Abstract

This document describes how track length based fluence actors *TLFLuenceActor* and *TLFluenceDistributionActor* were implemented in GATE and how to use them. Examples on how to use the actors accompany the code. These examples were also used to validate the actors functionality.

Description of track length based fluence actors (*TLFLuenceActor* and *TLFluenceDistributionActor*) for fluence measurement in GATE

Anders Garpebring[1]

[1]Department of Radiation Sciences, Umeå Univeristy, Umeå, Sweden , anders.garpebring@umu.se

February 18, 2020

1 Introduction

Several Actors designed for different tasks are available in GATE; e.g. a dose actor. However Actors enabling measurement of spatially resolved fluence and avuncularly resolved fluce are missing. The goal of the developed *TLFLuenceActor* was to create an actor with which fluence (and energy fluence) can be measured in a spatially resolved way in GATE. The goal of the developed *TLFluenceDistributionActor* was to create an actor with which differential fluence can be measured in GATE.

2 Theory

The radiation field can be completely described by the particle density $n(t, \mathbf{r}, E, \mathbf{\Omega})$, where t denotes time, $\mathbf{r}$ denotes position, E is energy and $\mathbf{\Omega}$ is direction. From the particle density, one can define the angular flux density as $\Phi(t, \mathbf{r}, E, \mathbf{\Omega}) = v(E)n(t, \mathbf{r}, E, \mathbf{\Omega})$. Integrating the angular flux density over time, energy, and angles yields the fluence of particles, $\Phi(\mathbf{r})$ i.e.

$$\Phi(\mathbf{r}) = \int_{-\infty}^{\infty} dt \int_{-\infty}^{\infty} dE \int_{4\pi} d\mathbf{\Omega} \Phi(t, \mathbf{r}, E, \mathbf{\Omega}). \quad (1)$$

In a finite volume V the average fluence is given by $\bar{\Phi} = \int_V d\mathbf{r}\Phi(\mathbf{r})$. If the particle tracks are considered classical (and non-interacting in V) the particle density can be written as a sum over individual particle tracks

$$n(t, \mathbf{r}, E, \Omega) = \sum_i \delta(\mathbf{r} - \mathbf{r}_i - \mathbf{v}_i t)\delta\left(\mathbf{\Omega} - \frac{\mathbf{v}_i}{\|\mathbf{v}_i\|}\right)\delta(E - E_i), \tag{2}$$

where δ is the Dirac delta function. Inserting this into the equation for average fluence gives a relationship between total track length and fluence, see [SD15] for more details,

$$\bar{\Phi} = \sum_i \tau_i, \tag{3}$$

where τ_i is the total track length for particle i in volume V.

3 Implementation

The track length was summed for each step in the particle tracking to yield to total tracking length and subsequently divided by the voxel volume to yield fluence. To enable uncertainty calculations total track length for a particular primary event was added before the squared length was evaluated for that event.

3.1 TLFluenceActor specific

To assign track length for a particular step to a specific voxel (needed for the TLFluenceActor) one ideally needs to know the fraction of the track length that belong to each voxel. Although this is possible to do by considering the start and end point of a step and the corners of the voxels, another much simpler Monte Carlo based approach was used here. Fractions of the track length was randomly (uniform distribution) assigned to voxels along the step in such way that the total assigned track length to all voxels equalled the total step length. The algorithm to assign fractions of the step to each voxel is:

1. For a step, let the number of samples be $n_s = \lceil \alpha s \cdot max_{i\in\{x,y,z\}}|d_i|/v_i \rceil$, where v_i is the voxel size, d_i is a directional cosine of the step which has length s. A high α improve homogeneity of how the length of the step is distributed among the voxels. At the moment $\alpha = 1$ (hard coded) implying that the step is subdivided into approximately the same number of voxels as the step crosses.

2. Draw n_s positions uniformly and randomly along the step.

3. For each position acquire which voxel it belong to and add the track length s/n_s to that voxel.

4 Usage

4.1 TLFluenceActor

To use the *TLFLuenceActor* simply add it to a geometric object and specify the number of voxels in each dimension that the actor should have. An example of how this is accomplished looks like this:

```
/gate/actor/addActor TLFluenceActor Detector
/gate/actor/Detector/save output/detector.mhd
/gate/actor/Detector/attachTo detectorPlace
/gate/actor/Detector/setResolution 10 10 10
```

The first line names the actor *Detector*. The second line tells where to store the output and how the files should be named. The third line attaches the detector to a geometric object by the name *detectorPlace*. Finally, on the last line the resolution of the detector is set to 10 by 10 by 10 voxels.

What information to store ((energy) fluence, (energy) fluence uncertainty and squared (energy) fluence) can be set by enabling/disabling properties.

```
# Enable properties - Fluence
/gate/actor/Detector/enableFluence true
/gate/actor/Detector/enableFluenceUncertainty true
/gate/actor/Detector/enableSquaredFluence true

# Enable properties - Energy Fluence
/gate/actor/Detector/enableEnergyFluence true
/gate/actor/Detector/enableEnergyFluenceUncertainty true
/gate/actor/Detector/enableSquaredEnergyFluence true
```

Similar to other actors, filters can be used to select what kind of fluence the actor should register. E.g. to only register photons

```
/gate/actor/Detector/addFilter particleFilter
/gate/actor/Detector/particleFilter/addParticle gamma
```

The units for the fluence is in cm^{-2} and for energy fluence it is MeV cm^{-2}. Uncertainties are relative to the mean value.

4.2 TLFluenceDistributionActor

To use the *TLFluenceDistributionActor* simply add it to a geometric object and enable the features that you want. An example of how this is accomplished looks like this:

```
# Create the fluence distribution detector
/gate/actor/addActor TLFluenceDistributionActor Detector
/gate/actor/Detector/save output/detector_data.root
/gate/actor/Detector/attachTo balldetector
/gate/actor/Detector/enableThetaHistogram true
/gate/actor/Detector/setThetaMin 0 deg
/gate/actor/Detector/setThetaMax 180 deg
/gate/actor/Detector/setNThetaBins 100
/gate/actor/Detector/enablePhiHistogram true
/gate/actor/Detector/setPhiMin -180 deg
/gate/actor/Detector/setPhiMax 180 deg
/gate/actor/Detector/setNPhiBins 100
/gate/actor/Detector/enableEnergyHistogram true
/gate/actor/Detector/setEnergyMin 0 MeV
/gate/actor/Detector/setEnergyMax 1.3 MeV
/gate/actor/Detector/setNEnergyBins 100
/gate/actor/Detector/setAsciiFile output/detector_data.dat
```

The first line names the actor *Detector*. The second line tells where to store the output and how the files should be named. The third line attaches the detector to a geometric object by the name *balldetector*. Line 4-15 enables and formats various histograms to be stored in the root file. Finally, on the last line an ascii-file storing all events is created. This file can take up quite a lot of space but can be useful if some specialized kind of plot is needed.

5 Validation

Examples of how to use the *TLFluenceActor* and *TLFluenceDistributionActor* are available and can be used for becoming acquainted with the actor. These example was also used for validation of the code. To run the analysis in these examples you need to have python installed with required packages. See the `main.mac` files for more information about required packages.

5.1 TLFluenceActor

Figure 1 shows the setup. A homogeneous beam of 4×4 cm^2 hits a $2 \times 2 \times 2$ cm^3 detector which is tilted relative to the beam. Attached to the detector region is a *TLFluenceActor* with $10 \times 10 \times 1$ pixels. All materials were set to vacuum to ensure no scattering and hence a known fluence.

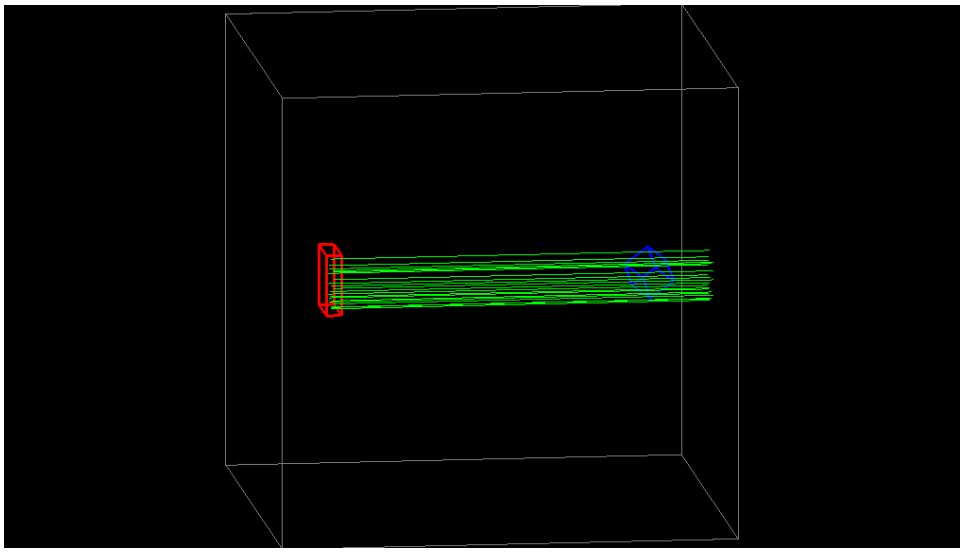

Figure 1: The simulation geometry. A square and uniform beam of photons are emitted from a plane source at the red box. Submerged in the beam of photons is box, tilted 45° which corresponds to the detector.

From the number of photons and the area of the source, the expected fluence can be calculated. In each voxel standard deviation can be calculated using standard MC methods. Since the detector contains 100 voxels an empirical standard deviation can also be found. The results are shown in Fig. 2. And the results from the Monte Carlo calculations agrees well with the expected results.

5.2 TLFluenceDistributionActor

Figure 3 shows the setup used. Two sources are placed with 90° relative angle irradiating the ball shaped structure with 4 discrete energies. A *TLFluenceDistributionActor* is attached to the ball. All materials are set to vacuum to ensure no scattering and hence a known fluence.

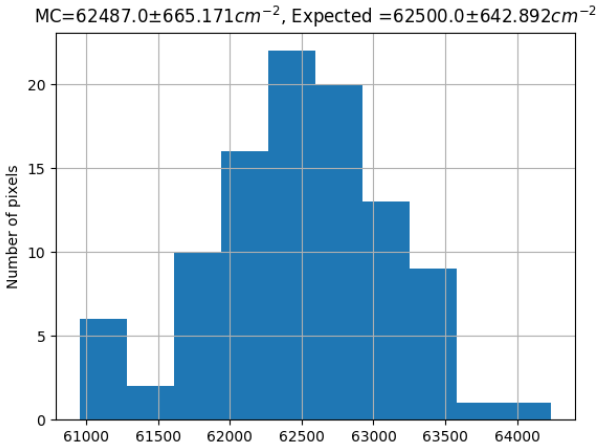

Figure 2: Histogram showing the number of voxels in the actor that counted a certain number of photons. Except from noise, the distribution is looking normal and centered around the value expected from the number of photons and the area of the beam. The standard deviation also agrees between the value calculated in the Monte Carlo code and an empirical value obtained from all 100 voxels in the actor.

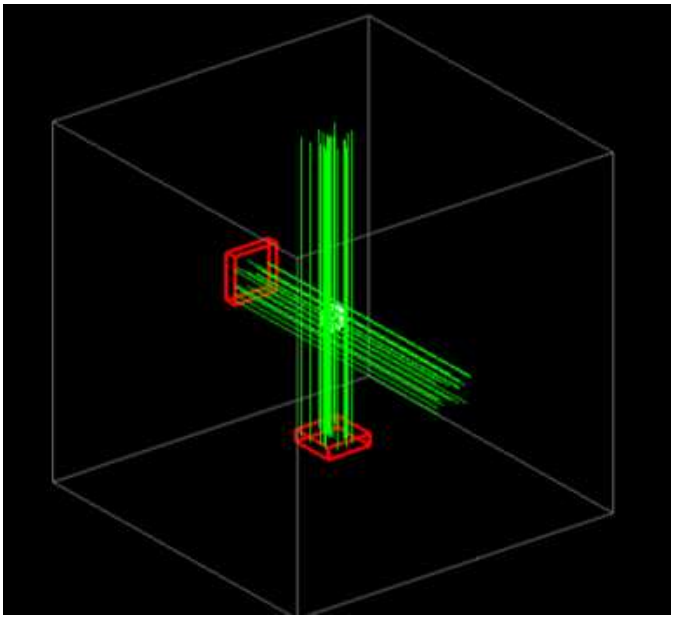

Figure 3: The simulation geometry. Two square-shaped and uniform beams of photons are emitted from plane sources located at the red boxes. Submerged in the beam of photons is a ball shaped detector to which a *TLFluenceDistributionActor* is attached.

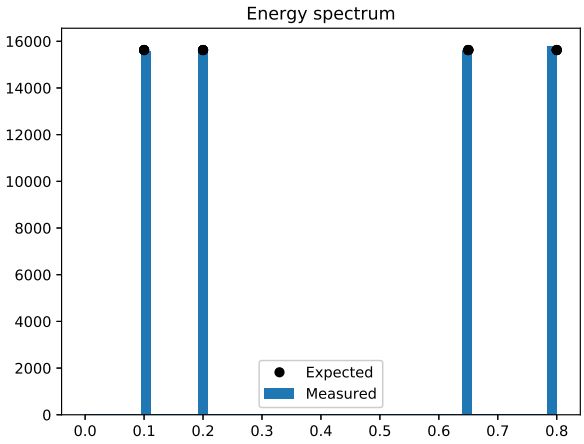

Figure 4: Detected energy spectrum with superimposed expected results, showing good agreement.

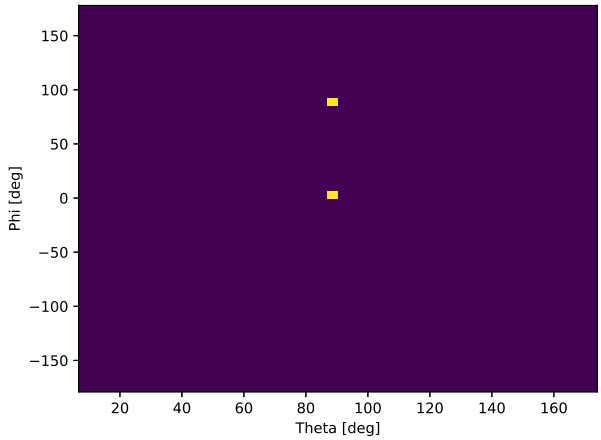

Figure 5: Detected angular distribution of the fluence at the detector. Two beams with a relative angle of 90° is expected and also observed.

6 Discussion

Two new actors for measuring fluence based on track-length has been implemented.

The *TLFLuenceActor* enables registration of spatially resolved particle fluence. It utilizes a Monte Carlo based method to assign the correct amount of track length and hence fluence to each voxel in the region of the actor.

The *TLFluenceDistributionActor* enables registration of angular and energy distribution of fluence. The results can be stored in both root-file format and as a ascii-file for further processing.

Some validations using a simple geometries have been performed showing that the actor behave as anticipated. However, more validation should be performed. In particular, the correctness of the actor when variance reduction techniques are used has not been validated.

To learn more on how to use the actors run the example codes.

References

[SD15] F. Salvat and Nuclear Energy Agency. Data-Bank. *PENELOPE-2014: A Code System for Monte Carlo Simulation of Electron and Photon Transport; Workshop Proceedings, Barcelona, Spain, 29 June*

- *3 July 2015*. OECD, 2015. URL: https://books.google.se/books?id=6tDKjgEACAAJ.