number of times the source will be called in this simulation. Thus if the source is configured to emit a single particle at a time, this argument defines directly the number of primaries for this simulation
- binSize - xyz size of the voxel (e.g. {1*mm,100*mm,100*mm} defines voxels with the size of 1 mm in x, and 100 mm in y and z directions)
- numBins - number of voxels in X,Y and Z directions (e.g. {100,1,1} configures the simulation to use 100 voxels in X and 1 voxel in Y and Z direction)
- origin - xyz coordinates of the origin of voxels (e.g. {-50*mm, -50*mm, -50mm})
- fileName - name of the output file
- energyDepositionMode - flag which selects what kind of information is to be collected: if set to false, the dose information will be collected, if true, the deposited energy will be collected

The generated ASCII output file contains a 3D histogram of the dose (or deposited energy) per voxel. The dose is given in Gy, and the deposited energy in MeV. The header (the first line of the file) start with the # character and contain information on the binning (json format). The file has the following format:

#{“BinSize”: [BinSizeX, BinSizeY, BinSizeZ], “NumBins”: [NumBinsX, NumBinsY, NumBinsZ], “Origin”: [OriginX, OriginY, OriginZ]}

Energy_1_1_1 Energy_1_1_2 …. Energy_1_1_N_z

Energy_1_2_1 Energy_1_2_2 …. Energy_1_2_N_z

….

Energy_1_N_y_1 Energy_1_N_y_2 …. Energy_1_N_y_N_z

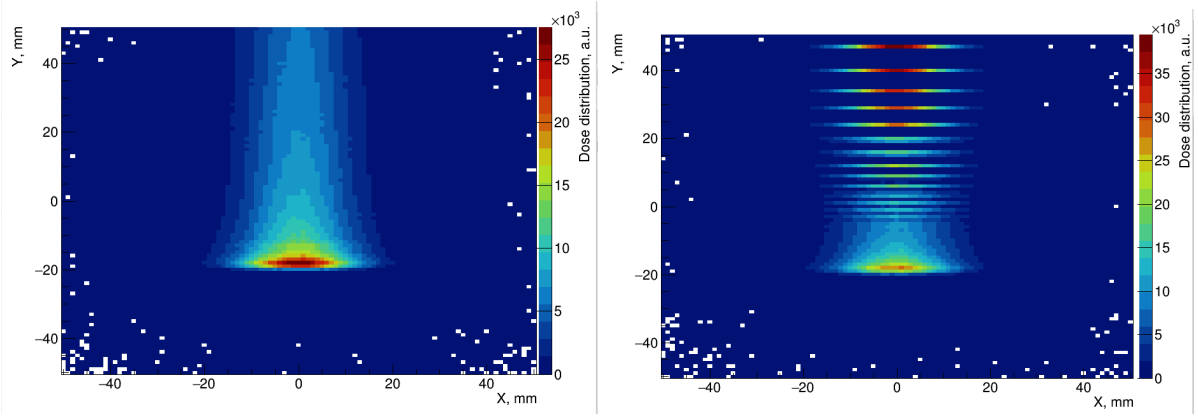
Energy_2_1_1 Energy_2_1_2 …. Energy_2_1_N_z

….

Energy_N_x_N_y_1 Energy_N_x_N_y_2 …. Energy_N_x_N_y_N_z

Each line is terminated with '\n' (end of line) symbol. Space character is used as the delimiter. N_x represents the number of bins in the x direction which is given by NumBins[0]. N_y represents the number of bins in the y direction which is given by NumBins[1]. N_z represents the number of bins in the z direction which is given by NumBins[2].

When using this simulation mode it is important to activate the step limiter (see additional options used to configure the simulation in chapter 2) to avoid artifacts in the output: For example, the following graphs show the dose distribution for a proton beam in a bone phantom for two cases, with and without activated step limiter of 1 mm.



With step limiter

Without step limiter

4.3 PES generation using full Monte Carlo approach

This simulation mode generates a list of positron emitting isotopes produced inside the phantom for the configured source of primary protons. The record for each individual isotope includes its symbol (e.g. C11 for carbon-11), XYZ position and timestamp.

The primary protons are tracked using Geant4, and all the generated secondaries are suppressed (not tracked). The generation of the positron emitter is conducted with a custom Monte Carlo procedure using the production cross-sections for a specific set of proton induced reactions.The default list of reactions with the corresponding cross-sections is taken from [Phys. Med. Biol. 58 (2013) 5193–5213; doi:10.1088/0031-9155/58/15/5193] and includes 12C(p, pn)11C, 16O(p, 3p3n)11C, 16O(p, pn)15O,16O(p, 2p2n)13N and 40 Ca(p, 2pn)38K. That list of implemented channels and the corresponding energy-resolved cross-sections can be extended or modified by the user (see the end of this chapter).

When the simulation starts the cross-sections are automatically loaded and the generation procedure is configured for all materials defined in the simulation. During each Geant4's tracking step of the primary proton the relative probability for all possible generation channels are calculated and using a random number generator a potentially triggered channel is selected and the trigger step is computed for the current material. If this step is shorter than the proton's step given by Geant4, the positron emitter is generated at the corresponding position and the timestamp is attributed (mean value of the timestamp of the proton at the beginning and the end of the current step). Triggering of the reaction is not causing stopping of the tracking of the primary proton.

Mode signature:

ModePesGenerator_MC(int numEevents, string outputFileName, bool binaryOutput)

Arguments:

- numEvents - number of proton events
- outputFileName - name of the output file
- binaryOutput - type of the output file (true for binary and false for ASCII)

Each record contains the symbol of the positron emitter (PES), energy (MeV), X Y and Z coordinates of the generation position in mm, a unitary direction vector and the timestamp in ns. Note that the energy and the direction are always 0 and (1,0,0), respectively. They are present in the records for compatibility with the particle source which generates primary particles based on the records in a particle file (see chapter 3.2.1).

For the ASCII files the format is the following:

#EventIndex

PES Energy X Y Z Direction[0] Direction[1] Direction[2] Time

Each line is terminated with '\n' (end of line) symbol. The '#' symbol precedes the event index. Space character is used as the delimiter. Note that more than one record can be generated per event.

For the binary files the format is the following:

0xEE EventIndex(int)

0xFF PES (zero terminated string) Energy(double) X(double) Y(double) Z(double) Direction[0](double) Direction[1](double) Direction[2](double) Time(double)

Note that 0xEE and 0xFF chars are used to indicate the type of the following record: Event index or PES information. There are no other delimiters.

The ascii file with the reaction channels and cross-section information is distributed with the framework (ProductionCrossSections.txt in the main directory). Note that during compilation this file is automatically copied to the run directory. So the modification had to be performed in the "source" directory and the framework recompiled for the changes to be in effect. The file has the following format:

#TargetZ TargetA PES DecayTime

Energy_1 CrossSection_1

Energy_2 CrossSection_2

…

where

- TargetZ is the atomic number of the isotope
- TargetA is the mass number of the isotope
- PES is the isotopes symbol, e.g. to represent the isotope of carbon-11 the symbol has to be C11
- Isotope's half-time in seconds
- Energy in MeV
- Cross-section in millibarn

Each line is terminated with '\n' (end of line) symbol. The '#' symbol marks the beginning of a data block for a new channel. Space characters are used as delimiters.

4.4 PES generation using probability-based approach

This simulation mode is an extension of the one described in chapter 4.3. Its purpose is to generate the spatial distribution of the PET activity avoiding as much as possible the statistical uncertainties in the position distribution characteristic to the full Monte Carlo-based approaches (as the one described in chapter 4.3). This mode uses a probabilistic approach: during each tracking step of the primary proton, the probabilities of isotope generation for each of the configured reaction channels (see chapter 4.3) are accumulated directly.

The user configures the 3D binning for the region of interest and the time interval when the PET scanner will acquire the coincidence events. As with the MC-based mode, only the primary protons are tracked and the custom production cross-section are used (see chapter 4.3). For each step, the crossed bins (and the corresponding material) are identified and the trajectory length inside each of them is established. Using the production cross-sections, the probability for generation of each PES is calculated. For each PES channel, using the proton's timestamp (assuming to give directly the isotope production time), and the half-time of the corresponding isotope, the expected contribution to the PET activity is computed and added to the corresponding bin of the spatial histogram for that PES channel.

This mode allows using a significantly smaller number of primary protons to generate activity spatial profiles of a comparable quality (low level of statistical fluctuations) in comparison to the approach of chapter 4.3. This leads to a dramatic reduction of the simulation time (by a factor of 1000), however, there is no individual (by each particle) information on the production time and position.

Mode signature:

ModePesGenerator_Prob(int numEvents, array of doubles binSize, array of ints numBins, array of doubles origin, array of a pair of doubles acquisitionFromDurationPairs)

Arguments:

- numEvents - number of events
- binSize - xyz size of the bin
- numBins - number of bins
- origin - xyz coordinates of the first bin location (corner, not center)
- acquisitionFromDurationPairs - time period of observation: it is a pair of values, the first is timeFrom, the second is duration.

Output of this mode is a set of ASCII files with 3D histograms of the PET activity in a specified region of space for each configured PES reaction channel. Each bin stores the number of observable decays (during the configured time windows) in that location. In the file header the information on the spatial binning is also stored. The ASCII files have the following format:

#{"BinSize": [BinSizeX, BinSizeY, BinSizeZ], "NumBins": [NumBinsX, NumBinsY, NumBinsZ], "Origin": [OriginX, OriginY, OriginZ]}

NumDecays_1_1_1 NumDecays_1_1_2 …. NumDecays_1_1_N_z

NumDecays_1_2_1 NumDecays_1_2_2 …. NumDecays_1_2_N_z

….

NumDecays_1_N_y_1 NumDecays_1_N_y_2 …. NumDecays_1_N_y_N_z

NumDecays_2_1_1 NumDecays_2_1_2 …. NumDecays_2_1_N_z

….

NumDecays_N_x_N_y_1 NumDecays_N_x_N_y_2 …. NumDecays_N_x_N_y_N_z

Each line is terminated with '\n' (end of line) symbol. The '#' symbol marks the start of the header with the binning information in the json format. The space character is used as the delimiter. N_x represents the number of bins along the x axis: numBinsX (same approach for N_y and N_z).

4.5 PET activity generation using probability-based approach

This simulation mode is similar to the one described in 4.4, but with one substantial difference: instead of providing the spatial distribution of the observable activity for each reaction channel independently, this mode outputs the combined activity for all channels. More specifically, this output represents the combined PET activity induced by the treatment and observable during the specified time intervals in the given region of space.

Mode signature:

ModeActivityGenerator(int numEvents, array binSize, array numBins, array origin, array of pair acquisitionFromDurationPairs, string fileName)

Arguments:

- numEvents - number of events
- binSize - xyz sizes of the bin
- numBins - number of bins in x, y and z directions
- origin - xyz coordinate of the first bin location (corner, not center)
- acquisitionFromDurationPairs - time period of observation: it is a pair of values, the first is timeFrom, the second is duration
- fileName - the name of the output file.

Output: an ASCII file with a 3D histogram of the observable PES activity. Each bin stores the expected number of decays in that location during the specified time intervals of the PET data acquisition. The file has the following format:

#{"BinSize": [BinSizeX, BinSizeY, BinSizeZ], "NumBins": [NumBinsX, NumBinsY, NumBinsZ], "Origin": [OriginX, OriginY, OriginZ]}

NumDecays_1_1_1 NumDecays_1_1_2 …. NumDecays_1_1_N_z

NumDecays_1_2_1 NumDecays_1_2_2 …. NumDecays_1_2_N_z

….

NumDecays_1_N_y_1 NumDecays_1_N_y_2 …. NumDecays_1_N_y_N_z

NumDecays_2_1_1 NumDecays_2_1_2 …. NumDecays_2_1_N_z

….

NumDecays_N_x_N_y_1 NumDecays_N_x_N_y_2 …. NumDecays_N_x_N_y_N_z

Each line is terminated with '\n' (end of line) symbol. The '#' symbol marks the start of the header with the binning information in the json format. The space character is used as the delimiter. N_x represents the number of bins along the x axis: numBinsX (same approach for N_y and N_z).

4.6 Positron annihilation positions

This simulation mode is used to collect spatial distribution of the positron annihilation positions in a specified region of space. The user configures the 3D binning for the region of interest. Each bin represents the number of annihilated positrons in that location.

Mode signature:

ModeAnnihilationLogger(int numEvents, array binSize, array numBins, array origin, string fileName)

Arguments:

- numEvents - number of events
- binSize - xyz sizes of the bin
- numBins - number of bins in x, y and z directions
- origin - xyz coordinate of the first bin location (corner, not center)
- fileName - name of the output file

Output: an ASCII file with a 3D histogram of the number of positrons annihilations in the specified region of space. The file has the following format:

#{"BinSize": [BinSizeX, BinSizeY, BinSizeZ], "NumBins": [NumBinsX, NumBinsY, NumBinsZ], "Origin": [OriginX, OriginY, OriginZ]}

PosAnnihil_1_1_1 PosAnnihil_1_1_2 …. PosAnnihil_1_1_N_z

PosAnnihil_1_2_1 PosAnnihil_1_2_2 …. PosAnnihil_1_2_N_z

….

PosAnnihil_1_N_y_1 PosAnnihil_1_N_y_2 …. PosAnnihil_1_N_y_N_z

PosAnnihil_2_1_1 PosAnnihil_2_1_2 …. PosAnnihil_2_1_N_z

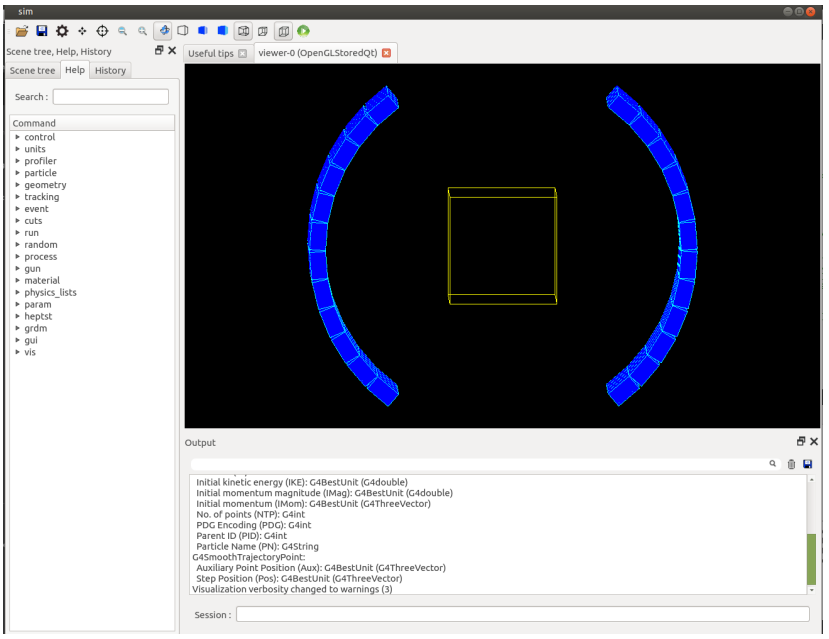
….

PosAnnihil_i_N_y_1 PosAnnihil_i_N_y_2 …. PosAnnihil_i_N_y_N_z

Each line is terminated with '\n' (end of line) symbol. The '#' symbol marks the start of the header with the binning information in the json format. The space character is used as the delimiter. N_x represents the number of bins along the x axis: numBinsX (same approach for N_y and N_z).

4.7 Visualization mode

This mode is used to draw the detector's geometry using the Qt-based visualization tool provided by Geant4. It can also be used to show the tracks resulting from a simulation run with one call of the configured source of the primaries.



Example of visualization of the detector geometry.

The toolbar with buttons in the top toolbar of the window can be used for several important actions such as changing the type of the 3D object representation (wireframe or solid), viewport manipulation, and, most importantly, start a simulation with a single run of the particle source (the green triangular button on the most right).

In the left sidebar there is a list of commands that can be activated using the "Session" text box at the bottom. For example, the detector geometry can be checked for geometric conflicts (overlaps and extrusions) using the following command: /geometry/test/run. Note that it is possible to select a command using the mouse double-click and then execute it by pressing the Enter key.

Mode signature:

ModeGui()

There are no arguments.

4.8 Particle tracing

This simulation mode can be used to perform different tests of the detector geometry by configuring a source of test particles ("geantino") with a fixed direction and then observing the log of the trajectory of the particle. The mode automatically starts the Geant4's

visualization tool (see chapter 4.7). Tracking can be started by pressing the green triangle button on the top-right of the window.

Mode signature:

ModeTracing()

There are no arguments.

Output: A log in the application terminal with the tracking information containing the names of the geometric volumes crossed by the particle as well as the entrance points for the volumes.

4.9 Particle logger

This simulation mode can be used to split time-consuming simulations in stages. It allows storing information on the particles which arrive at the special geometric volume “ParticleLogger” (see section 2.2) in a file. The information consists of the full particle name, energy, position and direction. After saving the information for a particle to the file, tracking for that particle is stopped. The information contained in the file can be read using a special type of the particle source (see section 3.2.1) to generate the primaries so the simulation can be continued in the next stage.

Mode signature:

ModeParticleLogger(int numEvents, string fileName, bool bBinary)

Arguments:

- numEvents - number of events
- fileName - name of the output file
- bBinary - format of the output: the flag is set to true for binary and false for ASCII

Output: a file with a list of particle records. Each record contains the particle name (e.g. “gamma” or “He4”), energy in keV, XYZ coordinates (mm), a unit vector of the momentum direction and the timestamp (ns).

For the ASCII files the format is the following:

#EventIndex

Particle Energy X Y Z DirectionX DirectionY DirectionZ Time

Each line is terminated with ‘\n’ (end of line) symbol. The ‘#’ symbol precedes the number of the event. Space character is the delimiter. More than one particle record can be present in an event.

For the binary files the format is the following:

0xEE EventIndex(int)

0xFF Particle(zero_terminated_string) Energy(double) X(double) Y(double) Z(double) DirectionX(double) DirectionY(double) DirectionZ(double) Time(double)

Note that 0xFF and 0xEE chars are used to indicate the type of the following record: event index or the particle record. There are no other delimiters.

4.10 Test modes

There are some other “miscellaneous” modes which were created during the framework development phase for test purposes. They are, however, can be useful if someone would like to modify existing modes or add custom ones.

The list of test modes:

- ModeShowEvent
- ModeTestScintPositions
- ModeTestAcollinearity
- ModeTestAnnihilations
- ModeTestLysoNatRad
- ModeTestDepositionStat