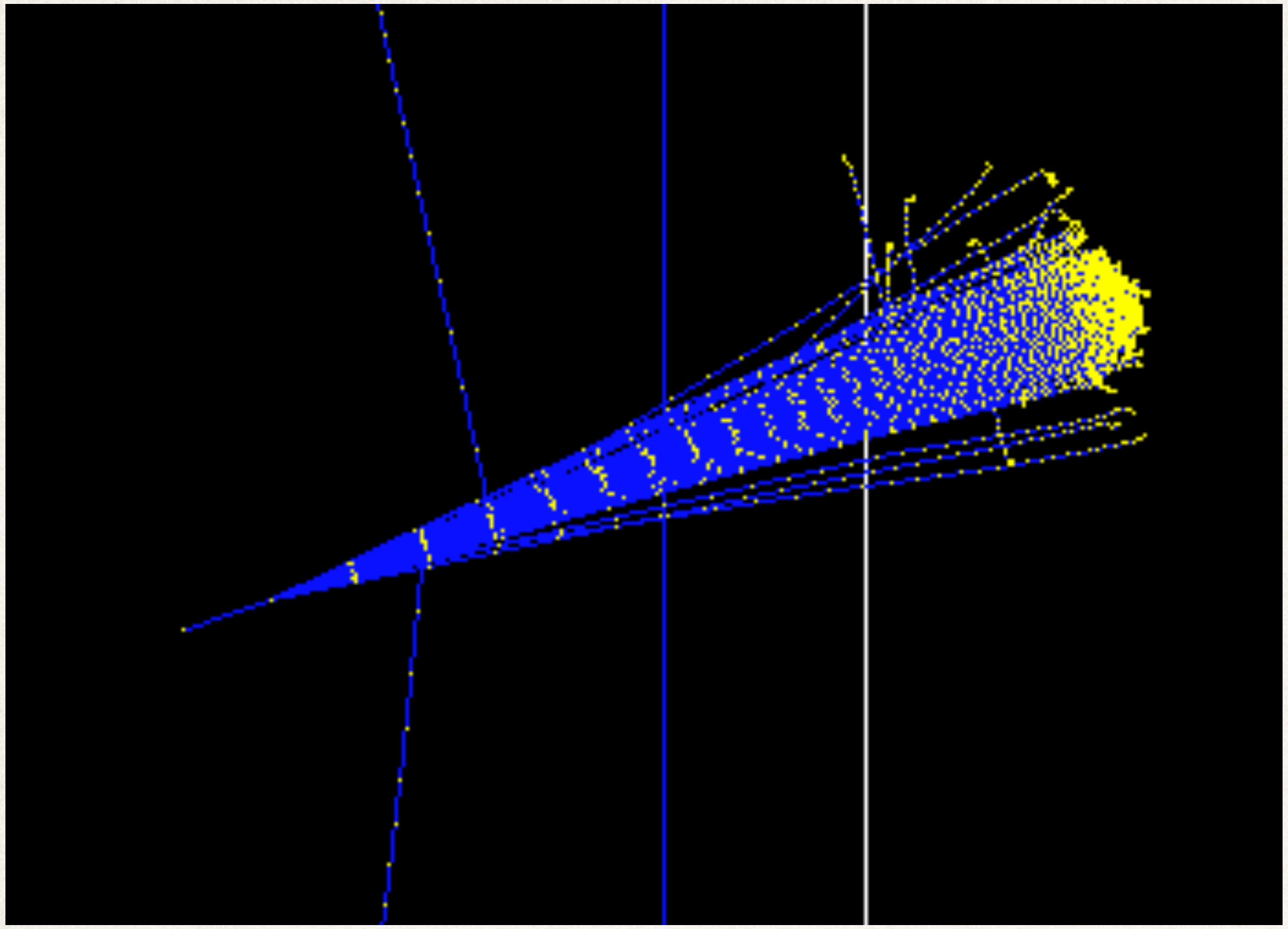
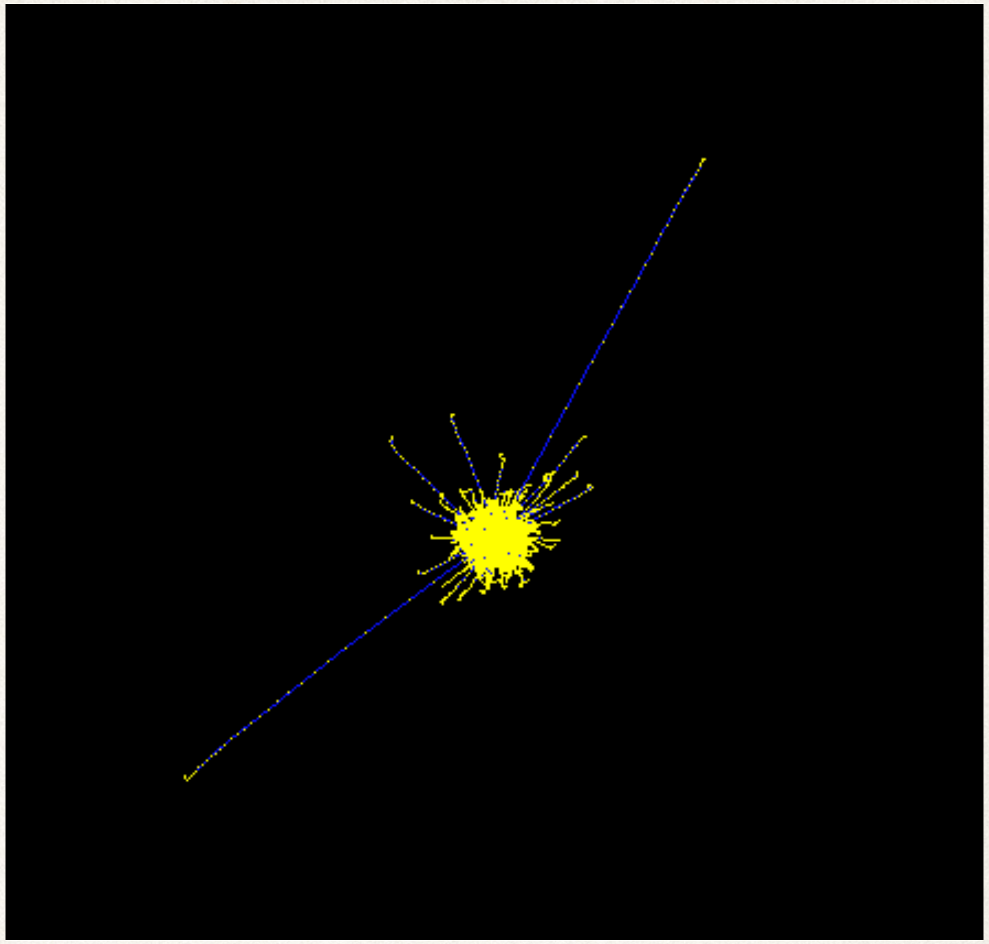



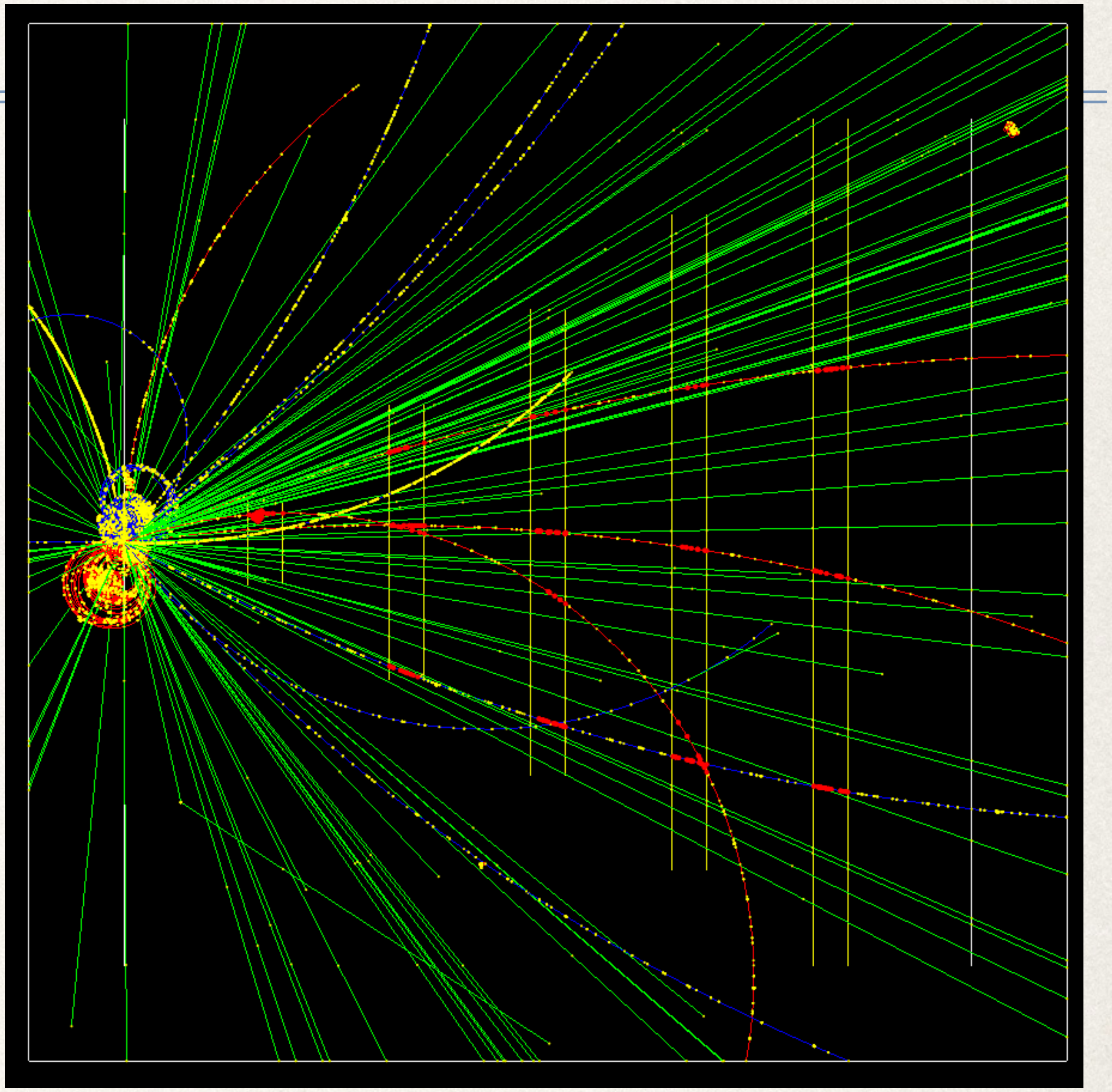
Lesson 3

Particles and Physical Processes
Optional (but very useful) Classes

03/10/2023

Particle tracking slang

- Each particle is associated with a **track**
  - **green**: neutral particles
  - **red**: negatively charged particles
  - **blue**: positively charged particles
- Tracks are divided in **steps**
- A new step every time the particle:
  - crosses a border
  - has a physical interaction
- The point joining two steps is a **hit**



- **Secondary particles** resulting from physical processes are treated in the same way as primaries
- An **event** is thus composed of many tracks, one for each particle

Mandatory classes

- To have a working simulation, the user must develop:
  - main() ✓
  - DetectorConstruction() - geometry / materials ✓
  - PrimaryGenerator() - primary particles ✓
  - **PhysicsList() - define what physics to include** ←

PhysicsList

1. Define which particles to track
   - other particles may be created during interactions, but if they have no associated process they will just leave the geometry without interacting

2. Attach physics processes to each particle
   - each process may have one or more models that the user can choose from (or even use simultaneously, for different energy intervals)

3. Set (production) cuts
   - GEANT4 follows each particle until it stops or leaves the geometry. The notion of “cut” is connected to the production of secondary particles *(more on this later)*

PhysicsList — a word of caution

- This is a simulation, not the real world!
- So we can pick and choose the physics to use.
- <u>This is useful but also dangerous!</u>
  - Allows us to study particular physics processes and their effect in detail
  - Forgetting to include all the required physics processes will lead to wrong/incomplete results
  - To make things more difficult, processes often have multiple models

Declaring particles

- This must be done inside *ConstructParticle()*
- GEANT4 already includes classes for most particles
- Particle properties already included in these classes
- Uses a standard naming convention, making them very easy to use

Declaring particles

- Some examples:
  - *G4Gamma::GammaDefinition()*
  - *G4Positron::PositronDefinition()*
  - *G4NeutrinoE::NeutrinoEDefinition()*
  - *G4PionMinus::PionMinusDefinition()*
  - *G4AntiNeutron::AntiNeutronDefinition()*
  - *G4GenericIon::GenericIonDefinition()*

Declaring particles

- There are also methods to declare all the particles of a given type
    - #include “G4BosonConstructor.hh"
      G4BosonConstructor consBos; consBos.ConstructParticle();
    - #include "G4LeptonConstructor.hh"
      G4LeptonConstructor consLep; consLep.ConstructParticle();

Declaring physical processes

- Must be done in the *ConstructProcess()* method
- Transportation is a fundamental process in GEANT4, otherwise particles will not be tracked in the geometry (!)
  - *AddTransportation()*
- GEANT4 already includes processes for most interactions
- Some have more than one model available
- If needed the user may create a new process
  (don't worry, we will not be doing this!)

Declaring physical processes

- (Simplified) example for gammas:

```
#include "G4ComptonScattering.hh"
#include "G4GammaConversion.hh"
#include "G4PhotoElectricEffect.hh"
if (particleName == "gamma") {
    // gamma
    pmanager->AddDiscreteProcess(new G4PhotoElectricEffect);
    pmanager->AddDiscreteProcess(new G4ComptonScattering);
    pmanager->AddDiscreteProcess(new G4GammaConversion);
}
```

Declaring physical processes

- Radioactive decay example:

```
// Add Decay Process
G4Decay* theDecayProcess = new G4Decay();
theParticleIterator->reset();
while( (*theParticleIterator)() ){
  G4ParticleDefinition* particle = theParticleIterator->value();
  G4ProcessManager* pmanager = particle->GetProcessManager();
  if (theDecayProcess->IsApplicable(*particle)) {
    pmanager ->AddProcess(theDecayProcess);
}}
```

Declaring physical processes

- Example of a process with more than one model:
  - Changeover between models at 19 MeV

```
// neutron
pmanager = G4Neutron::NeutronDefinition()->GetProcessManager();
// elastic model
G4LElastic* neutronElasticModel1(new G4LElastic);
neutronElasticModel1->SetMinEnergy(19*MeV);
G4NeutronHPElastic* neutronElasticModel2(new G4NeutronHPElastic);
neutronElasticModel2->SetMaxEnergy(19.1*MeV);
// elastic process
G4HadronElasticProcess* neutronElasticProcess(new G4HadronElasticProcess());
neutronElasticProcess->AddDataSet(new G4NeutronHPElasticData);
neutronElasticProcess->RegisterMe(neutronElasticModel1);
neutronElasticProcess->RegisterMe(neutronElasticModel2);
pmanager->AddDiscreteProcess(neutronElasticProcess);
```

Declaring physical processes

- GEANT also offers classes with sets of physical processes for typical use cases. For example, for electromagnetic physics:

```
#include "G4EmStandardPhysics_option3.hh"

G4EmStandardPhysics_option3* emPhysicsList
                    = new G4EmStandardPhysics_option3();

emPhysicsList->ConstructProcess();
```

Several options available, with different models and accuracy/speed

- And also **full** physics lists for typical applications (high energy, low background experiments, medical physics , etc.).
  They already include:
  - electromagnetic physics
  - hadronic physics
  - optical processes
  - decay
  - …

These are easier (and safer!) to use, we will use them in most examples

Setting cuts

- By default, all the particles are tracked until they stop
  (or leave geometry boundaries)
- We can set cuts in the production of secondary particles
  - If the expected range of a secondary particle is smaller than the set cut…
  - … instead of creating a new particle, the energy is deposited locally
- We may set different cuts for different particles…
- … and/or different materials or volumes

Quick example on the effect of cuts

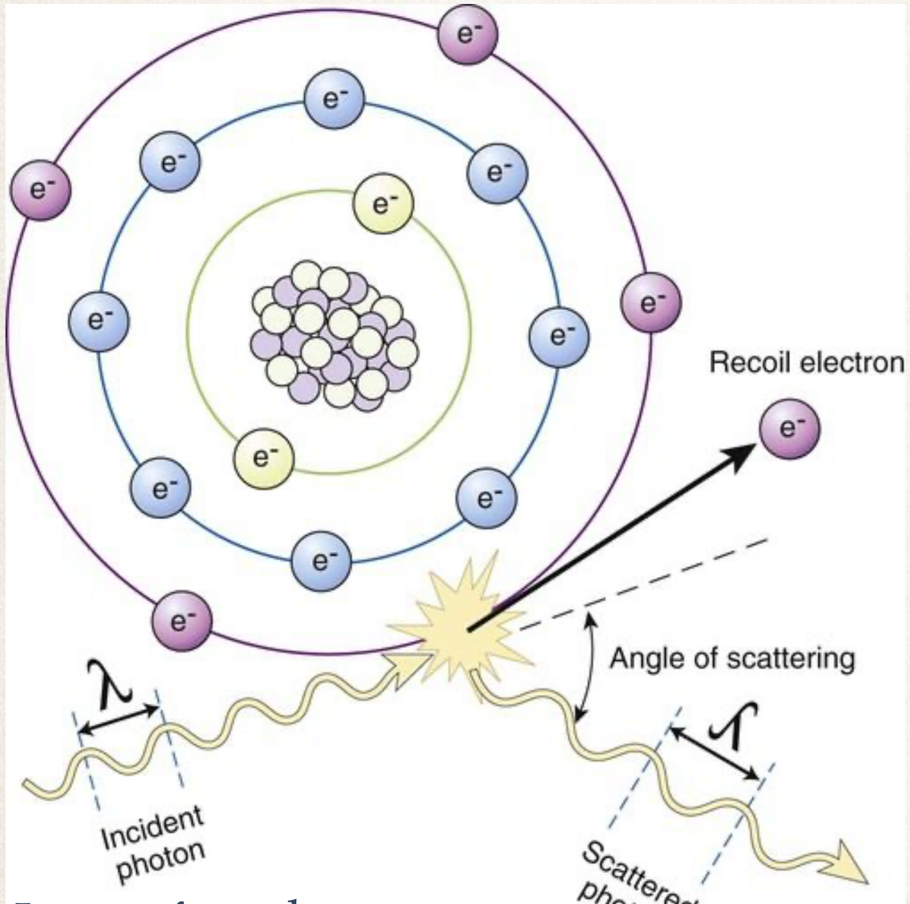


Image from here

- A gamma-ray (our primary particle) interacts with an electron from an outer shell via Compton scatter
- The recoiling electron is emitted from the atom
  - This will be a secondary particle
  - If this electron has an energy above the cut, it will be created and tracked in the simulation, depositing energy as it goes (possibly far away from the origin)
  - If the energy is lower, no secondary particle is produced, and the energy is deposited in the Compton scatter position
- Note that the cut is usually expressed in range (distance) and not energy
  - These cuts are usually associated with the position resolution of our detectors

PhysicsList

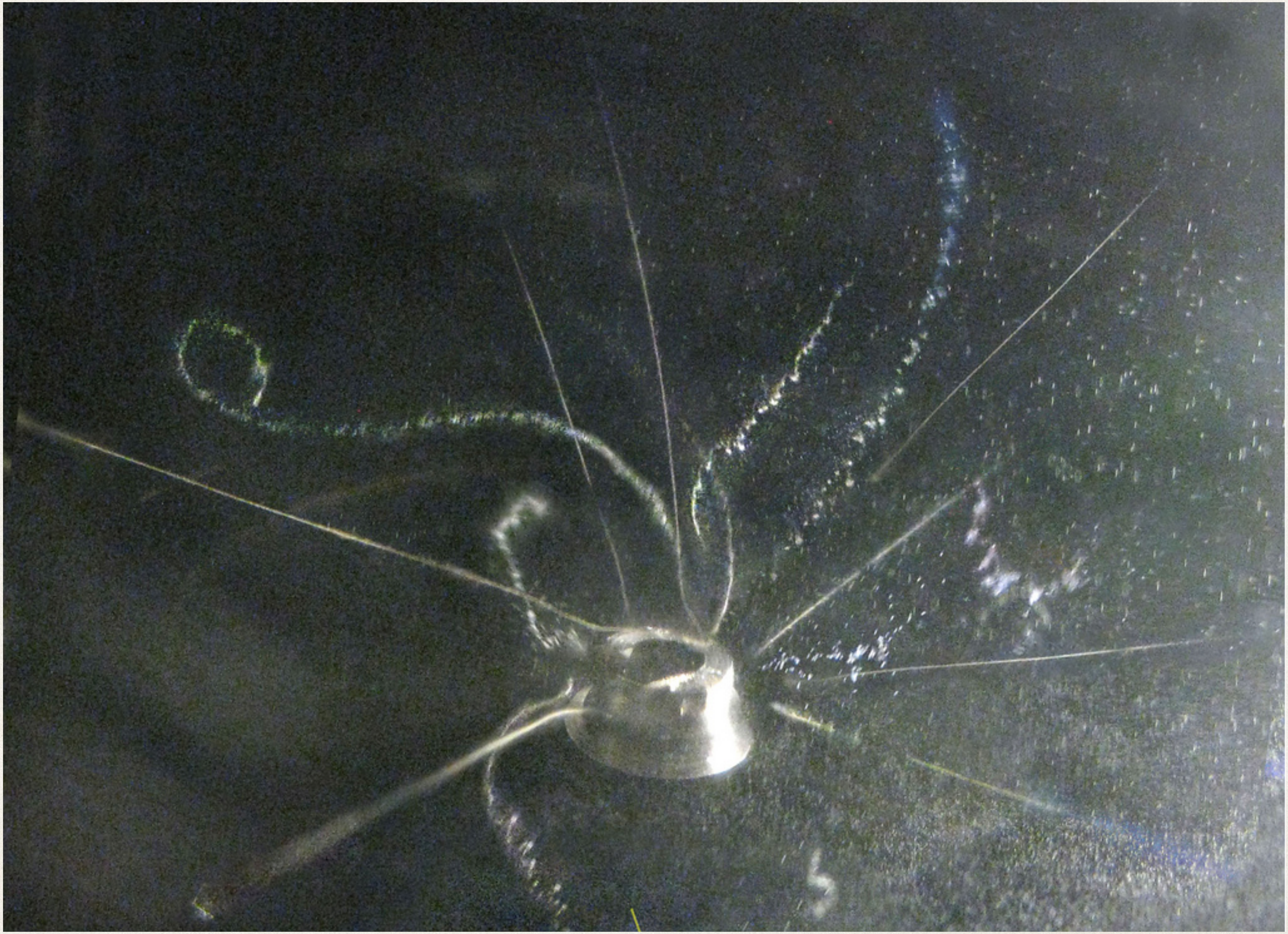
- Let’s see an example:
    - Open PhysicsList.cc
    - From the BraggPeak example
    - Download the zip and move the folder inside your geant4 folder

PhysicsList

- Let's see an example:
  - Open PhysicsList.cc
  - From the BraggPeak example
  - Download the zip and move the folder inside your geant4 folder

➡ This is just for you to get an idea of the structure of a PhysicsList
  - In future examples we will always use pre-defined PhysicsLists from GEANT4

Simulation of alpha particles in air



Run the BraggPeak simulation

- Check the code in DetectorConstruction and PrimaryGenerator
- Compile (type *make*) and run the simulation (type *BraggPeak*)
  - type *tracking/verbose 1* to get information about what is happening in each step
  - start an event by typing */run/beamOn 1*
  - zoom in to have a closer look at the alpha track
  - rotate the geometry to see the longitudinal and transverse profiles
  - scroll through the information printed out, check what happened in each step
  - Change the production cut (in PhysicsList) to 0.3 mm, re-compile and run it again (*what changed? remember to put it back to 0.3 cm*)

➡ We can easily get the range for individual events

- ➡ But we already know that single events are not a good representation of the response of the system
- ➡ How do we accumulate statistics and do proper analysis?

Optional classes

- By now, we can already create working simulations
- But it's not easy to get useful information from them
- There are several classes that the user can define to collect information and store it outside the simulation, so that it can be analysed later

Useful optional classes

These are the ones we'll be using in our classes

- StepppingAction()
- EventAction()
- RunAction()

EventAction

- Provides methods that are called by GEANT4 right before and right after each event
- Allows to perform analysis on data that was collected during each event
- **RunAction** works in a similar way, but at the level of sets of events

StepppingAction

- This class is called every time a particle has a step
- Independently of what particle is being tracked…
- … or the volume it is in
- Can make the simulation significantly slower
  - especially with complicated geometries
  - and events with many tracks
  - **this generally doesn't apply to us… ;)**

SteppingAction

- It's the easiest and most straight-forward way of collecting information!
- Let's go back to the BraggPeak example to see how it works
- Open file <u>SteppingAction.cc</u>
- We will use this class to get:
  - 1) the range of alpha particles in air
  - 2) the Bragg curve

Run & analysis instructions

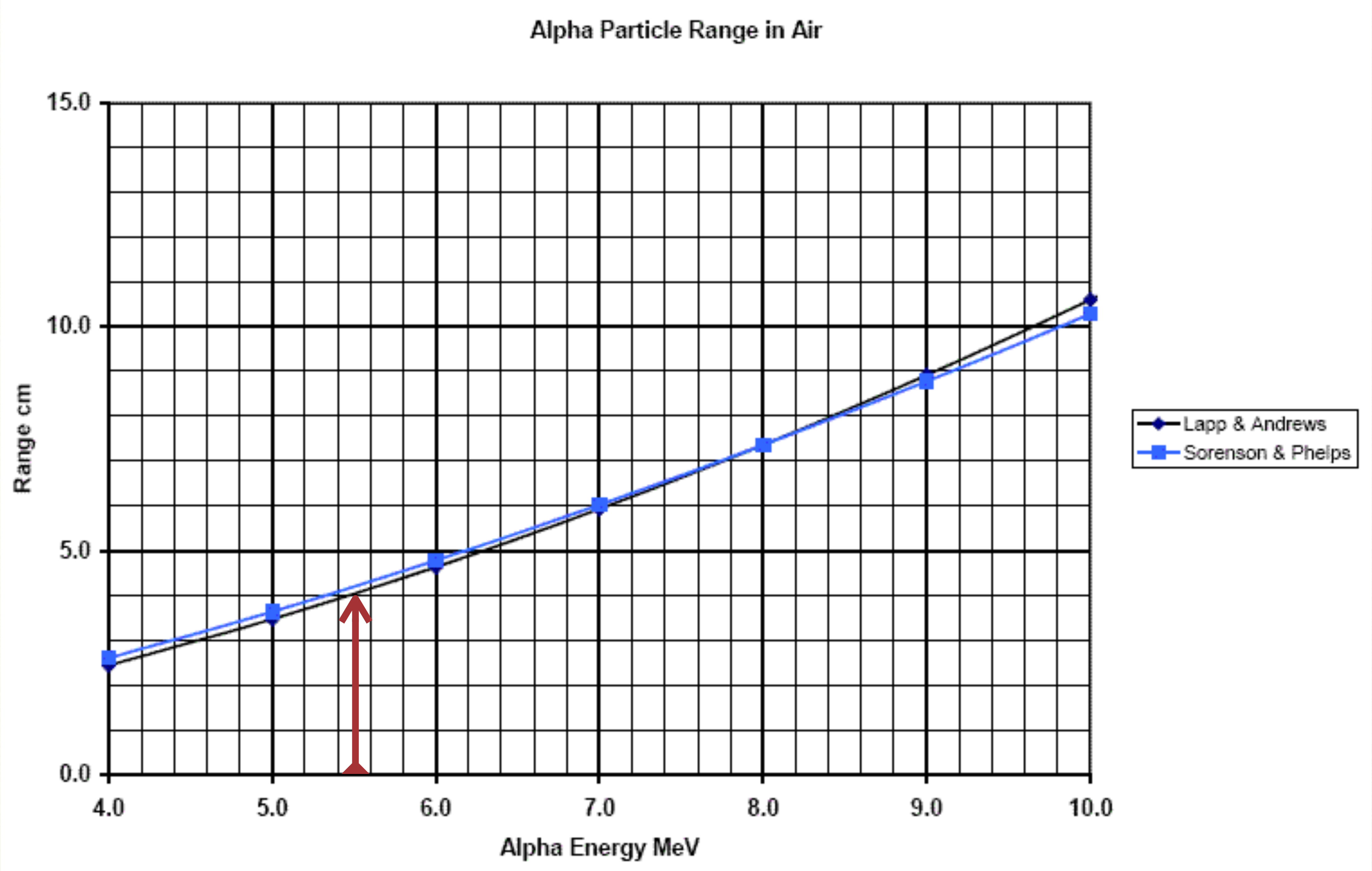
- In the SteppingAction file, check the code after line 23
  "// USE THIS METHOD FOR STEP #1 -- GETTING THE RANGE OF ALPHA PARTICLES"
    - We collect information about the volume the particle is in, and the particle itself
    - When the particle stops, we collect information about its position and write to an output file
- Compiling and running the code:
    - *make*
    - *BraggPeak batch_mode.mac*
      (note that this will not open the graphical interface, but will run 1000 events in batch mode)
- This will create an output file: *BraggPeak.out*
- Run the script alphaRange.C in ROOT:
    - *root -l*
    - *.x alphaRange.C*
    - *.q* to quit ROOT

A couple of other plots you can try:

test->Draw("radius");

test->Draw("track:radius","","colz");

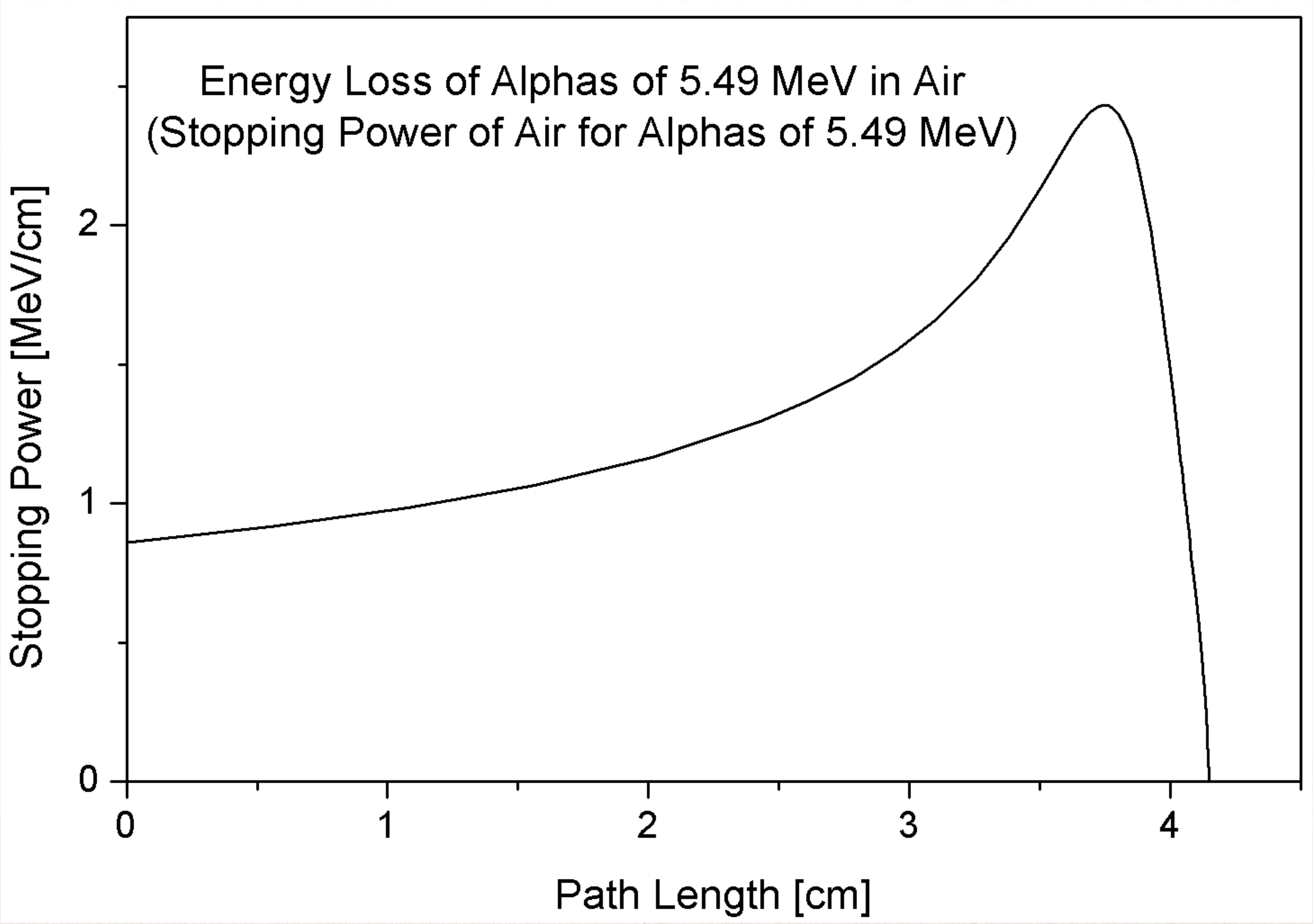
Range of 5.5 MeV alpha particles in air



Run & analysis instructions

- Comment the method which starts at line 23, and uncomment the one starting at line 45 "// USE THIS METHOD FOR STEP #2 -- PRODUCING THE BRAGG PEAK"
- This will save in the output file, **for each step:**
  - the total track length (**x**)
  - the energy deposited in the step (**dE**)
  - the size of the step (**dx**)
- Edit the file *batch_mode.mac* and reduce the number of events to 1000 (if needed)
- Compile and run the simulation
- Run the analysis scripts in ROOT (*bragg_peak*.C and *bragg_peak_advanced*.C)
- Compare the observed curve with what's expected, in the next slide

Bragg curve for 5.5 MeV alphas in air



Homework (actually class-work!)

That's it! We have done our first simulation!

The next one is for you to do "on your own"

- Start from the Shielding.zip simulation "skeleton"
- Consider a collimated beam of photons with 1 MeV
- What is the required thickness of Pb to reduce the intensity of the beam to 1/10th of its initial value?
- What is the mean energy of the photons that get through?
- What fraction of these photons suffered at least one Compton scatter?
- Repeat the exercise using copper as shielding material.