University College London
Department of Medical Physics and Biomedical Engineering

UCL

High energy physics simulations using GEANT4 and GATE

Maria Laura Pérez-Lara, MSc.

PhD student at University College London

V Uniandes Particle Detector School

December, 2021

Universidad de los Andes
Facultad de Ciencias

Overview

1- Introduction
- Monte Carlo methods
- MC in High Energy Physics
- Fundamental concepts

2- Geant4 basics
- Geometry
- Physics lists
- Primary generator
- User actions
- Scoring/output

3- The step-by-step on GATE
- The steps to build a simulation
- Visualization

4- Examples

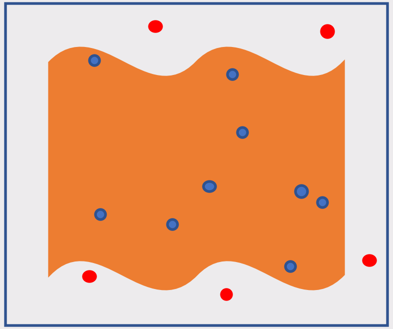
5- Summary

Monte Carlo method

- Stochastic method of numerical integration
- Based on **random number generation**
- An alternative to complex calculations or a large number of experiments
- Example: what is the area of this figure?

How do I find the area of this thing *analytically*?

Probability distribution



Shoot randomly at a position in the square (known area)

Find ratio between the bullets that hit the figure and the ones who didn't

Multiply the area of the square by this ratio

Monte Carlo in HEP

Problems

- Too complex to be analytical – Amount of particles in a single scenario
- Experimental limitations – Time, money, availability

Simulations

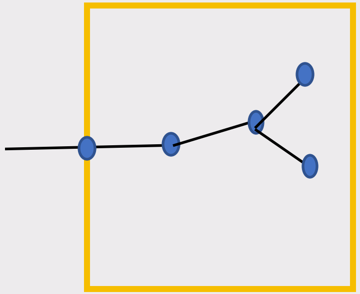
- Predict the outcome if the initial conditions and physical probabilities are given
- Randomly sample probability density functions, and the result is the average of all observations

Outcomes

- Generate simulated events and interaction cross sections (e.g. collisions)
- Generate particle interactions with matter (e.g. detector response)

Fundamental concepts

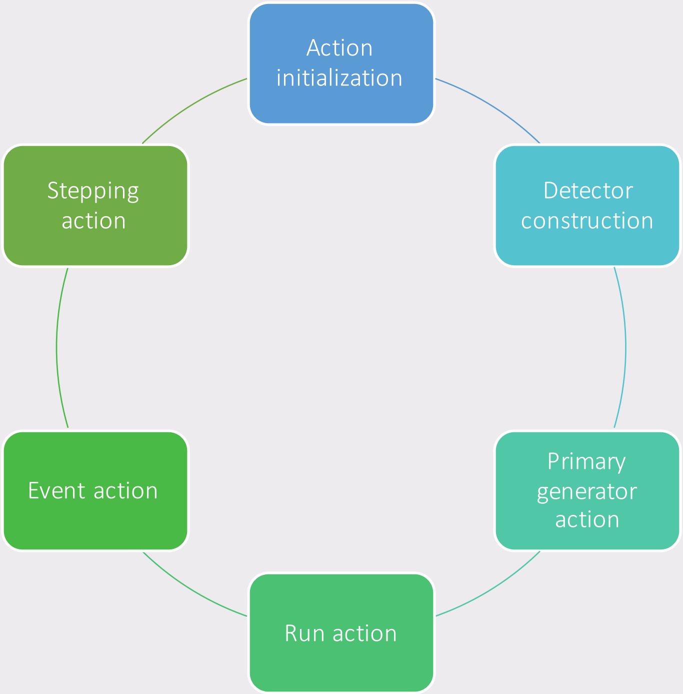
- **Step =** delta information with two points
- **Track =** snapshot of a particle (gets updated at every step)
- **Event =** basic unit of simulation, set of tracks
- **Run =** set of events
- **Hit =** snapshot of an interaction within a sensitive region of the detector (e.g. pos, t, p, E)
- **Single =** total information from a set of hits within a defined volume (e.g. a pixel)

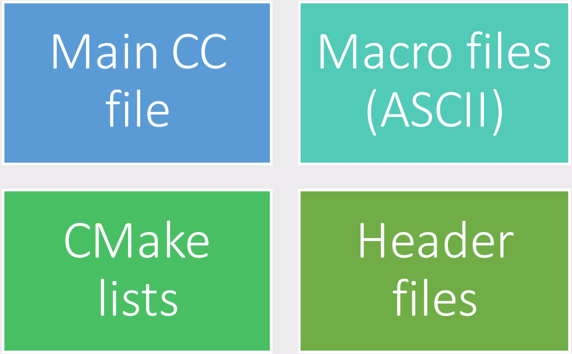


Geant4: what is it?

- It is a **simulation toolkit** created at CERN that enables you to create applications and tools
- Worldwide collaboration of physicists and software engineers
- Based on C++ programming (object-oriented)
- Included aspects:
  - Geometry
  - Materials
  - Particles
  - Generation of primaries
  - Tracking of particles through materials and fields
  - Physical processes
  - Response of detector components
  - Generation of event data
  - Storage of event and track information
  - Visualization of geometry and trajectories

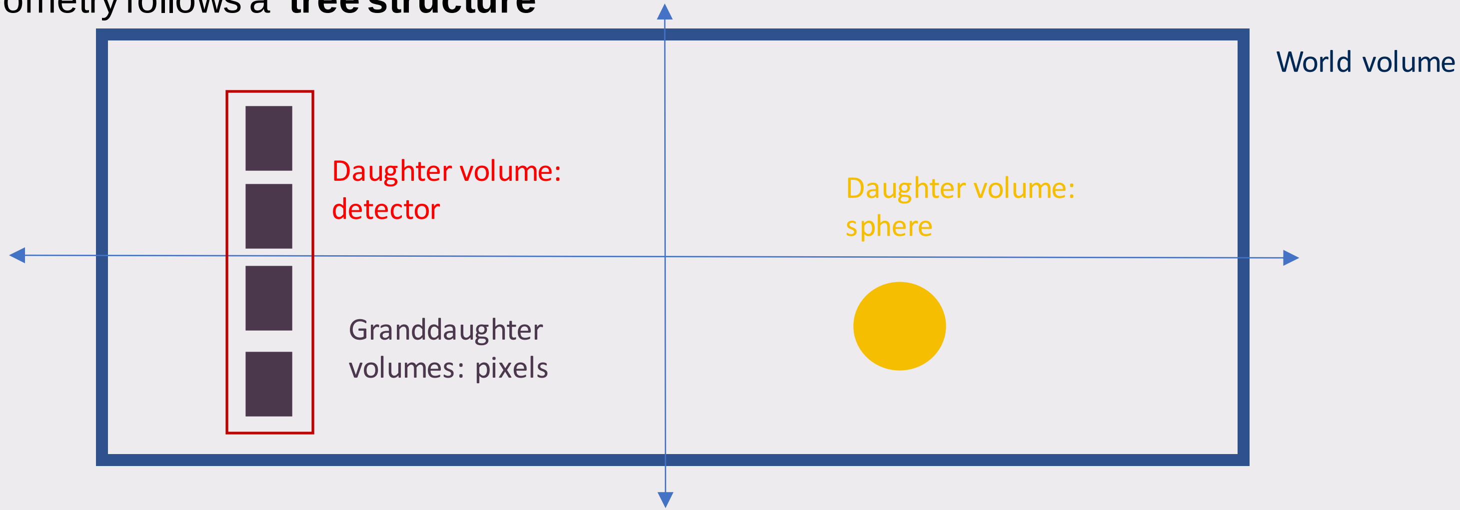
The structure of a simple G4 simulation





Geometry definition

- 3 layers for each volume: **Solid** (shape, size)**, logical** (geometrical hierarchy, material) **and physical** (placement, rotation, repetitions)
- Start with the **'world volume'** (which defines the global coordinate system)
- The geometry follows a **'tree structure'**



Materials

Two ways to define materials:

- "Materials are made of elements, elements are made of isotopes"

```
a = 112.414 * g / mole;
G4Element *elCd  = new G4Element(name = "Cadmium", symbol = "Cd" , z = 48., a);
a = 65.38 * g / mole;
G4Element *elZn  = new G4Element(name = "Zinc"   , symbol = "Zn" , z = 30., a);
a = 127.6 * g / mole;
G4Element *elTe  = new G4Element(name = "Tellurium"  , symbol = "Te" , z = 52., a);
G4Material *CZT = new G4Material(name = "CZT", density = 5.8 * g / cm3, ncomponents = 3);
CZT->AddElementByNumberOfAtoms(elCd, 1);
    CZT->AddElementByNumberOfAtoms(elZn, 1);
    CZT->AddElementByNumberOfAtoms(elTe, 1);
```

- Import materials from a database (e.g. NIST)

```
G4NistManager *nistManager = G4NistManager::Instance();
nistManager->FindOrBuildMaterial("G4_CADMIUM_TELLURIDE");
```

The physics behind G4

- Use **physics lists –** Specify particles and their physical processes
- This helps in case we want to optimize the simulation (how much accuracy/speed do we need?)
- What can be covered:
- Electromagnetic (standard, low energy)
- Weak interactions (decays)
- Hadronic physics (strong, nuclear interactions, neutron high precision)
- Pre-packaged lists (e.g. **QGSP_BIC_HP_EMZ**)

QGSP: Quark gluon string, precompound de-excitation model

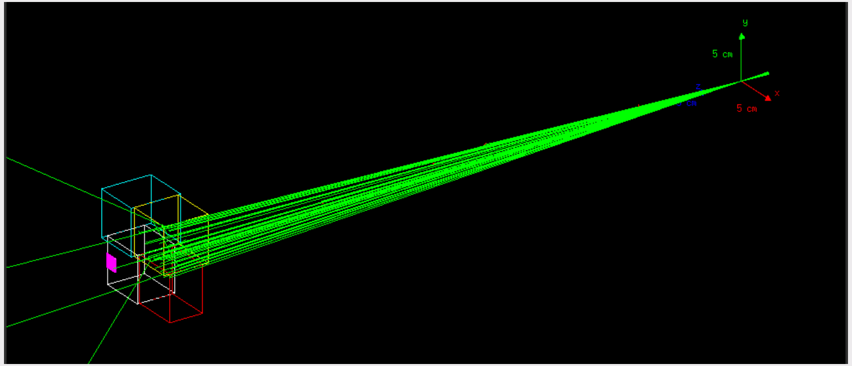
BIC: Binary cascade

HP: Neutron high precision model

EMZ: Electromagnetic, option 4 (most accurate)

Primary generation: the particle source

- **Primary particles** are created with a nature defined by the user (energy, type, direction, angular distribution)
- **Primary vertices** have the information of position and time
- There are several implementations (e.g. general particle source, particle gun)
- The shape, distribution, spectrum of the source can be defined
- Radioactive sources can also be implemented



User actions

- They are optional classes in G4
- Fully customizable
- Useful to extract information about the run
- Tell the simulation what to do at the beginning and end of a run/event/step/etc

- User action initialization
- User primary generator action
- Run action
- Event action
- Tracking action
- Stepping action
- Stacking action

Scoring

Extract information that is useful to you (e.g. energy, dose deposition, secondary particles)

User hooks	Sensitive detector and Hits	Primitive scorer
• Write your own code to get what you need after every step/event • Full access to all information	• Write own code for extra classes (SD and hit files) • Assign SD to a logical volume • Process hits by analyzing hit collections per event	• Common physical quantities available • Define scoring mesh • Many scorers in a volume

GATE: what is it?

- It stands for **Geant4 Application for Tomographic Emission**
- Scripting is done via command language – No need for C++
- Encapsulates G4 libraries to make it easier for the user to perform simulations in the field of medical physics

GATE vs G4: simple detector on G4 (1/2)

```
#include "DetectorConstruction.hh"
#include "G4Material.hh"
#include "G4NistManager.hh"
#include "G4Box.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
#include "G4GeometryManager.hh"
#include "G4PhysicalVolumeStore.hh"
#include "G4LogicalVolumeStore.hh"
#include "G4SolidStore.hh"
#include "G4PhysicalConstants.hh"
#include "G4SystemOfUnits.hh"
#include "G4SDManager.hh"
```

```
DetectorConstruction::DetectorConstruction()
: G4VUserDetectorConstruction(),
   fCheckOverlaps(true)
{}

DetectorConstruction::~DetectorConstruction()
{}

G4VPhysicalVolume* DetectorConstruction::Construct()
{
  DefineMaterials();

  return DefineVolumes();
}
```

GATE vs G4: simple detector on G4 (2/2)

```
void DetectorConstruction::DefineMaterials()
{
  NistManager
->FindOrBuildMaterial("G4_CADMIUM_TELLURIDE");
}

G4VPhysicalVolume* DetectorConstruction::
DefineVolumes()
{
  auto detector_material =
G4Material::GetMaterial("G4_CADMIUM_TELLURIDE");
  auto worldSolid = new G4Box("World", 100/2 * mm,
100/2 *mm, 100/2 *mm);

auto worldLogical = new G4LogicalVolume(worldSolid
,  defaultMaterial, "World");
```

```
auto worldPhysical = new
G4PVPlacement(0, G4ThreeVector(), "World",
0, false, 0, fCheckOverlaps);

auto hexSolid = new G4Box("HEX", /2,
5/2 *mm, hexThickness/2);

auto hexLogical =
new G4LogicalVolume(hexSolid, hexMaterial,
"HEX");
  Auto hexPhys = new
G4PVPlacement(0, G4ThreeVector(0, 0,
10 *cm), hexLogical, "HEX", worldLogical,
false, 0, fCheckOverlaps);

  return worldPhysical;
}
```

GATE vs G4: simple detector on GATE

```
/gate/world/geometry/setXLength 10. cm
/gate/world/geometry/setYLength 10. cm
/gate/world/geometry/setZLength 10. cm
/gate/world/setMaterial Air

/gate/world/daughters/name detector
/gate/world/daughters/insert box
/gate/detector/placement/setTranslation 0. 0. 10. cm
/gate/detector/geometry/setXLength 5 mm
/gate/detector/geometry/setYLength 5 mm
/gate/detector/geometry/setZLength 1. mm
/gate/pixel/setMaterial CdTe
```

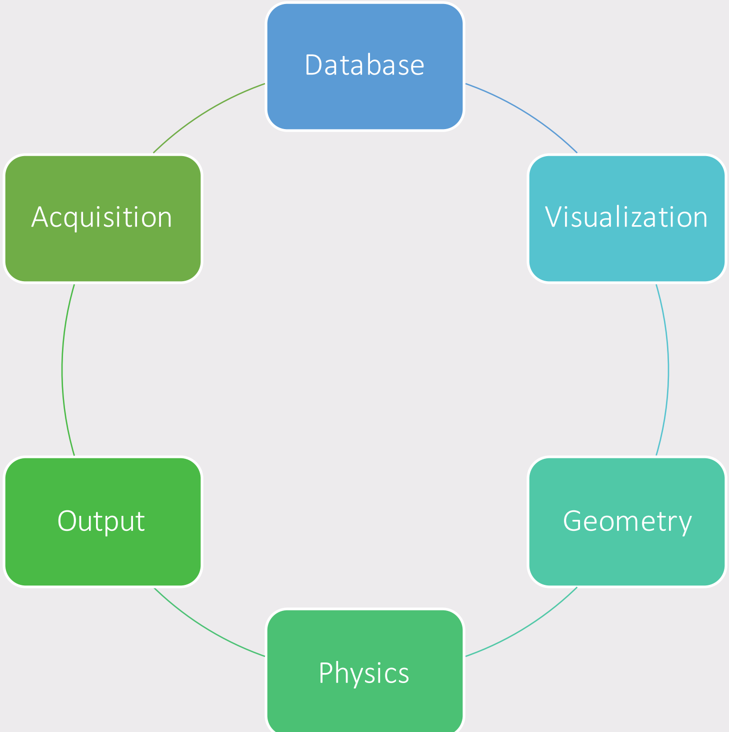
Check structure of the command line:
Directory, command and parameters

G4 vs GATE: Sensitive detector

```
void DetectorConstruction::ConstructSDandField()

{

    DetectorSD *sensorSD;

    sensorSD = dynamic_cast<DetectorSD
*>(G4SDManager::GetSDMpointer()-
>FindSensitiveDetector("SensorSD", false));


    if (!sensorSD) {

        sensorSD = new DetectorSD("SensorSD",
"SensorHitsCollection", nPixel * nPixel);

    }

      G4SDManager::GetSDMpointer()-
>AddNewDetector(sensorSD);

    SetSensitiveDetector("HEX", sensorSD);

}
```

```
/gate/detector/attachCrystalSD
```

Structure of a GATE simulation



All set up files are macros, but we might need txt and db files. The 'main' macro calls all other necessary macro files:

```
/control/execute mymacro.mac
```

Step 0: Add a database

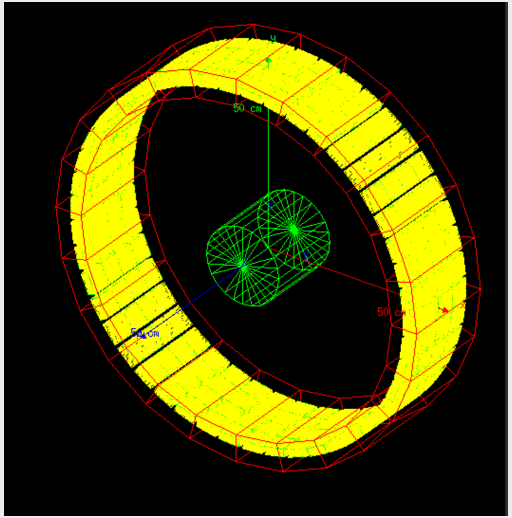
```
/gate/geometry/setMaterialDatabase MyMaterialDatabase.db
```

This is very important to define geometries!

Step 1: Choose the architecture

Yellow: crystals
Green: phantom
Red: PET element

- Application-dependent:
- If your application is imaging, choose a system to define the tree structure of the geometry
- If your application is dosimetry or radiotherapy, there is no system needed
- Some systems:
- Scanner – subdivided in levels, geometry is not fixed
- CTscanner – subdivided in module, cluster and pixel
- CylindricalPET – subdivided in rsector, module, submodule, crystal and layer
- SPECThead – subdivided in crystal and pixel



```
/gate/world/daughters/name SystemName
```

```
systems/SystemName/Level/attach UserVolumeName
```

Step 2: Build the geometry

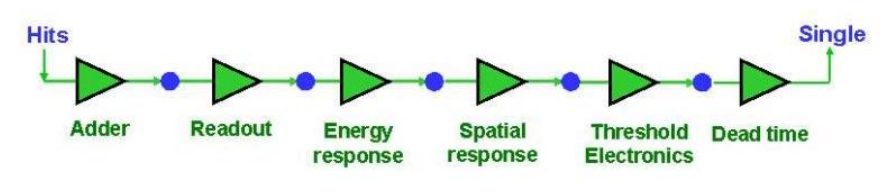
- Every volume must be a daughter of the world volume
- Set the parameters in the commands

```
/gate/world/daughters/name waterCylinder
/gate/world/daughters/insert cylinder
/gate/waterCylinder/geometry/setRmin 0. mm
/gate/waterCylinder/geometry/setRmax 8. mm
/gate/waterCylinder/geometry/setHeight 20. mm
/gate/waterCylinder/placement/setRotationAxis 1 0 0
/gate/waterCylinder/placement/setRotationAngle 90. deg
/gate/waterCylinder/setMaterial Water
/gate/waterCylinder/vis/forceWireframe
/gate/waterCylinder/vis/setColor cyan
```

→ Introduce the volume (name)
→ What is its shape? (insert)
→ What are its dimensions? (setRmin, setRmax, setHeight)
→ How about rotating it? (setRotationAxis, setRotationAngle)
→ What is it made of? (use database) (setMaterial)
→ What about making it pretty? (forceWireframe, setColor)

Step 3: Set your detector

- Detectors are defined like any other volume
- Respect the hierarchy of the system
- Make the detector SD and/or add actors
- Add a **digitizer** to simulate detector response




```
/gate/digitizer/Singles/insert adder
/gate/digitizer/Singles/insert readout
/gate/digitizer/Singles/readout/setDepth 3
/gate/digitizer/Singles/insert thresholder
/gate/digitizer/Singles/thresholder/setThreshold 5 keV
```

- `insert adder` → Insert module: addition of hits in a single detector element
- `setDepth 3` → Do the addition in a pixel? In a cluster?
- `setThreshold 5 keV` → Remove events below threshold

Step 4: Set up the physical processes

- Either add them manually or use pre-packaged lists
- Set production cuts

```
/gate/physics/addProcess PhotoElectric
/gate/physics/processes/PhotoElectric/setModel
StandardModel
/gate/physics/processList Enabled
/gate/physics/processList Initialized
```

```
/gate/physics/addPhysicsList emstandard_opt4
```

QGSP? BERT? BIC? HP? EM?

```
/gate/physics/Gamma/SetCutInRegion      phantom 0.25 mm
/gate/physics/Electron/SetCutInRegion   phantom 0.25 mm
/gate/physics/Positron/SetCutInRegion   phantom 0.25 mm
```

Step 5: Initialize

After the initialization, the geometry can no longer be changed

```
/gate/run/initialize
```

Step 6: Generate the source

```
/gate/source/addSource mybeam           gps
/gate/source/mybeam/gps/particle        gamma
/gate/source/mybeam/gps/ene/type        Mono
/gate/source/mybeam/gps/ene/mono       30. keV
/gate/source/mybeam/gps/pos/centre      0 0 -10 mm
/gate/source/mybeam/gps/pos/type        Plane
/gate/source/mybeam/gps/pos/shape       Circle
/gate/source/mybeam/gps/pos/radius      2 mm
/gate/source/mybeam/gps/direction       0 0 -1
/gate/source/mybeam/gps/ang/type        focused
/gate/source/mybeam/gps/ang/focuspoint 0 0 0 mm
/gate/source/mybeam/setActivity           5. becquerel
```

- gps → Add it and name it – what type of source is it?
- gamma → What is the primary particle?
- Mono / 30. keV → Monoenergetic (which energy?) or user spectrum (file?)
- 0 0 -10 mm → Where are primaries created?
- Plane / Circle / 2 mm → What is the distribution? Beam, plane, surface, point? How does it look like?
- 0 0 -1 → What is the primary momentum unitary vector?
- focused → What is the angular distribution? Isotropic, focused?
- 5. becquerel → How many particles per second?

Step 7: Set the global output

- According to the architecture, you will have different **global output files** available (e.g. ROOT, ASCII)
- The output commands should always go after initialization

```
/gate/output/ascii(**binary**)/enable
/gate/output/ascii/setFileName   test
/gate/output/ascii(**binary**)/setOutFileHitsFlag   1
/gate/output/ascii(**binary**)/setOutFileSinglesFlag   1
/gate/output/ascii(**binary**)/setOutFileCoincidencesFlag   1
/gate/output/ascii(**binary**)/setOutFileSingles_digitizerModule_Flag   1
```

```
/gate/output/root/enable
/gate/output/root/setFileName test
/gate/output/root/setRootHitFlag            0
/gate/output/root/setRootSinglesFlag        1
/gate/output/root/setRootNtupleFlag         0
/gate/output/root/setRootCoincidencesFlag   0
```

Step 8: Add actors

- In case you want a way to interact with the simulation, use actor commands
- Implementation of G4's user hooks
- There are plenty of options!

```
/gate/actor/addActor ActorType ActorName
/gate/actor/ActorName/attachTo  VolumeName
/gate/actor/ActorName/save FileName
```

Dose
- DoseActor
- DoseByRegions

Phase space
- PhaseSpaceActor

Statistics
- SimulationStatisticsActor
- ParticleInVolumeActor

Secondaries
- SecondaryProductionActor

Step 9: Set acquisition details

```
/gate/random/setEngineName MersenneTwister
/gate/random/setEngineSeed auto
/gate/application/setTotalNumberOfPrimaries 1000
OR
/gate/application/setTimeSlice     1. s
/gate/application/setTimeStart     0.  s
/gate/application/setTimeStop      10. s
/gate/application/startDAQ
```

→ Pick your random generator (Ranlux64, James Random or Mersenne Twister) and seed

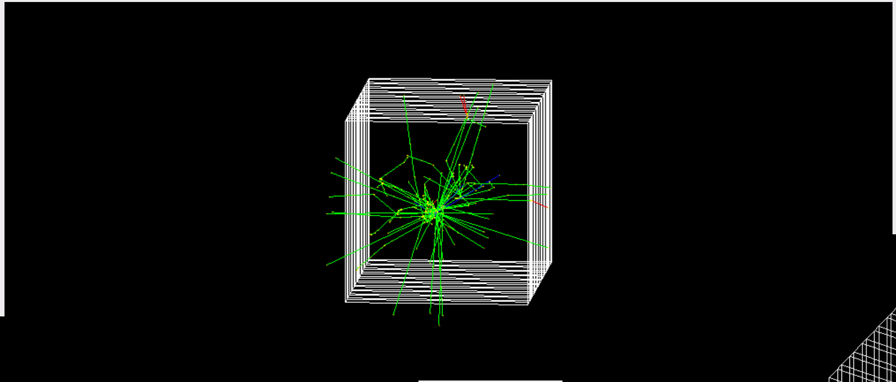
→ Simulate a number of particles or by time?

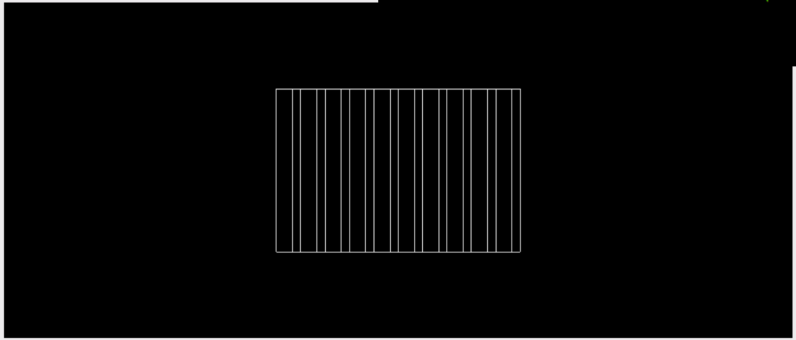
→ Run!

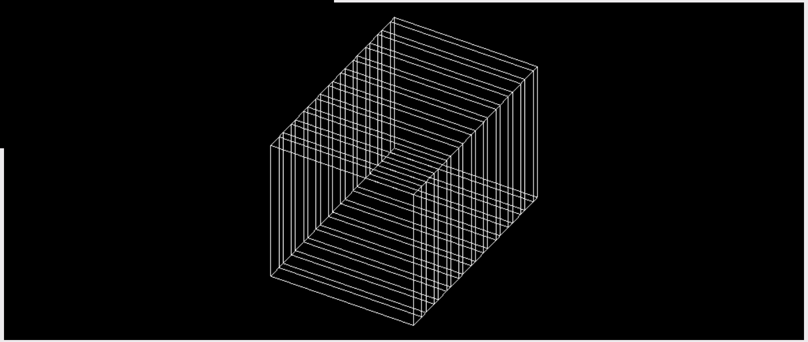
Step 10: How to visualize your geometry?

Create a view with a determined viewer →	`/vis/open OGL 600x600-0+0`
Start drawing →	`/vis/drawVolume`
Change the angle of vision →	`/vis/viewer/set/viewpointThetaPhi 90. 180.`
Zoom into the scene →	`/vis/viewer/zoom 2`
Add some coordinates →	`/vis/scene/add/axes 0 0 0 5 cm`
Take a look at the tracks →	`/vis/scene/add/trajectories smooth`
Take a look at the hits →	`/vis/scene/add/hits`
In case you only want to see certain particles, use filters →	`/vis/filtering/trajectories/create/particleFilter` `/vis/filtering/trajectories/particleFilter-0/add gamma`
Show all events in a run at once →	`/vis/scene/endOfEventAction accumulate`

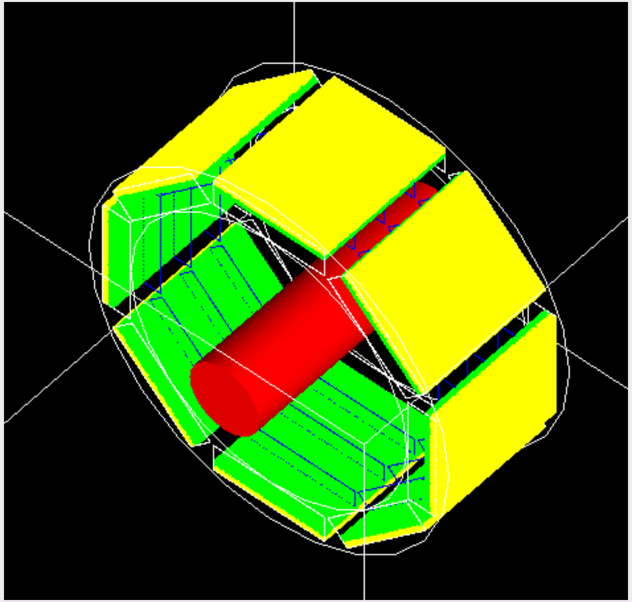
Software examples: a basic calorimeter

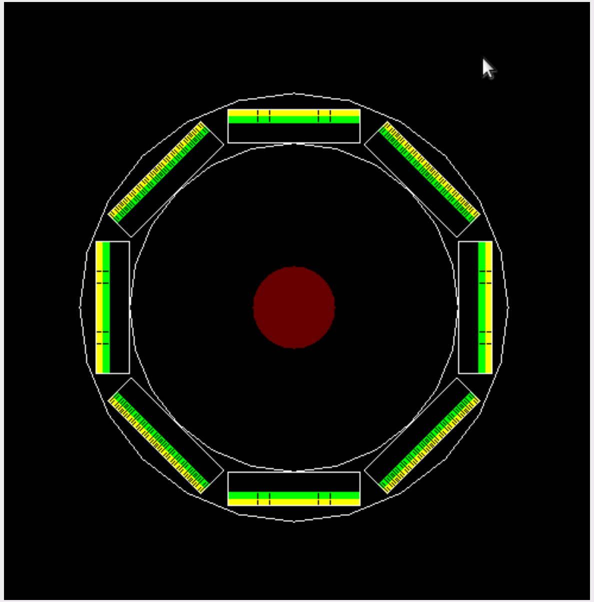




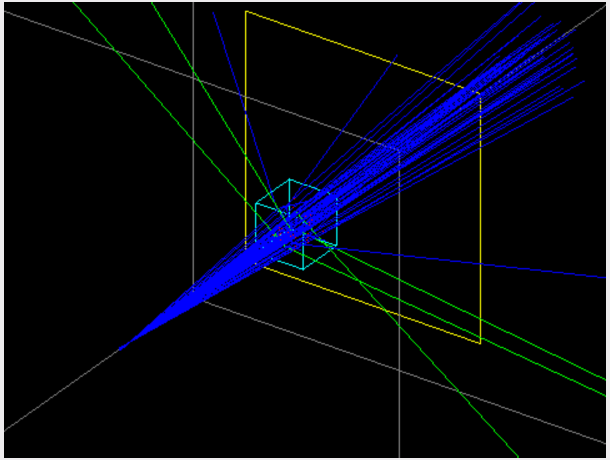


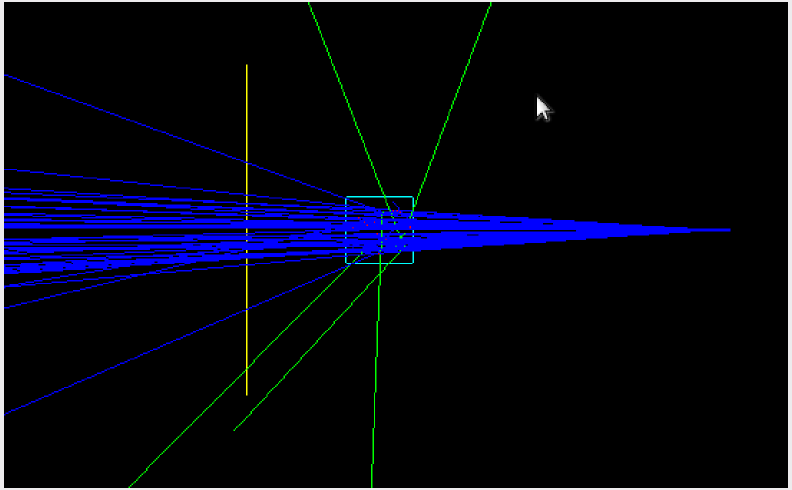
Software examples: Tracker





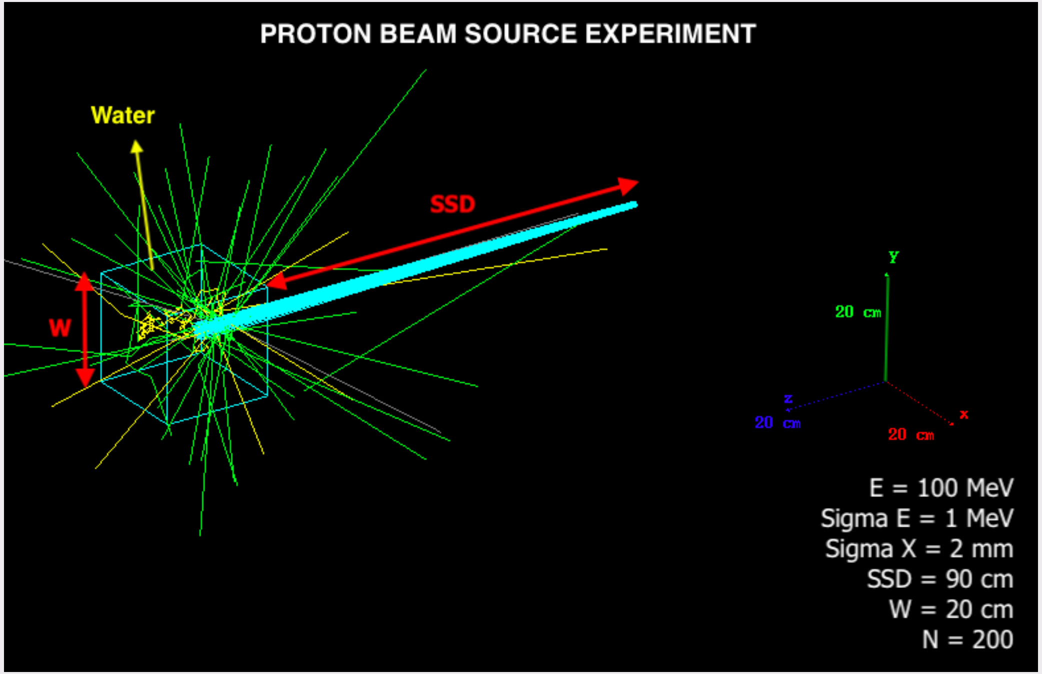
Software examples: Proton Radiography



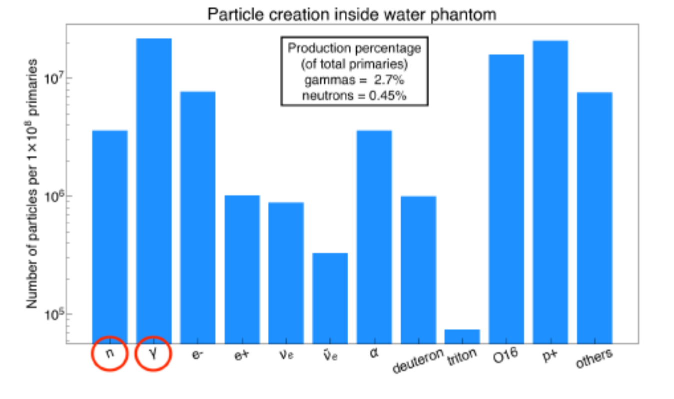


Own example: Proton beam therapy

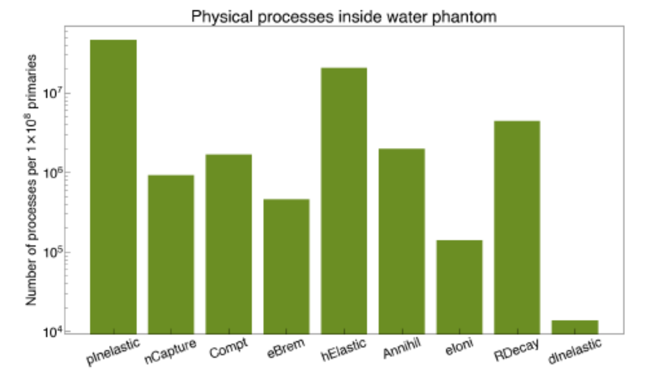
Focus on geometry



Own example: Proton beam therapy



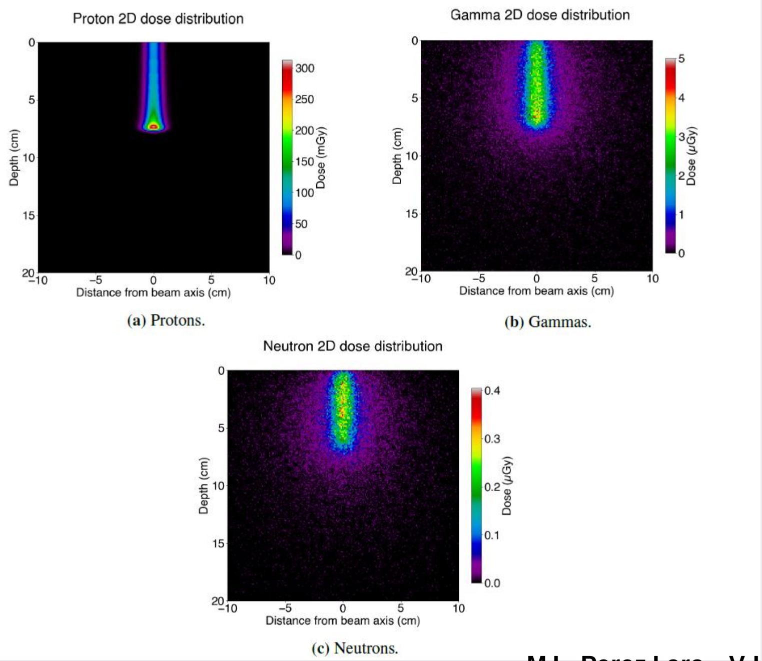
(a) Secondary particles.



(b) Processes.

PhaseSpaceActor: particle creation and processes

Own example: Proton beam therapy



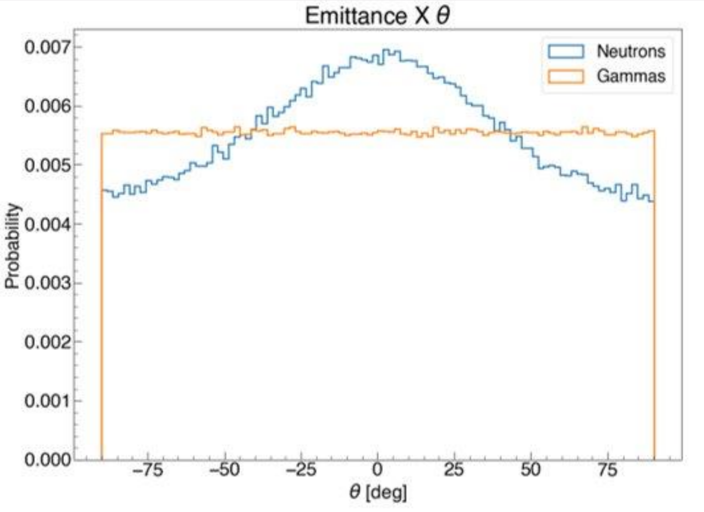
(a) Protons.

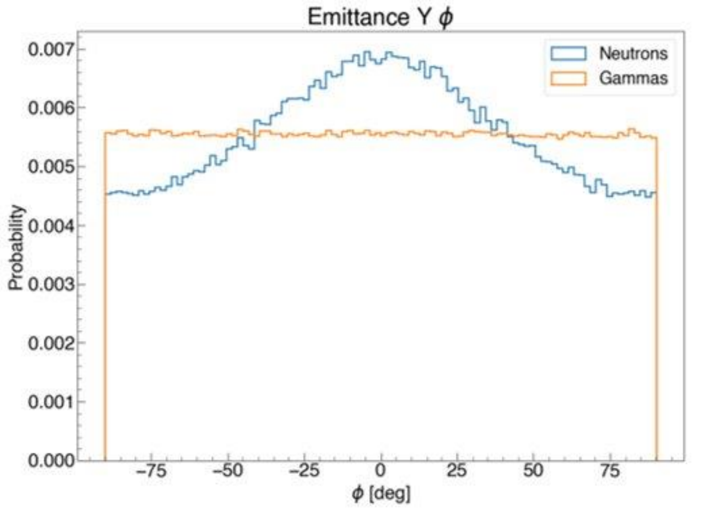
(b) Gammas.

(c) Neutrons.

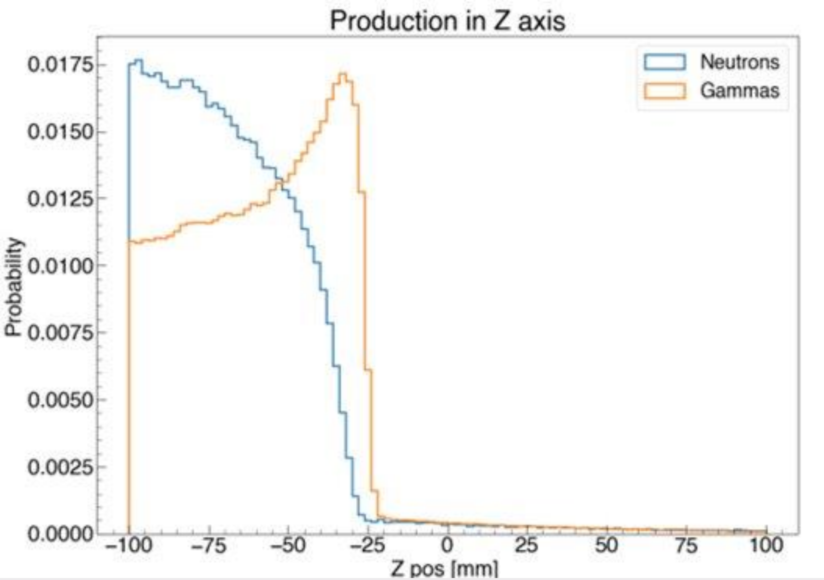
DoseActor with filters for each particle

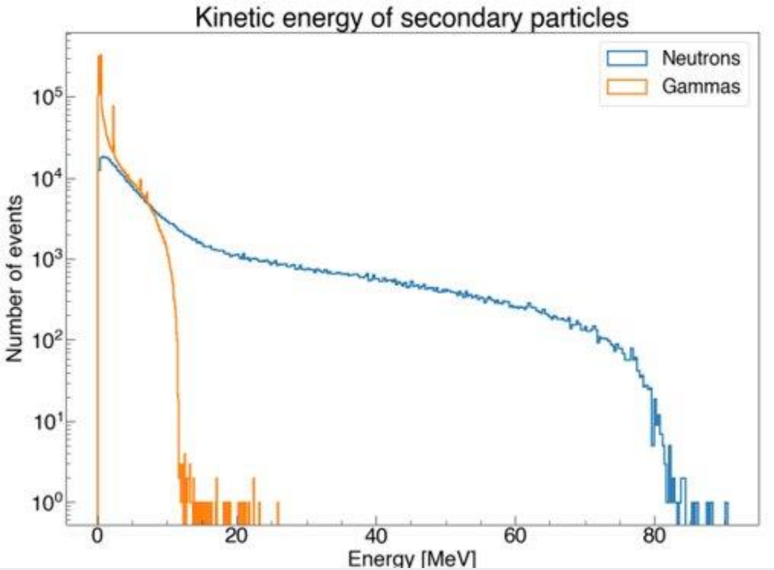
Own example: Proton beam therapy



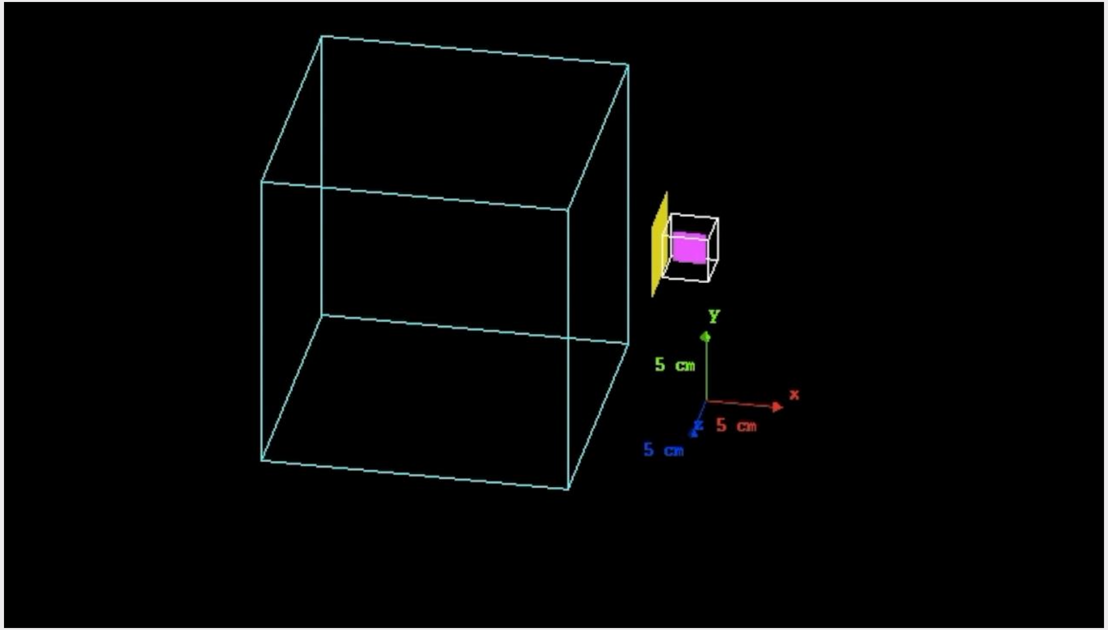


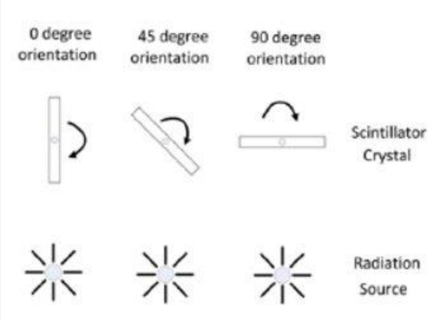
Own example: Proton beam therapy

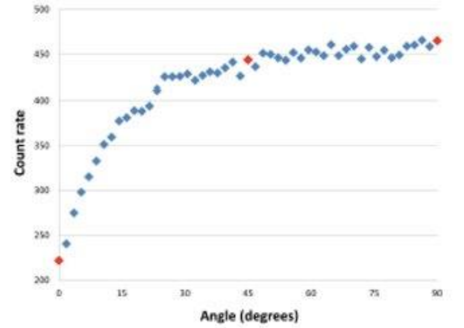




Own example: rotating detector



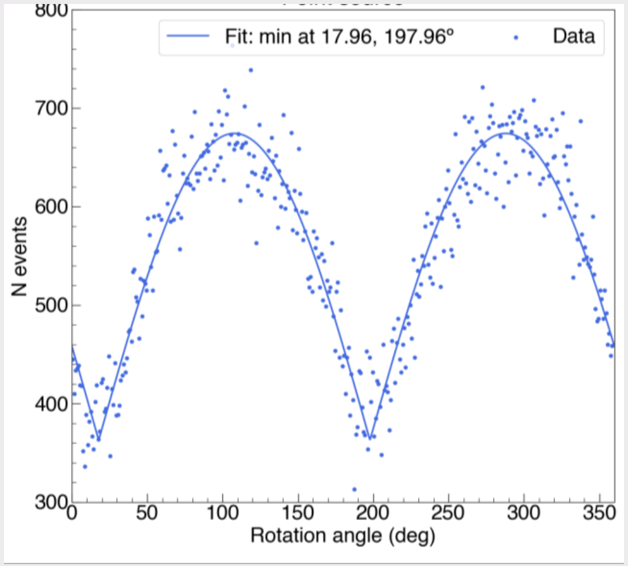




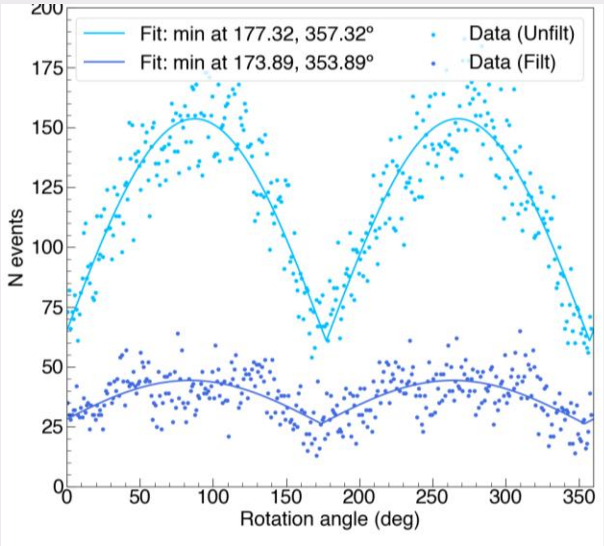
Randall et al. 2014 J Inst 9 P10011

Own example: rotating detector

Gamma source curve



Neutron source curve



Use of global output

Summary

Monte Carlo-based simulations are a clever way to approach HEP problems by sampling probability density functions and providing initial conditions

Geant4 is a simulation toolkit that allows full freedom to model accurate geometries, interactions and detector responses via C++ programming

GATE is a dedicated scripting mechanism that extends the native command interpreter of Geant4, consisting of a set of 10 steps that are useful for medical physics applications, from a simple calorimeter to a complex proton beam therapy model

Thank you!
Any questions?