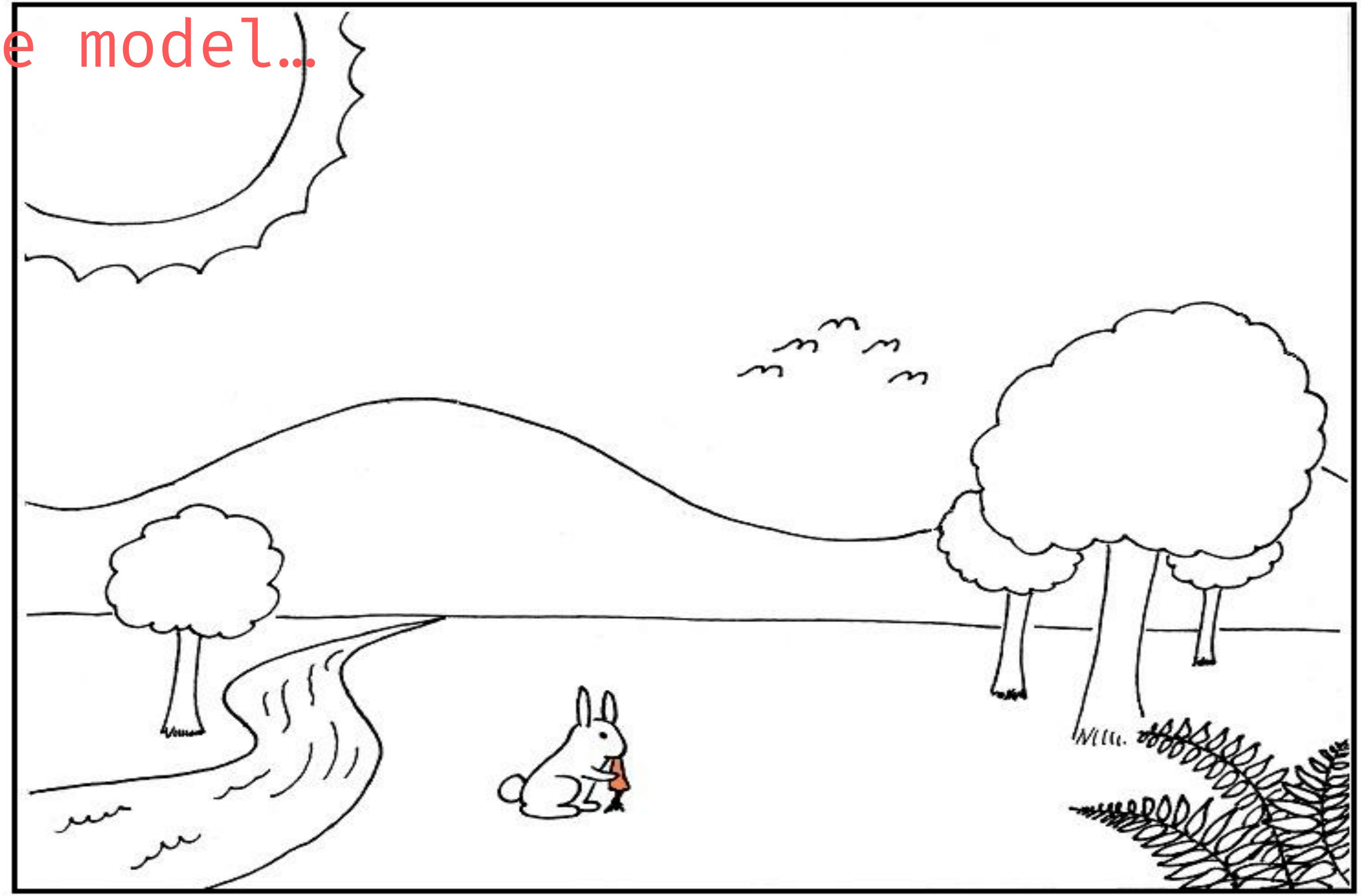
```
G4Course *MyNewCourse::Construct() {

    G4Course *course = new G4Course();
    course→title("Geant4 for Beginners. A crash course");
    course→author("Hernán Asorey");
    course→email("asoreyh@gmail.com");
    course→description("a hands-on Geant4 crash course");
    course→school("La Conga Physics");
    course→site("github.com/asoreyh/geant4-course");
    course→year(2023);
    course→duration(4*h);
    course→license("CC0 1.0 Universal");
    return course;
}
```

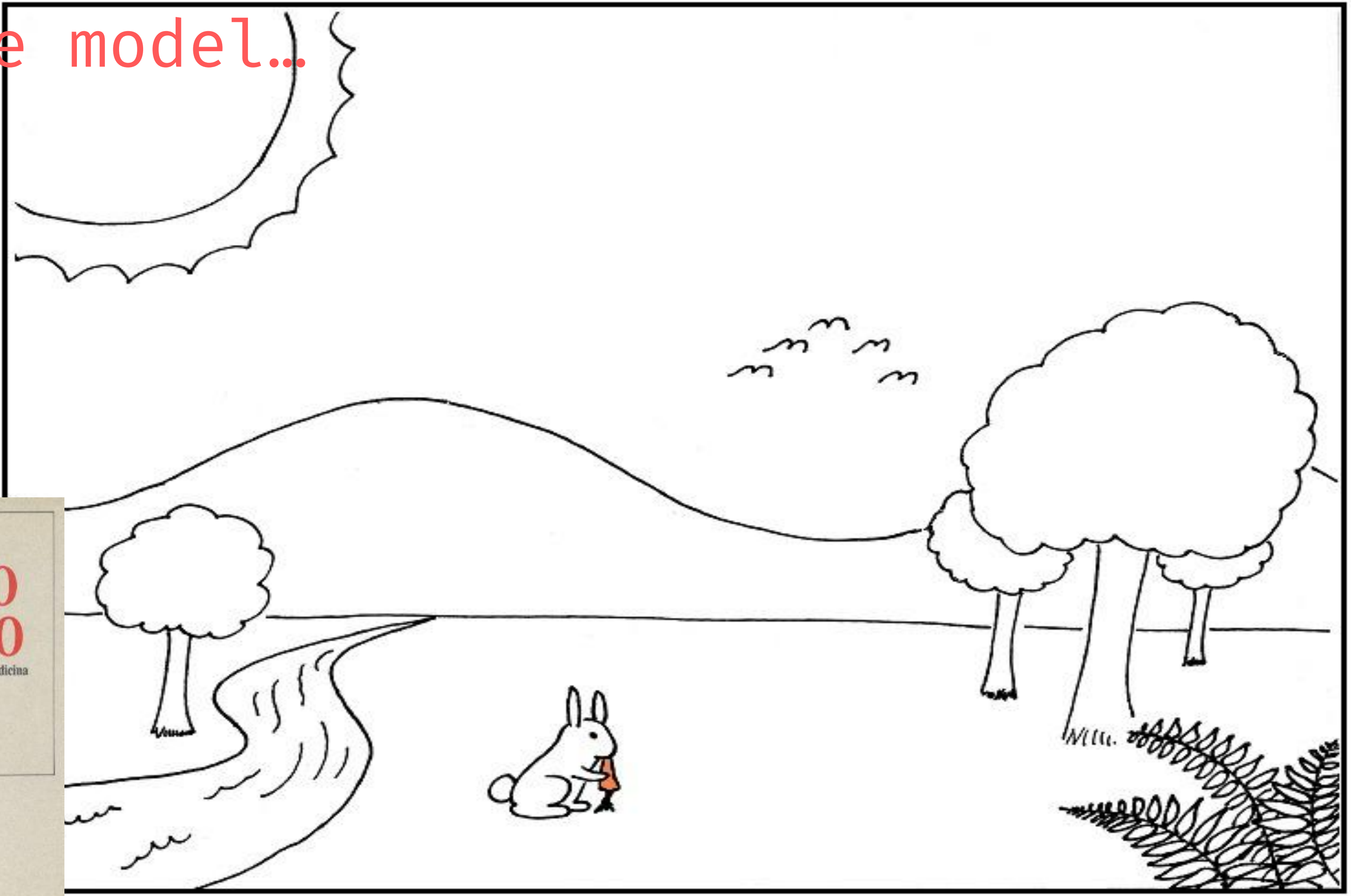
Disclaimers

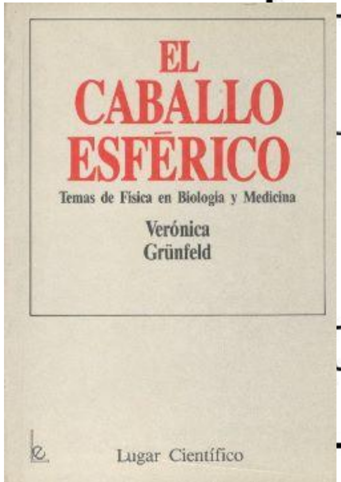
The reality…

The model…

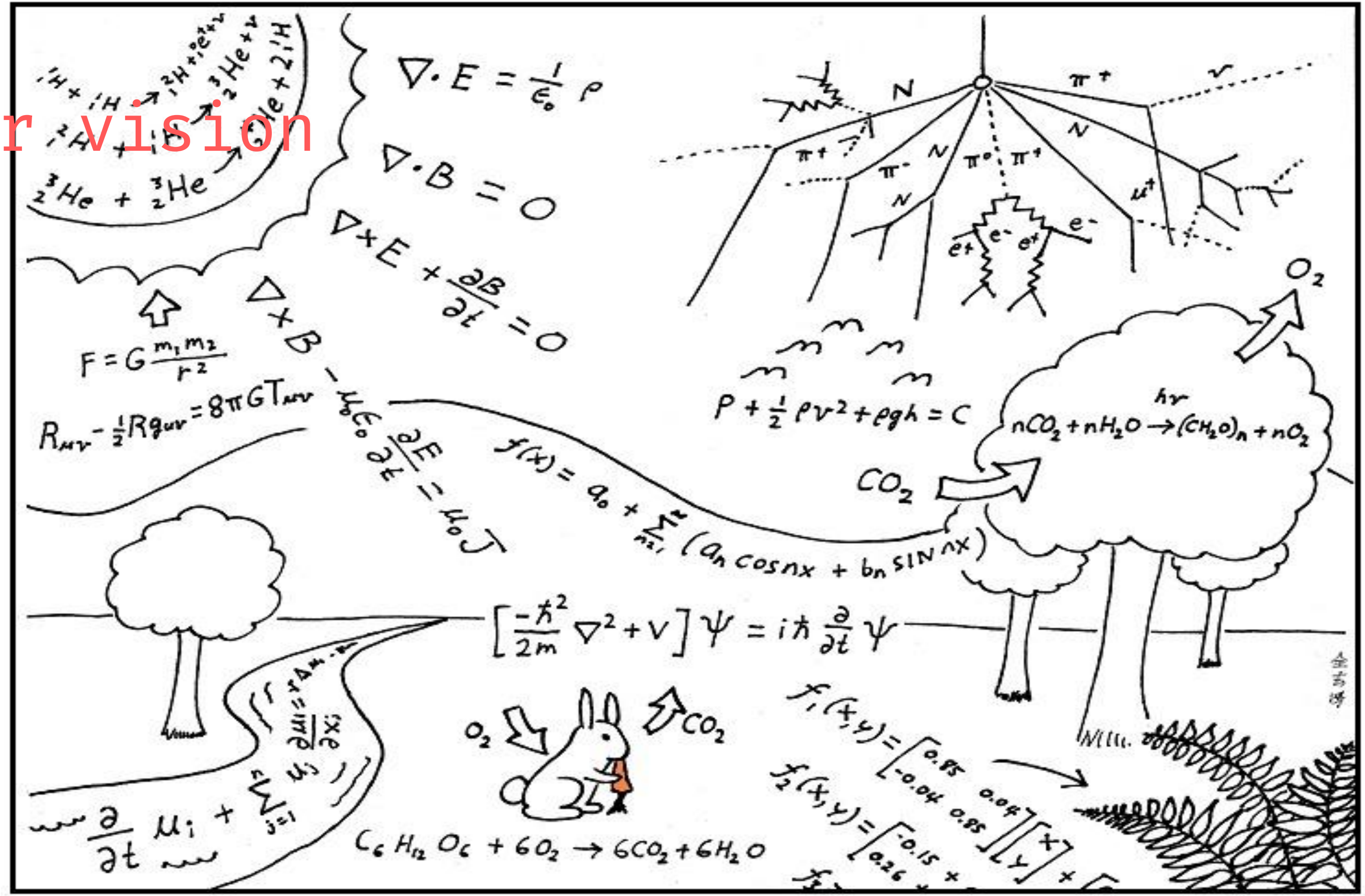


The model…



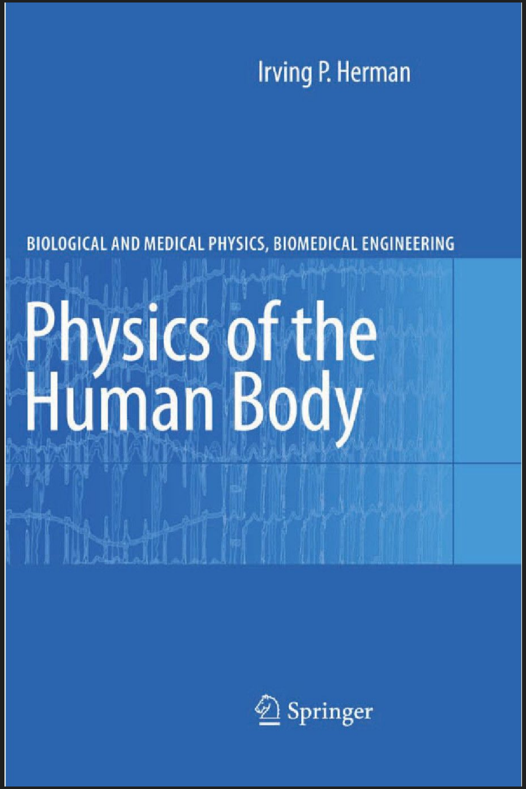

EL CABALLO ESFÉRICO
Temas de Física en Biología y Medicina
Verónica Grünfeld
Lugar Científico

Our vision

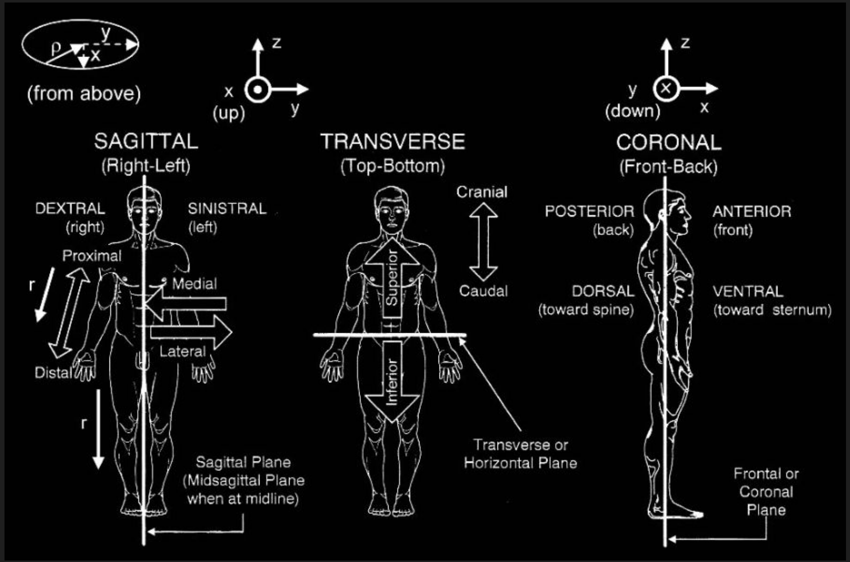


²₃He + ³₂He
F = G m₁m₂ / r²
Rμν − ½Rgμν = 8πGTμν
∇·E = 1/ε₀ ρ
∇·B = 0
∇×E + ∂B/∂t = 0
∇×B − μ₀ε₀ ∂E/∂t = μ₀J
N
e⁻
O₂
P + ½ρv² + ρgh = C
hν
nCO₂ + nH₂O → (CH₂O)ₙ + nO₂
CO₂
f(x) = a₀ + Σ (aₙ cos nx + bₙ sin nx)
[−ħ²/2m ∇² + V]ψ = iħ ∂/∂t ψ
O₂
CO₂
C₆H₁₂O₆ + 6O₂ → 6CO₂ + 6H₂O

The jargon



“Much of the problem we have in comprehending specialists in any field is in understanding their jargon, and not in understanding their ideas. This is particularly true for medicine”



The conventions for this course

This is not a theoretical course. This is a **hands-on** course. So we will work on the natural environments of Geant4: an editor (or **IDE**) and the (Linux) **terminal** (Windows users→some IDEs included their own terminal)

Geant4 is enterelly written in **C++** (mandatory), and some **bash** knowledge is always recommended. So, within this course, slides are written in english and following the highlight conventions for **C++** or **bash** within these slides.

There are some python approaches to Geant4 but they are out of the scope of this crash course.

Conventions for this course

- Bash conventions:

```bash
# This is a comment in bash
$ make # this means run at the command in your CLI as user
# make # this means to run the command in your CLI as root (sudo)
```

- C++ conventions:

```cpp
/* This is a C++ long comment  */
// This is a C++ short comment, and below there is a typical IDE view
class MyDetectorConstruction : public G4VUserDetectorConstruction {
// your class goes here
};
```

The conventions for this course

OS: I personally recommend any updated ($\geqslant$ 22.04), ubuntu flavor (ubuntu, mint, xubuntu, kubuntu, …).

However, of course you should use whatever OS you feel comfortable, even Windows or iOS. You can also use virtualization environments for running G4 (later on this course).

IDEs: Integrated development environments

- I strongly recommend using **VIm** or an **IDE** for programming
- There are so many possible classes, not all of them following an standard naming convention, that it could be helpful to take the advantages of an IDE.
- You should explore the several available IDEs. At the end, all of them will have the functionalities you expected.
- I always used VIm, but recently started using PyCharm, but finally I migrated to VS Code. However, VIm works in every HPC environment, and PyCharm has the best LaTeX plugin I've found (→Texify).

PC

My personal list of VScode extensions

geant4

// A toolkit for the simulation of the passage of particles through matter.

// Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science

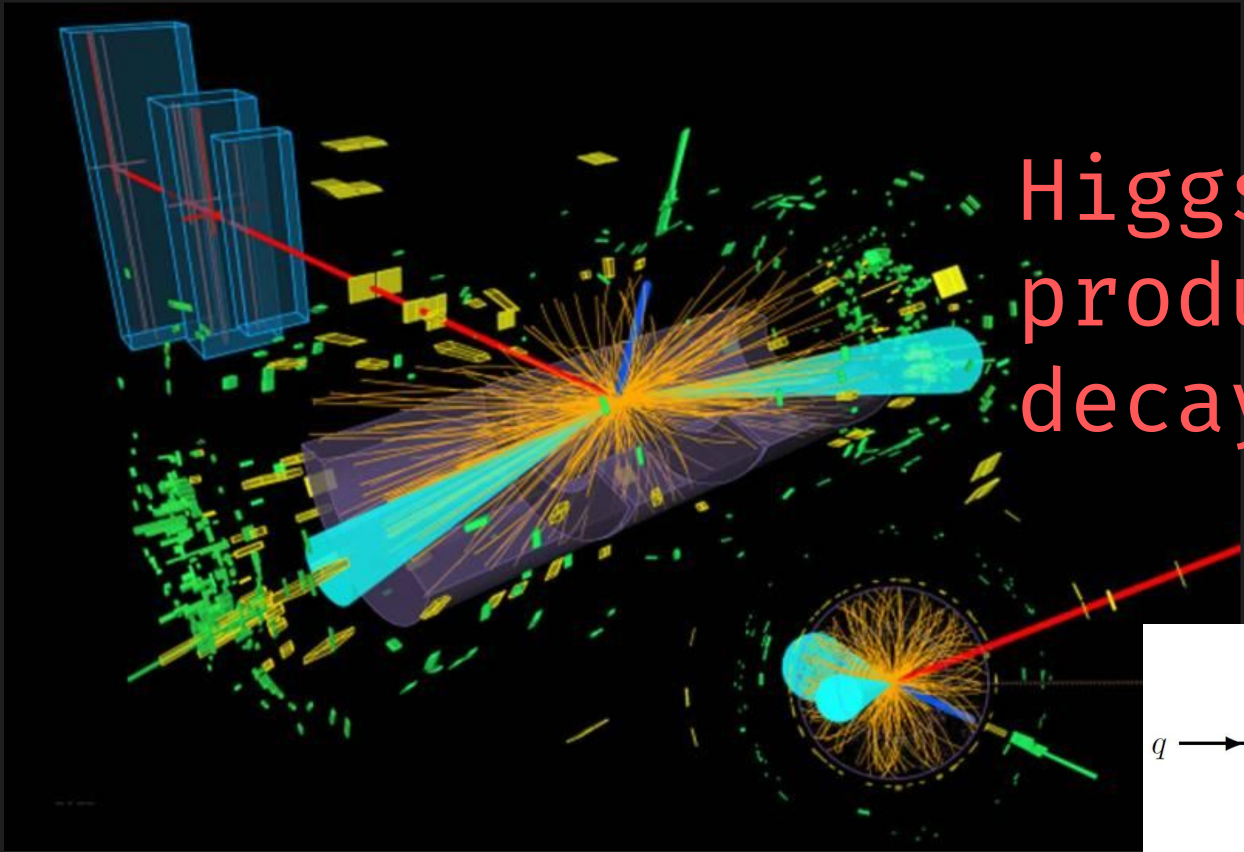
```
G4Download("geant4.web.cern.ch/");
G4Docs("geant4.web.cern.ch/docs/");
G4AppDocs("geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/index.html");
G4Examples("Check the ${geant4_examples}/ dir for extra fun");
```

// IMPORTANT NOTICE
// I will not spend time showing how to install G4 (and root) →

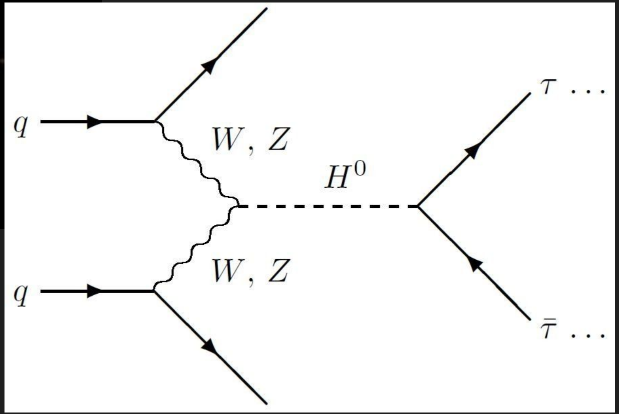
G4 Scope {

```
/*

It is integraly programmed in C/C++ and allow to build applications
including all the aspects of a Monte Carlo simulation process:

    * the geometry and building materials of the system;
    * the fundamental particles involved and all the physics process
        governing particle interactions;
    * the tracking of particles in matter and EM fields;
    * the medium/detector response to the passage of these particles
        and their by-products;
    * the visualization of the detector and particle trajectories; and
    * the capture and analysis of simulation data at different levels
        of detail and refinement.

*/
```

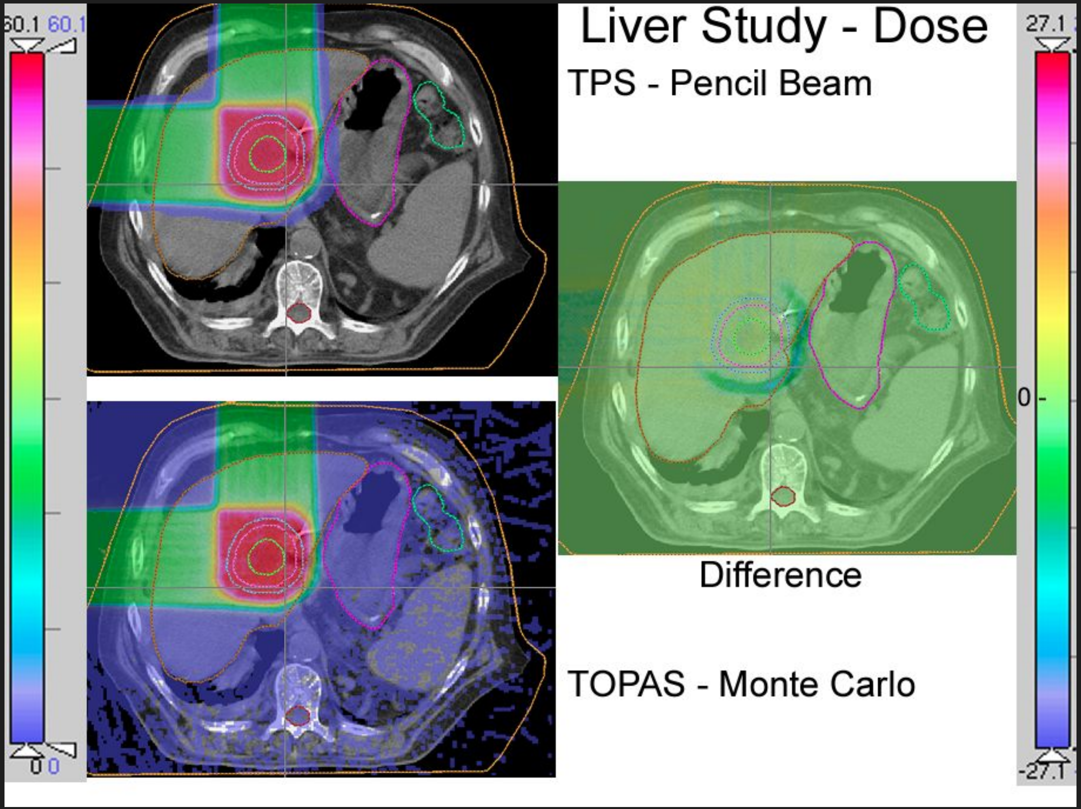
}



Higgs production and decay at ATLAS



$qq \rightarrow H^0 \rightarrow \tau\tau$



Allow absorbed dose ($E_D \rightarrow [E_D]=Gy=J\ kg^{-1}$) calculations

Allow time-evolving geometries

geant4

```
// A comment about versions.
/*
    By the end of 2022 a new major release, G4 11
    was released (current 11.01p01)

    In this course we will use the latest G4 10
    version (10.07p04)
*/
```

G4 install (by @asoreyh)

While Geant4 have multiple dependencies, some of them are required and some others are needed for optional features.

```
# Check the official installation guide at https://geant4-userdoc.web.cern.ch/UsersGuides/InstallationGuide/html

# I prepared a bash script for installing the required dependencies, root and geant4 at ${HOME}/work. (Ubuntu 23.04)
# Warning 1: it will change your .bashrc file.
# Warning 2: it will take time (up to several hours).

$ curl -Lo install-root-geant4.sh https://www.dropbox.com/s/ej67f1hc88u7w1a/install-root-geant4.sh?dl=1
$ chmod 744 install-root-geant4.sh
$ ./install-root-geant4.sh
```

G4 docker (by @asoreyh)

Docker is a platform designed to help devs build, share, and run modern apps. We handle the tedious setup, so you can focus on the code.

```
# I prepared two Dockerfiles for this course. Follow the instructions
and download them from (look for them at the utils directory):

$ git clone https://github.com/asoreyh/geant4-course.git

# Otherwise, you can pull the docker image from my docker hub:

$ docker pull asoreyh/root:latest   # root version 6.28.04 (2023)
$ docker pull asoreyh/geant4:latest # G4 version 10.07.04 (2022)

# Blank installation. Check the docs!
# There is also a virtual machine built by the Geant4 collaboration
```

G4 docker (by @asoreyh)

Important note: You will need to follow these steps and provide privileged access to this docker to be able to run the QT Geant4 visualization from docker.

```
# open a terminal and enable local access to xhost:

$ xhost +local:root

# and run the docker (if you don't download the docker images it will download them):

$ docker run --privileged -it -e DISPLAY=$DISPLAY -v /tmp/.X11-unix:/tmp/.X11-unix asoreyh/geant4:10.07.04
```

About this course

- This is a **hands-on course**. During this 4 hours we will code from the scratch two Geant4-based applications for:
  - calculate the **deposited dose** in an organ
  - simulate an EM **shielding**
- Hopefully:
  - you will reuse these codes for **building your own G4 apps**
  - this will give you a **global view** on building G4 apps
- The final version of codes are available at GitHub, but:
  - we will write them from scratch here. Use GitHub codes only as a reference in case of troubles
  - try different geometries, materials, particle beams, …
  - analyze the differences with your own codes (python, ...)

About this course

Introduction

A first contact with G4 and its examples

The basics

- Sensitivity and efficiency volumes
- Visualization and outputs
- Deposited energy

[1] — [2] — [3] — [4]

Building blocks

- Structure and common practices
- Detector construction
- Physics lists
- Generating particles

Apps

- Calculating the absorbed dose in a tissue
- Simulating the effect of shielding

Geant4 philosophy→ Toolkit

G4Help();

No main code, tools for building your own app — Toolkit

Actions — Components (interfaces) of your app

The building blocks of your app — Classes

Application — Your implementation

Declarations (.hh)

G4Help();

```
// In complex codes, it is probable you will include the same file at different declaration files.
// In some cases, this could introduce issues due multiple definitions.
// To avoid that, it is a common and highly recommended practice to start any declaration as:

# ifndef FILE_HH // (replace FILE by the declaration)
# define FILE_HH

/* Continue with your declarations here */

# endif
```

CMake {

Configuring geant4 apps is performed by using

```
$ cmake && make && make install
```

\# (or ccmake)
\# Read the docs for the details, but…
\# CMake is an open-source, cross-platform family of tools designed to build, test and package software.
\# CMake is used to **control the software compilation** process using simple platform and **compiler independent configuration** files, and generate native makefiles and workspaces that can be used in the compiler environment of your choice.
\# CMake improves the usual (and deprecated)

```
$ ./configure && make && make install
```

\# method, by providing an easy way to write Makefiles for compiling and installing applications

}

A geant4 application...

```
# … is actually a cmake project. First we need to create a folder

$ mkdir geant4

# and open this folder in VS Code (or your favorite IDE or editor). Then, create a new file called:

$ vim CMakeList.txt

# We will use it for configuring our project.

}
```

Our CMakeLists.txt

```
# Minimum version of cmake required for compiling or project (I recommended at least cmake v 2.8.12, better v 3)
cmake_minimum_required (VERSION 2.8.12 FATAL_ERROR)

# The name of the project, we will use "Dose" for the G4 dose calculation applications
project(G4Dose)

# Let's tell cmake what packages we required or need (not the same!)
# No UI or VIS, uncomment this and comment the other one
# find_package(Geant4 REQUIRED)
find_package (Geant4 REQUIRED ui_all vis_all)

# Include G4 libraries. ${string} are environmental variables
include(${Geant4_USE_FILE} )

# Locate sources and headers
# ${PROJECT_SOURCE_DIR} is the directory where CMakeLists.txt is located (${PWD})

include_directories (${PROJECT_SOURCE_DIR} /inc
                    ${Geant4_INCLUDE_DIR} )
file(GLOB headers  ${PROJECT_SOURCE_DIR} /inc/*.hh)
file(GLOB sources  ${PROJECT_SOURCE_DIR} /src/*.cc)

# The name of the excecutable we will use, and the files that will be linked to it
# dose is the excecutable, dose.cc the main source, etc
add_executable (dose dose.cc  ${sources}  ${headers} )

# The Geant4 libraries that will be linked to our excecutable...
target_link_libraries (dose  ${Geant4_LIBRARIES} )

# Linking the excecutable with our project
add_custom_target (G4Dose DEPENDS dose)
```

G4RunManager()

```
/*
    This object is the "heart" of any G4 application. It is always mandatory and
    should be defined in your main app.cc code (dose.cc in our example)

    It controls the "flow" of the run

    All the interfaces (G4 toolkit) are defined and provided here:

        * G4VUserDetectorConstruction          ← geometry construction

        * G4VUserPhysicsList              ← all your physics is here

        * G4VUserActionInitialization          ← actions

            * G4VUserPrimaryGeneratorAction    ← primary particles production

            * G4UserRunAction                  ← optionals…

            * G4UserSteppingAction, …

        * UIManager, VisManager, …

*/
```

G4RunManager()

```
/*
    This object is the "heart" of any G4 application. It is always mandatory and
    sho
    It
    A
                                                                              ction
            * G4UserRunAction                   ← optionals…
            * G4UserSteppingAction, …
        * UIManager, VisManager, …
*/
```

Now we will start to create our Geant4 app code

Create your G4RunManager

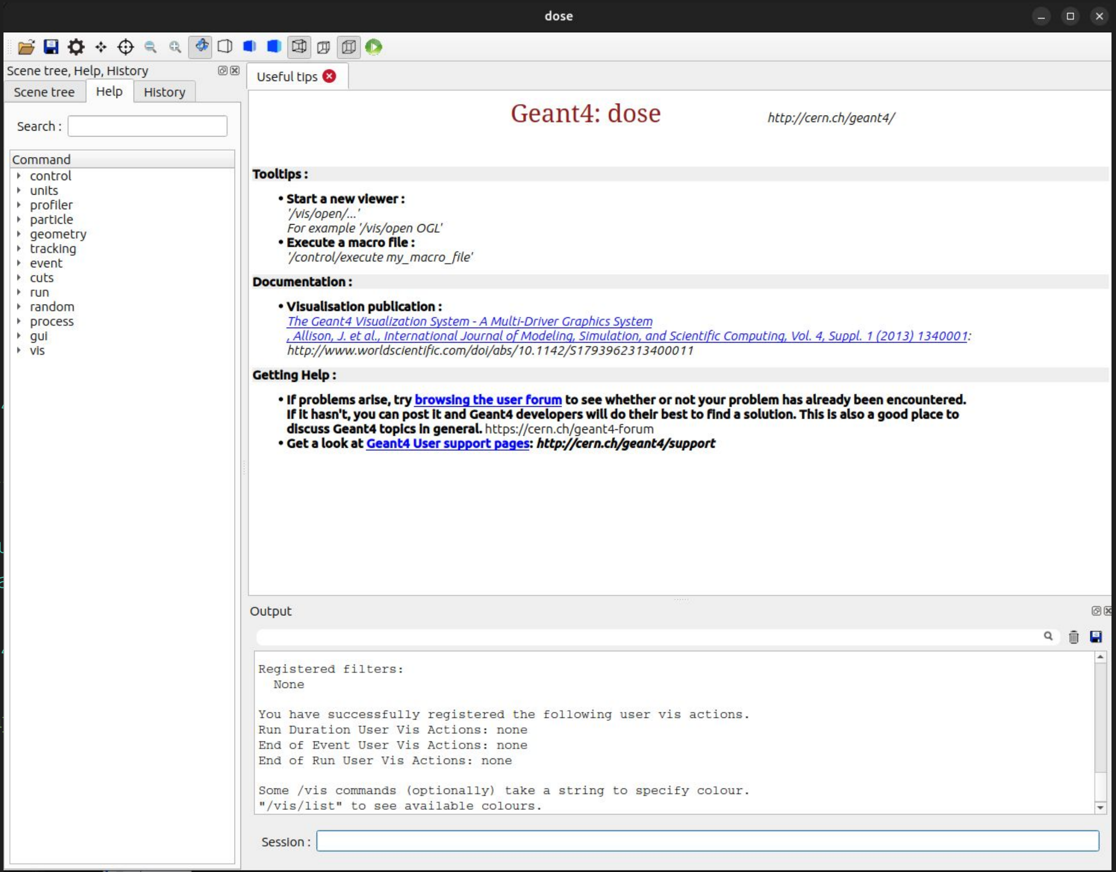
(your main code app → dose.cc)

First step: the main code (dose.cc)

```cpp
// 0. I/O operations
#include <iostream>
// 1. G4RunManager class
#include "G4RunManager.hh"
// 2. User interface
#include "G4UImanager.hh"
#include "G4UIExecutive.hh"
// 3. Visualization
#include "G4VisManager.hh"
#include "G4VisExecutive.hh"

int main(G4int argc, char** argv) {
	//1. create the G4RunManager object
	G4RunManager *runManager = new G4RunManager();
	//5. Initialize the runManager
	// runManager->Initialize(); // uncomment to see what happens
	//2. create the user interfase
	G4UIExecutive *ui = new G4UIExecutive(argc, argv);
	G4UImanager *UIManager = G4UImanager::GetUIpointer();
	//3. visualization manager
	G4VisManager *visManager = new G4VisExecutive();
	visManager->Initialize();
	// 4. start the session - and compile to see what happens
	ui->SessionStart();
	return 0;
}
```

First step: the main code (dose.cc)

```
// 0. I/O operations
#include <iostream>
// 1. G4RunManager class
#include "G4RunManager.hh"
// 2. User interface
#include "G4UImanager.hh"
#include "G4UIExecutive.hh"
// 3. Visualization
#include "G4VisManager.hh"
#include "G4VisExecutive.hh"

int main(G4int argc, char** argv) {
        //1. create the G4RunManager obje
        G4RunManager *runManager = new G4
        //5. Initialize the runManager
        // runManager->Initialize(); // u
        //2. create the user interfase
        G4UIExecutive *ui = new G4UIExecu
        G4UImanager *UIManager = G4UImana
        //3. visualization manager
        G4VisManager *visManager = new G4
        visManager->Initialize();
        // 4. start the session - and com
        ui->SessionStart();
        return 0;
}
```



First step: the main code (dose.cc)

```cpp
// 0. I/O operations
#include <iostream>
// 1. G4RunManager class
#include "G4RunManager.hh"
// 2. User interface
#include "G4UImanager.hh"
#include "G4UIExecutive.hh"
// 3. Visualization
#include "G4VisManager.hh"
#include "G4VisExecutive.hh"

int main(G4int argc, char** argv) {
        //1. create the G4RunManager object
        G4RunManager *runManager = new G4RunMana
        //5. Initialize the runManager
        runManager->Initialize(); // uncommen
        //2. create the user interface
        G4UIExecutive *ui = new G4UIExecutive(ar
        G4UImanager *UIManager = G4UImanager::Ge
        //3. visualization manager
        G4VisManager *visManager = new G4VisExecutive();
        visManager->Initialize();
        // 4. start the session - and compile to see what happens
        ui->SessionStart();
        return 0;
}
```

```
[100%] Built target dose
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ ./dose

**************************************************************
 Geant4 version Name: geant4-10-07-patch-04    (9-September-2022)
                      Copyright : Geant4 Collaboration
                     References : NIM A 506 (2003), 250-303
                                : IEEE-TNS 53 (2006), 270-278
                                : NIM A 835 (2016), 186-225
                            WWW : http://geant4.org/
**************************************************************


-------- EEEE ------- G4Exception-START -------- EEEE -------
*** G4Exception : Run0033
      issued by : G4RunManager::InitializeGeometry
G4VUserDetectorConstruction is not defined!
*** Fatal Exception *** core dump ***
 **** Track information is not available at this moment
 **** Step information is not available at this moment

-------- EEEE -------- G4Exception-END --------- EEEE -------

*** G4Exception: Aborting execution ***
Aborted (core dumped)
```

// Clearly we are still not ready for **initialize()** the **runManager** as we need to continue defining our **basics building blocks**

```
G4Course *MyNewCourse::Construct() {

    G4Course *course = new G4Course();
    course->title(“Geant4 for Beginners. A crash course”);
    course->author(“Hernán Asorey”);
    course->email(“asoreyh@gmail.com”);
    course->description(“a hands-on Geant4 crash course”);
    course->school(“La Conga Physics”);
    course->site(“github.com/asoreyh/geant4-course”);
    course->year(2023);
    course->duration(4*h);
    course->license(“CC0 1.0 Universal”);
    return course;
}
```

First step: the main code (dose.cc)

```
// 0. I/O operations
#include <iostream>
// 1. G4RunManager class
#include "G4RunManager.hh"
// 2. User interface
#include "G
#include
// 3. Vi
#include
#include

int main
    //
    G4
    //
    ru
    //
    G4
    G4UIm
    //3. visualization manager
    G4VisManager *visManager = new G4VisExecutive();
    visManager->Initialize();
    // 4. start the session - and compile to see what happens
    ui->SessionStart();
    return 0;
}
```

```
[100%] Built target dose
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ ./dose
Aborted (core dumped)
```

Before to continue we need to define our "volumes", i.e., where your app detectors and volumes will exist and what are they made of? (always 3 volumes, see next)

Create your G4VUserDetectorConstruction
(and register it at your runManager)

// Clearly we're still not ready for **initialize()** the **runManager** as we need to continue defining our **basic building blocks**

```
// Materials
G4NistManager *nist = G4NistManager::Instance();

// Materials are made of elements, elements are made of isotopes
G4MaterialsDocs(“https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/GettingStarted/materialDef.html”);
```

- G4Isotope ← name & index. Properties atoms (Z,N(ucleons),molar mass)
- G4Element ← name, index & symbol. Properties elements (Z_{eff}, N_{eff}, A_{eff})
- G4Material ← name & index, macroscopic properties (ρ, T, p, state)

// Materials

// Materials are made of elements, elements are made of isotopes

G4MaterialsDocs("https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/GettingStarted/materialDef.html");

- G4Isotope ← name & index. Properties atoms (Z,N(ucleons),molar mass)
- G4Element ← name, index & symbol. Properties elements (Z_{eff}, N_{eff}, A_{eff})
- G4Material ← name & index, macroscopic properties (ρ, T, p, state)

```
// let's create a molecule of H2
// create the natural isotopes of H
G4Isotope *H = new G4Isotope("H", 1, 1, 1.*g/mole);
G4Isotope *D = new G4Isotope("D", 1, 2, 2.*g/mole);
// create the element as a mix of isotopes
G4Element *elH = new G4Element("Hydrogen", "H", 2);
elH->AddIsotope(H, 99.985*perCent);
elH->AddIsotope(D,  0.015*perCent);
// create the molecule as a material
G4Material *matH2 = new G4Material("H2", 0.08375 * kg/m3, 1);
matH2->AddElement(elH, 2);
```

Need to be created inside a function

```cpp
// Materials
G4NistManager *nist = G4NistManager::Instance();

// NIST database → >3000 isotopes, 108 elements, >~300 materials
// G4_Al, G4_C, G4_U, G4_Si, …     ← elements
// G4_AIR, G4_WATER, G4_CALCIUM_CARBONATE, …   ← compounds
// G4_DNA_ADENINE, G4_CYTOSINE, …   ← biochemical compounds
// G4_KEVLAR, G4_DACRON, …     ← industrial materials
G4NistMaterialsRef("https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/Appendix/materialNames.html?highlight=nist%20materials");
// HowTo
   // instanciate the NIST manager
   G4NistManager *nist = G4NistManager::Instance();
   G4Material *matWater = nist->FindOrBuildMaterial("G4_WATER");
   G4Material *matConcrete = nist->FindOrBuildMaterial("G4_CONCRETE");
   G4Material *matCaCo3 = nist->FindOrBuildMaterial("G4_CALCIUM_CARBONATE");
   // List NIST materials:
   nist->ListMaterials("all");//simple, compound, hep, space, bio, all
```

// 3 types of volumes: solid, logical, physical

G4GeometryDocs(“https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/Detector/Geometry/geometry.html”);

- Solid ← what is the shape and geometry (half and half!) of your volume?

```
G4Box *solidWorld = new G4Box("solidWorld", 0.5*m, 0.5*m, 0.5*m);
```

- Logical ← what is made of?

```
G4Material *air = nist->FindOrBuildMaterial("G4_AIR");
G4LogicalVolume *logicWorld = new G4LogicalVolume(solidWorld, air, "logicWorld");
```

- Physical ← where the magic (interactions, …) occurs

```
G4VPhysicalVolume *physWorld = new G4PVPlacement(
    0, G4ThreeVector(0.,0.,0.), logicWorld, "physWorld", 0, false, 0, true
);
```

construction.hh/.cc: materials, volumes

```cpp
#ifndef CONSTRUCTION_HH
#define CONSTRUCTION_HH
// 1. System of Units and Physical Constants (not mandatory, but...
always!)
#include "G4SystemOfUnits.hh"
#include "G4PhysicalConstants.hh"
// 2. NIST class, needed for standard materiales (idem)
#include "G4NistManager.hh"
// 3. detector construction class
#include "G4VUserDetectorConstruction.hh"
// 6. Volumes: physical, logicals and placements
#include "G4VPhysicalVolume.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
// some standard shapes
#include "G4Box.hh"
// #include "G4Sphere.hh"
// 4. choose the name of your class
class MyDetectorConstruction : public G4VUserDetectorConstruction{
// 5. The class constructor and destructor, they are public
public:
   MyDetectorConstruction();
   ~MyDetectorConstruction();
// 6. the constructor of the physical volume
//    it will construct the physical volume of your system
   G4VPhysicalVolume *Construct();
};
#endif
```

```cpp
#include "construction.hh"
// as always, first the constructor and destructor
MyDetectorConstruction::MyDetectorConstruction() {}
MyDetectorConstruction::~MyDetectorConstruction() {}
// now the construction function
G4VPhysicalVolume *MyDetectorConstruction::Construct() {
   // 1. your system is build using materials... using NIST:
   G4NistManager *nist = G4NistManager::Instance();
   // usually (but not always, world is made by Air, ie, G4_AIR)
   G4Material *worldMat = nist->FindOrBuildMaterial("G4_AIR");
   // 2. your world should have a shape. Let's do a box.
   // It always you have three volumes:
   // solid (the shapes), logical (the materials), physicals (the magic)
   // 2.1 solid: name, x/2, y/2, z/2, use the units!!!
   G4Box *solidWorld = new G4Box("solidWorld", 0.5*m, 0.5*m, 0.5*m);
   // 2.2 logical: assing the material: solid, material, name
   G4LogicalVolume *logicWorld = new G4LogicalVolume(
       solidWorld, worldMat, "logicWorld"
   );
   // 2.3 physical: where the magic occurs: rotation, position(x,y,z),
   //     associated logical volume, name, motherVolume?, bools (negate
   //     volumes!), numberOfCopies, check for overlaps(always)
   G4VPhysicalVolume *physWorld = new G4PVPlacement(
       0, G4ThreeVector(0.,0.,0.), logicWorld, "physWorld", 0, false, 0,
true
   );
   return physWorld;
}
```

And the new dose.cc

```cpp
// 0. I/O operations
#include <iostream>
// 1. G4RunManager class
#include "G4RunManager.hh"
// 2. User interface
#include "G4UImanager.hh"
#include "G4UIExecutive.hh"
// 3. Visualization
#include "G4VisManager.hh"
#include "G4VisExecutive.hh"
// once the detector construction is ready, include it
#include "construction.hh"
int main(G4int argc, char** argv) {
   //1. create the G4RunManager object
   G4RunManager *runManager = new G4RunManager();
   // once detector is created, then define it
   // but we are still not ready for init
   runManager->SetUserInitialization(new MyDetectorConstruction());
   //5. Initialize the runManager
   // runManager->Initialize(); //
   //2. create the user interfase
   G4UIExecutive *ui = new G4UIExecutive(argc, argv);
   G4UImanager *UIManager = G4UImanager::GetUIpointer();
   //3. visualization manager
   G4VisManager *visManager = new G4VisExecutive();
   visManager->Initialize();
   // 4. start the session - and compile to see what happens
   ui->SessionStart();
   return 0;
}
```

It compiles! :)

```
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ make
[ 33%] Building CXX object CMakeFiles/dose.dir/dose.cc.o
/home/asoreyh/Dropbox/projects/geant4/geant4-course/codes/src/dose.cc: In function
‘int main(G4int, char**)’:
/home/asoreyh/Dropbox/projects/geant4/geant4-course/codes/src/dose.cc:26:18:
warning: unused variable ‘UIManager’ [-Wunused-variable]
   26 |     G4UImanager *UIManager = G4UImanager::GetUIpointer();
      |                  ^~~~~~~~~
[ 66%] Building CXX object CMakeFiles/dose.dir/construction.cc.o
[100%] Linking CXX executable dose
[100%] Built target dose
```

But will not work… :/

And the new dose.cc

```
// 0. I/O operations
#include <iostream>
// 1. G4RunManager class
#include "G4RunManager.hh"
// 2. User interface
#include "G4UImanager.hh"
#include "G4UIExecutive.hh"
// 3. Visualization
#include "G4VisManager
#include "G4VisExecuti
// once the detector c
#include "construction
int main(G4int argc, ch
   //1. create the G4R
   G4RunManager *runMa
   // once detector is
   // but we are still
   runManager->SetUser
   //5. Initialize the
   // runManager->Init
   //2. create the use
   G4UIExecutive *ui = ne
   G4UImanager *UIManager = G4UImanager::GetUIpointer();
   //3. visualization manager
   G4VisManager *visManager = new G4VisExecutive();
   visManager->Initialize();
   // 4. start the session - and compile to see what happens
   ui->SessionStart();
   return 0;
}
```

It compiles! :)

```
t4-course/codes/src/build$ make
/dose.cc.o
urse/codes/src/dose.cc: In function

urse/codes/src/dose.cc:26:18:
variable]
r::GetUIpointer();

construction.cc.o
```

ork… :/

Before to continue we need to define our “physics”, i.e., what kind of physics our app will implement?

Create your G4VUserPhysicsList

(and register it at your runManager)

Physics, this is why we are here…

// Physics process

Physics processes describe how particles interact with materials.

- electromagnetic
- hadronic
- transportation
- decay
- optical
- photolepton_hadron
- Parameterisation

G4PhysicsListDocs("https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/TrackingAndPhysics/physicsProcess.html");

PhysicsList, make it simple…

// A physics list

- Specify and describe all the particles that will be allowed in the app
- Specify all the physics process assigned to them
- Provides a flexible way to setup the physics of your app

// Yes, but

- You need to know which process are relevant for the energy scales of your application
- Your physics model accuracy will depend on the physics lists you include, but…
- … include only what you need
- Many physics lists overlap each other

G4PhysicsListDocs("https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/UserActions/mandatoryActions.html?highlight=physicslist#physics-lists");

… but no so simple… So → Modular PL

- There are several ready-to-use modular physics lists
- They were constructed by experts, but they are given as it is…
- Many physics lists overlap each other
- Some current lists, all includes HAD, EM, Decays, neutrons
    - FTFP_BERT ← Current G4 default (collider physics)
    - FTFP_BERT_HP ← Idem FTFP_BERT but n with En < 20 MeV are treated separately by the HP neutron models. Requires G4NDL and RadiactiveDecay
    - Shielding ← Simulation of deep shielding (includes HP)
    - QGSP_BERT ← former G4 default. Similar but replaced by FTFP_BERT
    - QGSP_BERT_HP ← Idem
    - G4OpticalPhysics ← Cherenkov and Scintillation (→photon process)
    - G4EmStandardPhyscis ← EM constructors, see the docs

```
G4PhysicsListGuide("https://geant4-userdoc.web.cern.ch/UsersGuides/PhysicsListGuide/BackupVersions/V10.7/html/index.html");
```

So, Physics → 3 alternative ways

- Start from scratch → outside the scope of this course (see the docs)

- Use Physics constructors → let's see for a simple cases
  - Create a class (suggest to create physics.hh and physics.cc)

    ```
    class MyPhysicsList : public G4VModularPhysicsList
    ```

  - Construct the list

    ```
    MyPhysicsList::MyPhysicsList() {
          RegisterPhysics (new G4EmStandardPhysics()); // EM constructor
          RegisterPhysics (new G4OpticalPhysics()); // for optical processes
    }
    ```

  - Register the list in the RunManager (dose.cc)

    ```
    runManager->SetUserInitialization(new MyPhysicsList());
    ```

- Or, use Physics List factory → **highly recommended, directly at the main app**

  ```
  #include "G4PhysListFactory.hh" // uncoment if you are using PL factory
  const G4String physicsListName = "FTFP_BERT_HP";
  // [...]
  G4PhysListFactory physicsListFactory;
  physicsListFactory.SetVerbose(1);
  G4VModularPhysicsList *physicsList = physicsListFactory.GetReferencePhysList(physicsListName);
  runManager->SetUserInitialization(physicsList);
  ```

There we go… physics.hh/.cc

```cpp
#ifndef CONSTRUCTION_HH
#define CONSTRUCTION_HH
// 1. System of Units (not mandatory, but... always!)
#include "G4SystemOfUnits.hh"
// 2. NIST class, needed for standard materiales (idem)
#include "G4NistManager.hh"
// 3. detector construction class
#include "G4VUserDetectorConstruction.hh"
// 6. Volumes: physical, logicals and placements
#include "G4VPhysicalVolume.hh"
#include "G4LogicalVolume.hh"
#include "G4PVPlacement.hh"
// some standard shapes
#include "G4Box.hh"
// #include "G4Sphere.hh"
// 4. choose the name of your class
class MyDetectorConstruction : public G4VUserDetectorConstruction {
// 5. The class constructor and destructor, they are public
public:
   MyDetectorConstruction();
   ~MyDetectorConstruction();
// 6. the constructor of the physical volume
//    it will construct the physical volume of your system
   G4VPhysicalVolume *Construct();
};
#endif
```

```cpp
#include "physics.hh"

// 1. Create the constructor
MyPhysicsList::MyPhysicsList() {
RegisterPhysics (new G4EmStandardPhysics()); // only use what you need
RegisterPhysics (new G4OpticalPhysics());
}

// 2. Create the destructor
MyPhysicsList::~MyPhysicsList() {
}
// 3. Register in the main app file ->
```

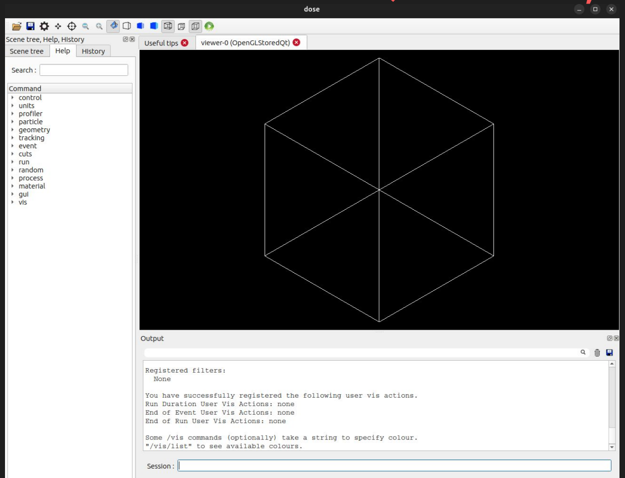
And the new lines at dose.cc

```cpp
// [...]
// once the physics list is ready, include it
#include "physics.hh"

int main(G4int argc, char** argv) {

// [...]

// once the physics is created, reigster it
runManager->SetUserInitialization(new MyPhysicsList());

// [...]
// after the physics, draw the OGL and set the viewpoint...
UIManager->ApplyCommand("/vis/open OGL");
UIManager->ApplyCommand("/vis/viewer/set/viewpointVector 1 1 1");
// ... draw the volumes
UIManager->ApplyCommand("/vis/drawVolume");

// [...]
}
```

It compiles! :)

```
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ make
[ 33%] Building CXX object CMakeFiles/dose.dir/dose.cc.o
/home/asoreyh/Dropbox/projects/geant4/geant4-course/codes/src/dose.cc: In function
'int main(G4int, char**)':
/home/asoreyh/Dropbox/projects/geant4/geant4-course/codes/src/dose.cc:26:18:
warning: unused variable 'UIManager' [-Wunused-variable]
   26 |     G4UImanager *UIManager = G4UImanager::GetUIpointer();
      |                  ^~~~~~~~~
[ 66%] Building CXX object CMakeFiles/dose.dir/construction.cc.o
[100%] Linking CXX executable dose
[100%] Built target dose
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ ./dose
```

And works! (I know, I know)



And the new lines at dose.cc

```
// [...]
// once the physics list is ready, include it
#include "physics.hh"

int main(G4int argc, char** argv) {

// [...]

// once the ph
runManager->Se

// [...]
// after the p
UIManager->App
UIManager->App
// ... draw th
UIManager->App

// [...]
}
```

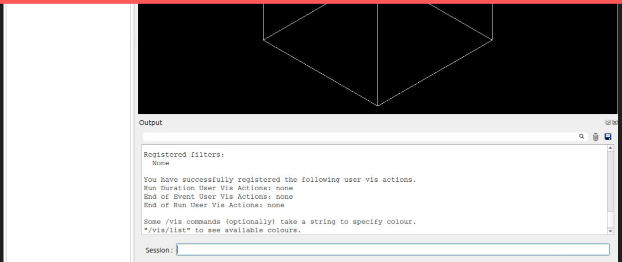
It compiles! :)

```
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ make
[ 33%] Building CXX object CMakeFiles/dose.dir/dose.cc.o
/home/asoreyh/Dropbox/projects/geant4/geant4-course/codes/src/dose.cc: In function
e.cc:26:18:
uild$ ./dose
```

know)

Ok, we have volumes, materials and physics…
We are almost ready → we need actions!

Create your G4VUserActionInitialization and G4VUserPrimaryGeneratorAction
(and register them at your runManager)



Actions, let's the things evolve

// We need two interfaces

- **G4VUserActionInitialization** is an interface to create and register the **G4VUserPrimaryGeneratorAction** (mandatory) and other user actions
  - Build() ← function

- **G4VUserPrimaryGeneratorAction** is an interface (action!) to describe how the primary particles (injection) should be produced
  - GenerateParticles() ← function
  - Typically, but not always → **G4ParticleGun**
  -

There we go… action.hh/.cc

```cpp
#ifndef ACTION_HH
#define ACTION_HH

#include "G4VUserActionInitialization.hh"

#include "G4VUserPrimaryGeneratorAction.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4SystemOfUnits.hh"

class MyActionInitialization : public G4VUserActionInitialization{
public:
MyActionInitialization();
~MyActionInitialization();
virtual void Build() const; // our main function
};

class MyPrimaryGenerator : public G4VUserPrimaryGeneratorAction{
public:
MyPrimaryGenerator();
~MyPrimaryGenerator();
virtual void GeneratePrimaries(G4Event*);

private:
G4ParticleGun *fParticleGun;
};
#endif
```

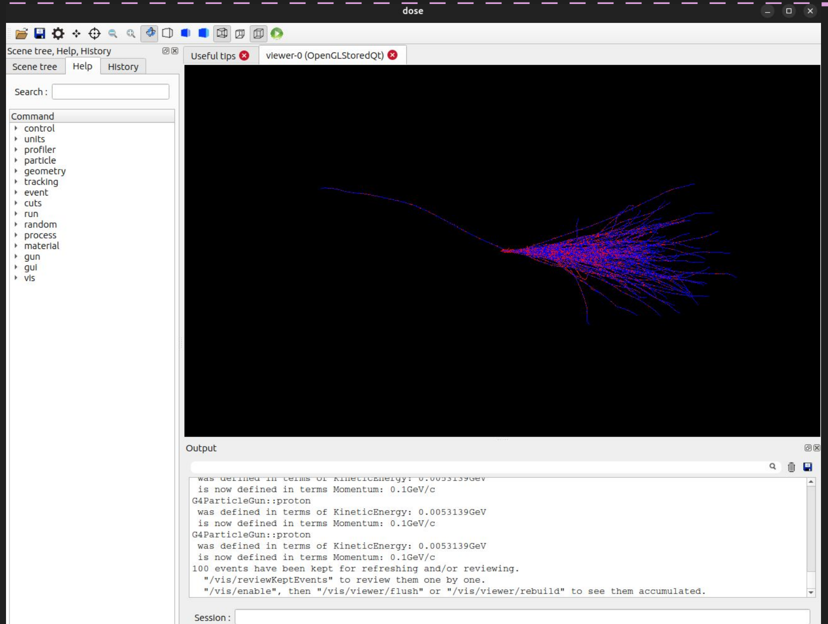
```cpp
#include "action.hh"
MyActionInitialization::MyActionInitialization() {}
MyActionInitialization::~MyActionInitialization() {}
void MyActionInitialization::Build() const {
MyPrimaryGenerator *generator = new MyPrimaryGenerator();
SetUserAction(generator);
}
MyPrimaryGenerator::MyPrimaryGenerator(){
fParticleGun = new G4ParticleGun(1);
}
MyPrimaryGenerator::~MyPrimaryGenerator(){
delete fParticleGun;
}
void MyPrimaryGenerator::GeneratePrimaries(G4Event *anEvent) {
G4ParticleTable *particleTable = G4ParticleTable::GetParticleTable();
G4String particleName = "proton";
G4ParticleDefinition *particle = particleTable->FindParticle(particleName);
G4ThreeVector pos(0., 0., 0.);
G4ThreeVector momdir(0., 0., 1.);
G4double particleKEnergy = 100. * MeV;
fParticleGun->SetParticlePosition(pos);
fParticleGun->SetParticleMomentumDirection(momdir);
fParticleGun->SetParticleEnergy(particleKEnergy);
fParticleGun->SetParticleDefinition(particle);
fParticleGun->GeneratePrimaryVertex(anEvent);
}
// and finally inform our application to draw the trajectory -> dose.cc
```

And the new lines at dose.cc

```cpp
// [...]
// once the action is ready, include it
#include "action.hh"


int main(G4int argc, char** argv) {

// [...]

// once the physics is created, reigster it
runManager->SetUserInitialization(new MyPhysicsList());

// [...]
// after actions ... draw the particle trajectory
UIManager->ApplyCommand("/vis/viewver/set/autorefresh true");
UIManager->ApplyCommand("/vis/scene/add/trajectories smooth");
// for viewing many trajectories together, comment if no needed
UIManager->ApplyCommand("/vis/scene/endOfEventAction accumulate");

// [...]
}
```

It compiles! :)

```
asoreyh@inferno:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ make
[ 20%] Building CXX object CMakeFiles/dose.dir/dose.cc.o
[ 40%] Building CXX object CMakeFiles/dose.dir/action.cc.o
[ 60%] Building CXX object CMakeFiles/dose.dir/construction.cc.o
[ 80%] Building CXX object CMakeFiles/dose.dir/physics.cc.o
[100%] Linking CXX executable dose
[100%] Built target dose
asoreyh@caronte:~/Dropbox/projects/geant4/geant4-course/codes/src/build$ ./dose
```



And it works! :)

And the new lines at dose.cc

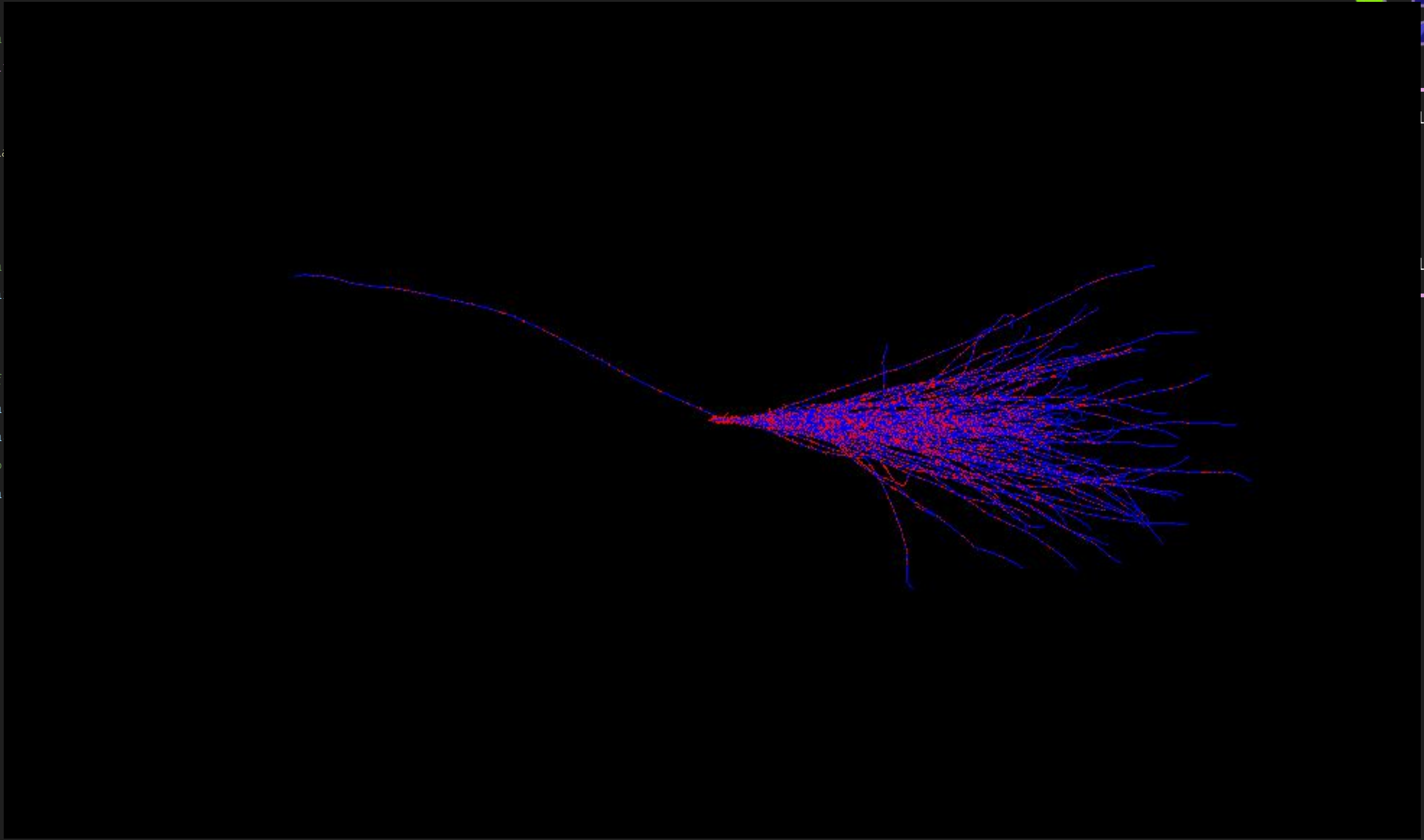
```
// Some useful uiManager commands

UIManager->ApplyCommand("/vis/open OGL");     // open the visualization
UIManager->ApplyCommand("/vis/viewer/set/viewpointVector 1 1 1");  // set view point
UIManager->ApplyCommand("/vis/ogl/set/displayListLimit 50000"); // vertex to be visualized

UIManager->ApplyCommand("/vis/drawVolume"); // draw the volumes
UIManager->ApplyCommand("/vis/viewver/set/autorefresh true"); // draw particle trajectories
UIManager->ApplyCommand("/vis/scene/add/trajectories smooth"); // use smooth traj (magnetic field!)
UIManager->ApplyCommand("/vis/scene/endOfEventAction accumulate 300"); // accumulate events display
```

positive particle **neutral particles** **negative particles**

```
UIManager->ApplyCommand("/vis/modeling/trajectories/create/drawByParticleID"); // use Id, no charge
UIManager->ApplyCommand("/vis/modeling/trajectories/create/drawByParticleID-0/set e- blue");
```

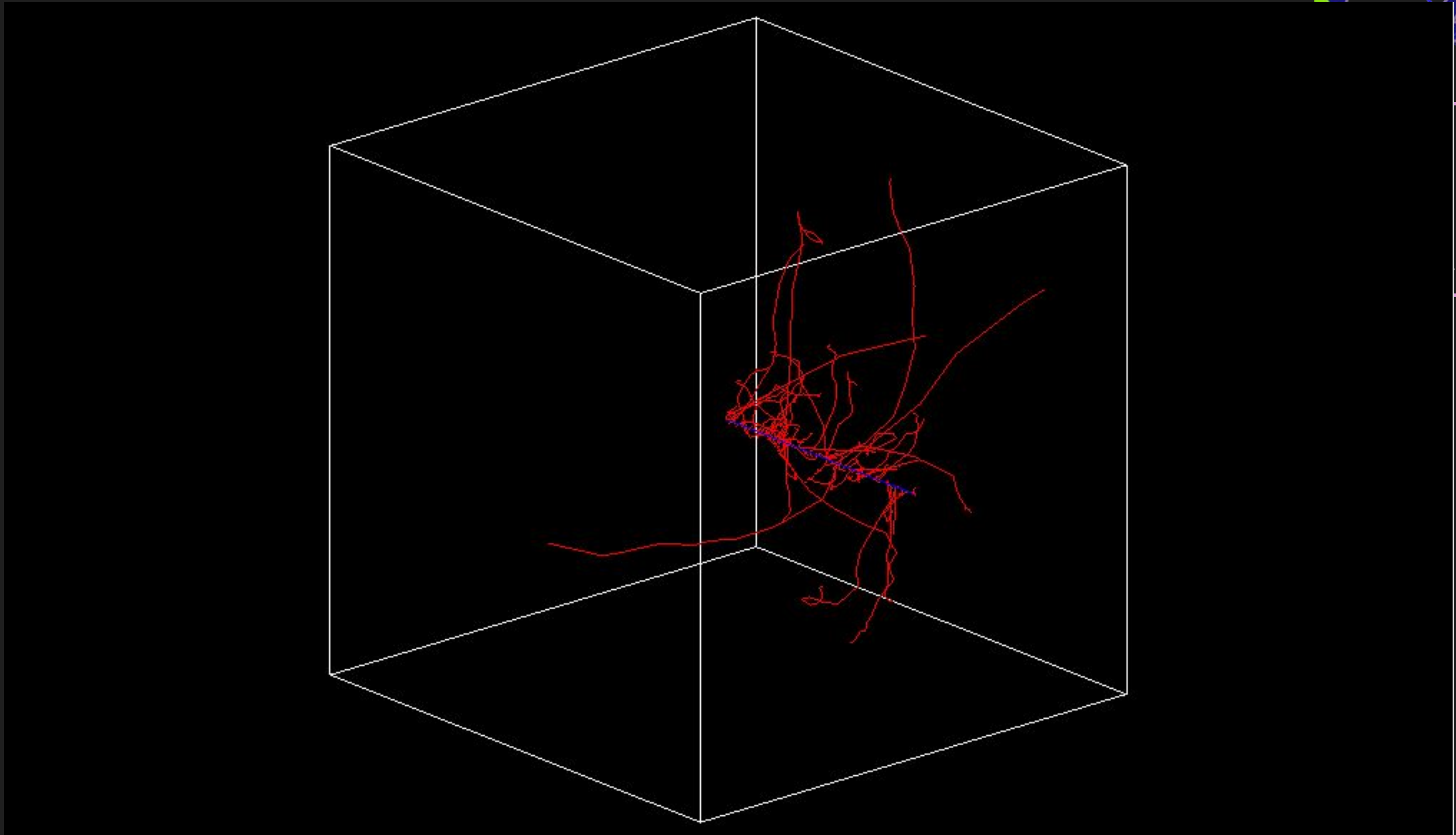
Computational physics is beautiful

```
// [.
// on
#incl

int ma

// [.

// on
runMa

// [.
// af
UIMan
UIMan
// fo
UIMan

// [.
}
```

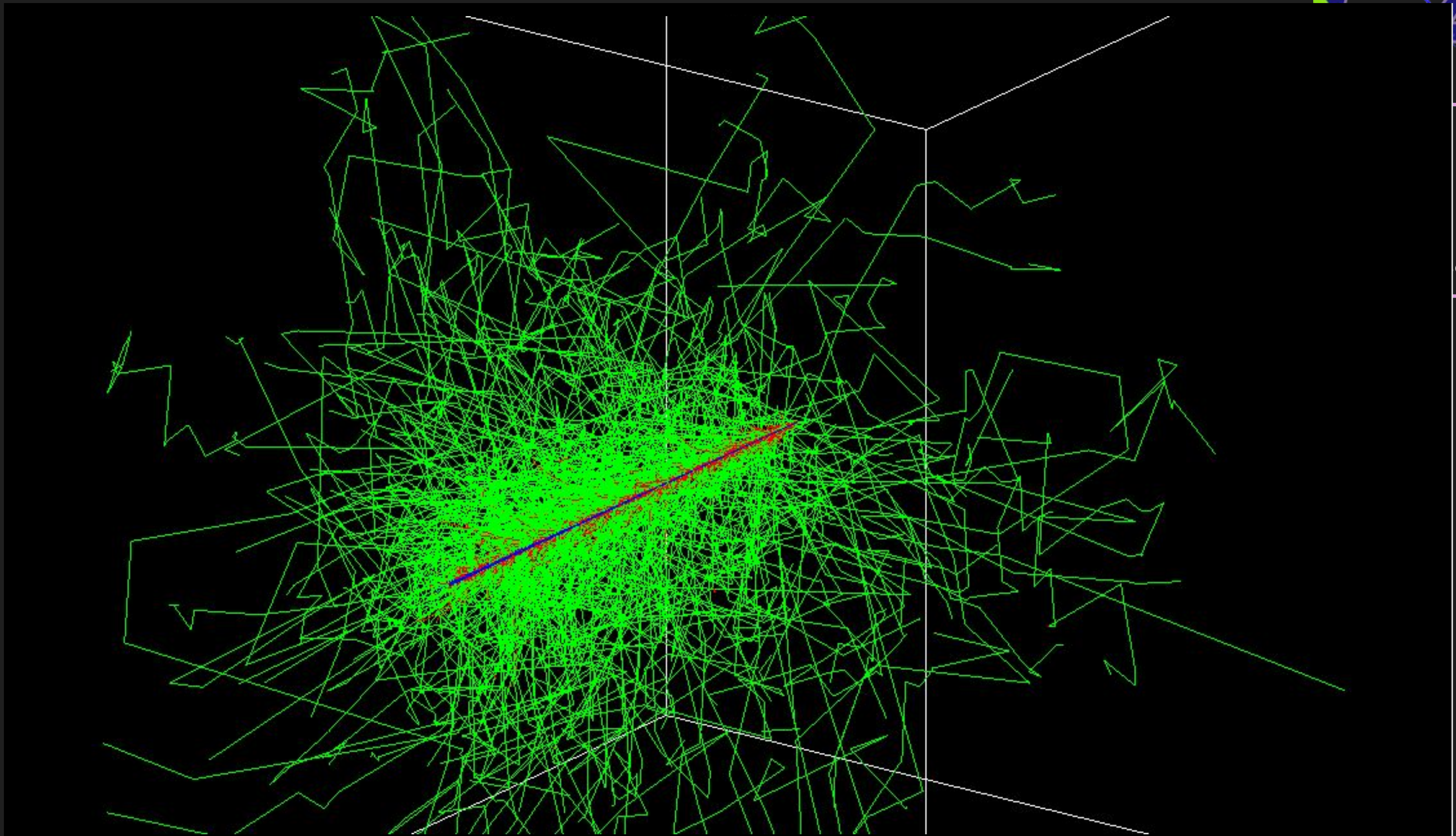


```
ld$ make

ld$ ./dose
```

More energy means more fun (10 GeV)

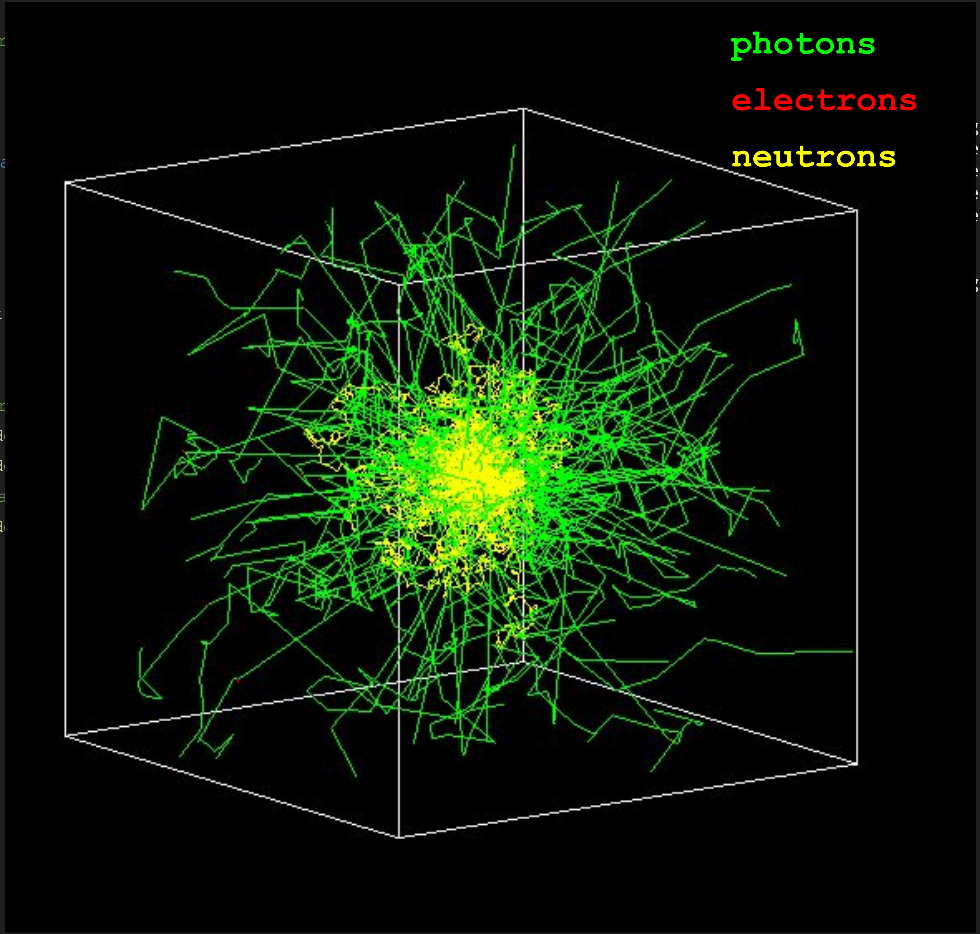


increase density, increase fun (water)



FTFP_BERT_HP, n, 1 MeV in water

```
// [...]
// once the action is r
#include "action.hh"


int main(G4int argc, cha

// [...]

// once the physics is
runManager->SetUserInit

// [...]
// after actions ... dr
UIManager->ApplyCommand
UIManager->ApplyCommand
// for viewing many tra
UIManager->ApplyCommand

// [...]
}
```



:)

```
geant4/geant4-course/codes/src/build$ make
es/dose.dir/dose.cc.o
es/dose.dir/action.cc.o
es/dose.dir/construction.cc.o
es/dose.dir/physics.cc.o

geant4/geant4-course/codes/src/build$ ./dose
```

Wait! Up to now:

We have the basis of an app for simulate many physics applications

- Your main app base code (dose.cc) including:
    - The visManager
    - The uiManager
    - The runManager
        - MyDetectorConstruction
        - MyPhysicsList
        - MyPrimaryGeneratorAction
        - MyActionInitialization

Wait! Before changing all your codes

- Git is a wonderful tool. If you don’t know how to use it please please please learn the basis (see e.g. this official tutorial):

G4GitTutorial(“https://docs.github.com/en/get-started/quickstart/hello-world”);

- I recommend you to create a new branch and work directly on it. So, in your repository, checkout the master branch:

```
$ git checkout -b testing
```

- Here, ‘testing’ is the name I selected for the branch. You can use whatever you prefer. Let’s check if we are in the correct branch

```
$ git branch
master
* testing
```

- For going back to the master branch, just:

```
$ git checkout master
```

Now we are safe, let's code:

- **Before to continue… you MUST play a lot with the different options. You should try, at least:**
    - Different materials (build your own and/or use the NIST DB)
    - Different particles (check the docs)
    - Different energies (see what happens)
    - Different shapes (check the docs)
    - Different physics lists

Now we are safe, let's code:

- Perhaps you introduce a lots of changes in your 'testing' branch
- You can merge them into the master branch or continue developing in your testing branch
- Now we let's start working in the 'final' app. Somethings needs to be done.
    - Copy the 'base' directory to the new 'final' directory

```
$ cd /path/to/apps/
$ cp -r base final
```

    - Simple but wrong way to avoid building issues: change the CMakelist name in the base app.

```
$ cd base
$ mv CMakeLists.txt CMakeLists.off    # use the name you want.
```

Let's define some bio materials and e^-

- We want to construct a new volume (box?) of skeletal muscle

```
G4Material *muscle = nist->FindOrBuildMaterial("G4_MUSCLE_SKELETAL_ICRP");
```

- And add the corresponding solid, logic and physical volume:
  - The 'solid' volume, shapes

```
G4Box *solidMuscle = new G4Box("solidMuscle", 0.025*m, 0.025*m, 0.025*m);
```

  - The 'logic' volume, materials

```
G4LogicalVolume *logicMuscle = new G4LogicalVolume(solidMuscle, muscle, "logicMuscle");
```

  - The 'physics' volume, placement and mother volumes

This is the 'mother' volume

```
G4VPhysicalVolume *physMuscle = new G4PVPlacement(
        0, G4ThreeVector(0., 0., 0.10*m), logicMuscle, "physMuscle", logicWorld, false, 0, true);
```

- Let's use a beam of 10 MeV e^- (action.cc)

```
G4String particleName = "e-";    // m_e = 0.511 MeV
G4double particleKEnergy = 10. * MeV;    // ~20 m_e
```

$$\begin{cases} E^2 = p^2 + m^2 \\ E = \gamma m \\ K = (\gamma - 1)m \end{cases}$$

A comment about doses

- **Absorbed dose** → Physical magnitude, the energy deposited in matter by ionizing radiation per unit mass

$$D = \frac{E_d}{m}$$

unit: gray (Gy) → [D] = Gy = J kg^{-1} (1 rad= 10^2 erg/g = 10^{-2} Gy)

- **Equivalent dose** → stochastic health effects by ionizing radiation R (biological effectiveness, depends on type and E) in a tissue T,

Total equivalent dose in the tissue T

$$H_T = \sum_R w_R D_{R,T}$$

Absorbed dose by the tissue T due to exposure

Sum over all the radiation types and energies involved

Radiation weighting factor (regulations)

- Unit: sievert (Sv) → [H] = Sv = J kg^{-1} (1 rem = 10^{-2} Sv)

A comment about doses

- **Effective dose** → stochastic health risk to the whole body due to radiation exposure. It takes the nature and biological response of each tissue

Tissue weighting factor (regulations)

Sum over all tissues

Equivalent dose absorbed by tissue T

Effective dose

$$E = \sum_T w_T H_T$$

Mass-averaged absorbed dose

$$E = \sum_T w_T \sum_R w_R \overline{D}_{R,T}$$

- Unit: sievert (Sv) → [H] = Sv = J kg^{-1} (1 rem = 10^{-2} Sv)

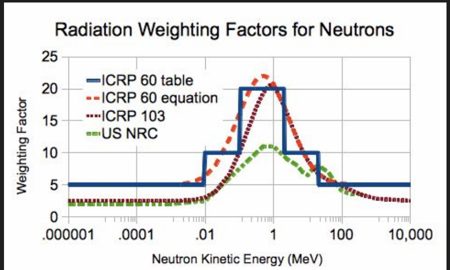
Weighting factors w_R and w_T

Wrixon (2008), doi:10.1088/0952-4746/28/2/R02

Table 2. Recommended radiation weighting factors.

Radiation type	Radiation weighting factor, w_R
Photons	1
Electrons and muons	1
Protons and charged pions	2
Alpha particles, fission fragments, heavy ions	20
Neutrons	A continuous function of neutron energy

$$w_n = \begin{cases} 2.5 + 18.2\exp\left[-\frac{1}{6} ln(E_n)^2\right] & E_n < 1\text{MeV} \\ 5.0 + 17.0\exp\left[-\frac{1}{6} ln(2E_n)^2\right] & 1 \le E_n \le 50\text{MeV} \\ 2.5 + 3.25\exp\left[-\frac{1}{6} ln(0.04E_n)^2\right] & E_n > 50\text{MeV} \end{cases}$$



w_n is still controversial, see, e.g., this presentation

Harrison (2021), doi:10.1088/1361-6498/abe548

Table 1. Summary of ICRP publication 103 nominal cancer risks and detriment for uniform whole-body exposure to gamma rays for the whole population, 0–84 years of age (from table A.4.1, publication 103, Annex A).

Tissue	Nominal risk coefficient (cases per 10 000 persons per Gy)	Detriment	Relative detriment⁺	Tissue weighting factor, w_T
Oesophagus	15	13.1	0.023	0.04
Stomach	79	67.7	0.118	0.12
Colon	65	47.9	0.083	0.12
Liver	30	26.6	0.046	0.04
Lung	114	90.3	0.157	0.12
Bone surface	7	5.1	0.009	0.01
Skin	1000	4.0	0.007	0.01
Breast	112	79.8	0.139	0.12
Ovary	11	9.9	0.017	—[a]
Bladder	43	16.7	0.029	0.04
Thyroid	33	12.7	0.022	0.04
Bone marrow	42	61.5	0.107	0.12
Other solid	144	113.5	0.198	0.12
Gonads (Heritable)	20	25.4	0.044	0.08
Total	1715	574	1.000	1.00[b]

[a] Included in w_T for Gonads.
[b] Brain and Salivary glands also each assigned $w_T = 0.01$.

Weighting factors w_R and w_T

Wrixon (2008), doi:10.1088/0952-4746/28/2/R02

Harrison (2021), doi:10.1088/1361-6498/abe548

Table 2. Recommended radiation weighting factors.

Radiation type	Radiation weighting factor, w_R
Photons	
Electrons and muons	
Protons and charged	
Alpha particles, fiss	
Neutrons	

$$w_n = \begin{cases} 2.5 + \\ 5.0 + \\ 2.5 + \end{cases}$$

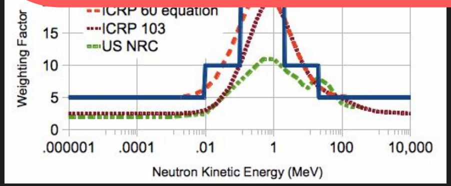


Table 1. Summary of ICRP publication 103 nominal cancer risks and detriment for uniform whole-body exposure to gamma rays for the whole population, 0–84 years of age (from table A.4.1, publication 103, Annex A).

				Tissue weighting factor, w_T
				0.04
				0.12
				0.12
				0.04
				0.12
				0.01
				0.01
				0.12
				—[a]
				0.04
				0.04
Bone marrow	42	61.5	0.107	0.12
Other solid	144	113.5	0.198	0.12
Gonads (Heritable)	20	25.4	0.044	0.08
Total	1715	574	1.000	1.00[b]

[a] Included in w_T for Gonads.
[b] Brain and Salivary glands also each assigned $w_T = 0.01$.

So, for calculating the absorbed dose, we need to get the deposited energy in a certain volume and its mass!

$$D = \frac{E_d}{m}$$

We need to know how determine the deposited energy

w_n is still controversial, see, e.g., this presentation

Weighting factors w_R and w_T

Wrixon (2008),
doi:10.1088/0952-4746/28/2/R02

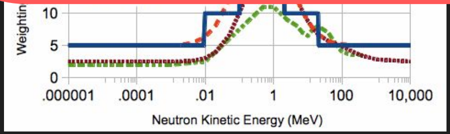
Harrison (2021),
doi:10.1088/1361-6498/abe548

Table 2. Recomm

Radiation type
Photons
Electrons and
Protons and c
Alpha particle
Neutrons

$w_n = \{ 2 \ldots, 5 \ldots, 2 \ldots$

Let's Think on how the total deposited energy can be obtained. What do we need to do?



...s and detriment for uni-
...ion, 0–84 years of age

				Tissue weighting factor, w_T
				0.04
				0.12
				0.12
				0.04
				0.12
				0.01
				0.01
				0.12
				—[a]
				0.04
				0.04
			...7	0.12
			...198	0.12
Gonads (Heritable)	20	25.4	0.044	0.08
Total	1715	574	1.000	1.00[b]

[a] Included in w_T for Gonads.
[b] Brain and Salivary glands also each assigned $w_T = 0.01$.

w_n is still controversial, see, e.g., this presentation

Weighting factors w_R and w_T

Wrixon (2008),
doi:10.1088/0952-4746/28/2/R02

Harrison (2021),
doi:10.1088/1361-6498/abe548

Table 2. Recomm

Radiation type

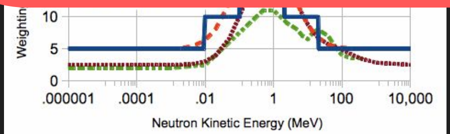
Photons
Electrons and
Protons and c
Alpha particle
Neutrons

$w_n = \begin{cases} 2 \\ 5 \\ 2 \end{cases}$

Let's Think on how the total deposited energy can be obtained. What do we need to do?

To do that, we need to include methods that allow us to perform "user" actions when:

- **G4UserRunAction**: the **run** starts/end
- **G4UserEventAction**: the **event** starts/end
- **G4UserStepAction**: the **step** starts/end



...ks and detriment for uni-
...ion, 0–84 years of age

				Tissue weighting factor, w_T
				0.04
				0.12
				0.12
				0.04
				0.12
				0.01
				0.01
				0.12
				—[a]
				0.04
				0.04
			...7	0.12
			...198	0.12
Gonads (Heritable)	20	25.4	0.044	0.08
Total	1715	574	1.000	1.00[b]

[a] Included in w_T for Gonads.
[b] Brain and Salivary glands also each assigned $w_T = 0.01$.

w_n is still controversial, see, e.g., this presentation

G4UserRunAction (and Event and Step)

- Allow tp get control before/after a run. Provides a run-object (G4Run)
  - It is needed when you want to take actions before/after the run starts/ends (before first event processing/after last event processed)

- Imagine you want to store all the hits in a certain volume (e.g. detector)

- In our case, for the sake of completitude, let’s use a root file instead of an ASCII (text) file

- We shall use the g4root standalone libraries for not depending on root installation

```
#include "g4root.hh"
```

- Done. Now, all your root knowledge can be included in your app
- **G4UserEventAction** and **G4UserRunAction** are similar but acting for **events** and **steps**

There we go… run.hh/.cc

```cpp
#ifndef RUN_HH
#define RUN_HH

#include "G4UserRunAction.hh"
#include "g4root.hh"  //always! root geant4 standalone
libraries

class MyRunAction : public G4UserRunAction {
public:
   MyRunAction();
   ~MyRunAction();

   virtual void BeginOfRunAction(const G4Run*);
   virtual void EndOfRunAction(const G4Run*);
};

#endif
```

```cpp
#include "run.hh"
MyRunAction::MyRunAction() {}
MyRunAction::~MyRunAction() {}

void MyRunAction::BeginOfRunAction(const G4Run*){
   G4AnalysisManager *root = G4AnalysisManager::Instance();
   root->OpenFile("doses.root");
   // create the NTuple
   root->CreateNtuple("doses", "doses");
   // information to be stored in columns
   // root->CreateNtupleIColumn("fEvent");
   root->CreateNtupleDColumn("fEDep");
   // root->CreateNtupleDColumn("fmass");
   // root->CreateNtupleDColumn("fAbsDose");
   root->FinishNtuple(0);    // close the NTuple
}

void MyRunAction::EndOfRunAction(const G4Run*){
   G4AnalysisManager *root = G4AnalysisManager::Instance();
   root->Write(); // always write before to close
   root->CloseFile();
}
```

G4UserEventAction (new files: event.hh/.cc)

```cpp
#ifndef EVENT_HH
#define EVENT_HH

#include "G4UserEventAction.hh"
#include "G4Event.hh"
#include "g4root.hh"
#include "run.hh"

class MyEventAction : public G4UserEventAction {
public:
   MyEventAction(MyRunAction*);
   ~MyEventAction();

   virtual void BeginOfEventAction(const G4Event*);
   virtual void EndOfEventAction(const G4Event*);

   void AddEDep(G4double EDep);

private:
   G4double fEDep;
};

#endif
```

```cpp
#include "event.hh"
MyEventAction::MyEventAction(MyRunAction*) {
   fEDep = 0.;
}

MyEventAction::~MyEventAction(){}

void MyEventAction::BeginOfEventAction(const G4Event*) {
   // we want to get the deposited energy for each event, then
   // we need to put it to 0. each time the event starts.
   // Otherwise, we will get the total accumulated Ed (later)
   // comment if you want the total accumulated energy
   // first test in an event basis
   fEDep = 0.;
}

void MyEventAction::EndOfEventAction(const G4Event*) {
   G4cout << "Energy deposition: " << fEDep << G4endl;
   G4AnalysisManager *root = G4AnalysisManager::Instance();
   root->FillNtupleDColumn(0, fEDep);
   root->AddNtupleRow(0);
}

void MyEventAction::AddEDep(G4double EDep) {
   fEDep += EDep;
}
```

G4UserStepAction (new files: stepping.hh/.cc)

```
#ifndef STEPPING_HH
#define STEPPING_HH

#include "G4UserSteppingAction.hh"
#include "G4Step.hh"

#include "construction.hh"
#include "event.hh"

class MySteppingAction : public G4UserSteppingAction {
public:
   MySteppingAction(MyEventAction *eventAction);
   ~MySteppingAction();

   virtual void UserSteppingAction(const G4Step*);

private:
   MyEventAction *fEventAction;
};
#endif
```

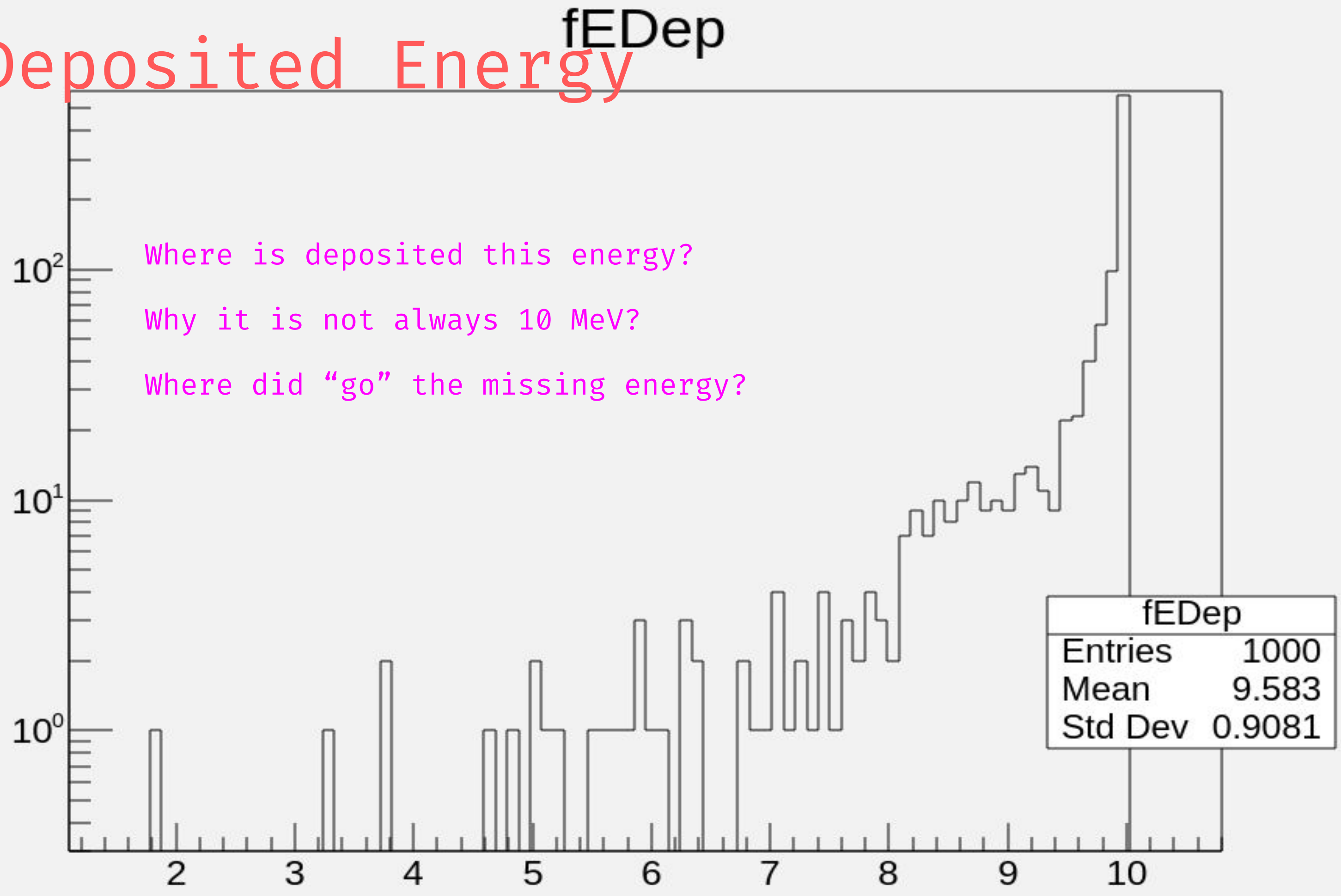
```
#include "stepping.hh"

MySteppingAction::MySteppingAction(MyEventAction *eventAction){
   fEventAction = eventAction;
}

MySteppingAction::~MySteppingAction() {}

void MySteppingAction::UserSteppingAction(const G4Step *step) {
   // deposited energy is stored at each step
   G4double stepEDep = step->GetTotalEnergyDeposit(); // this is
for all volumes
   fEventAction->AddEDep(stepEDep);
}
```

Deposited Energy

Where is deposited this energy?

Why it is not always 10 MeV?

Where did “go” the missing energy?

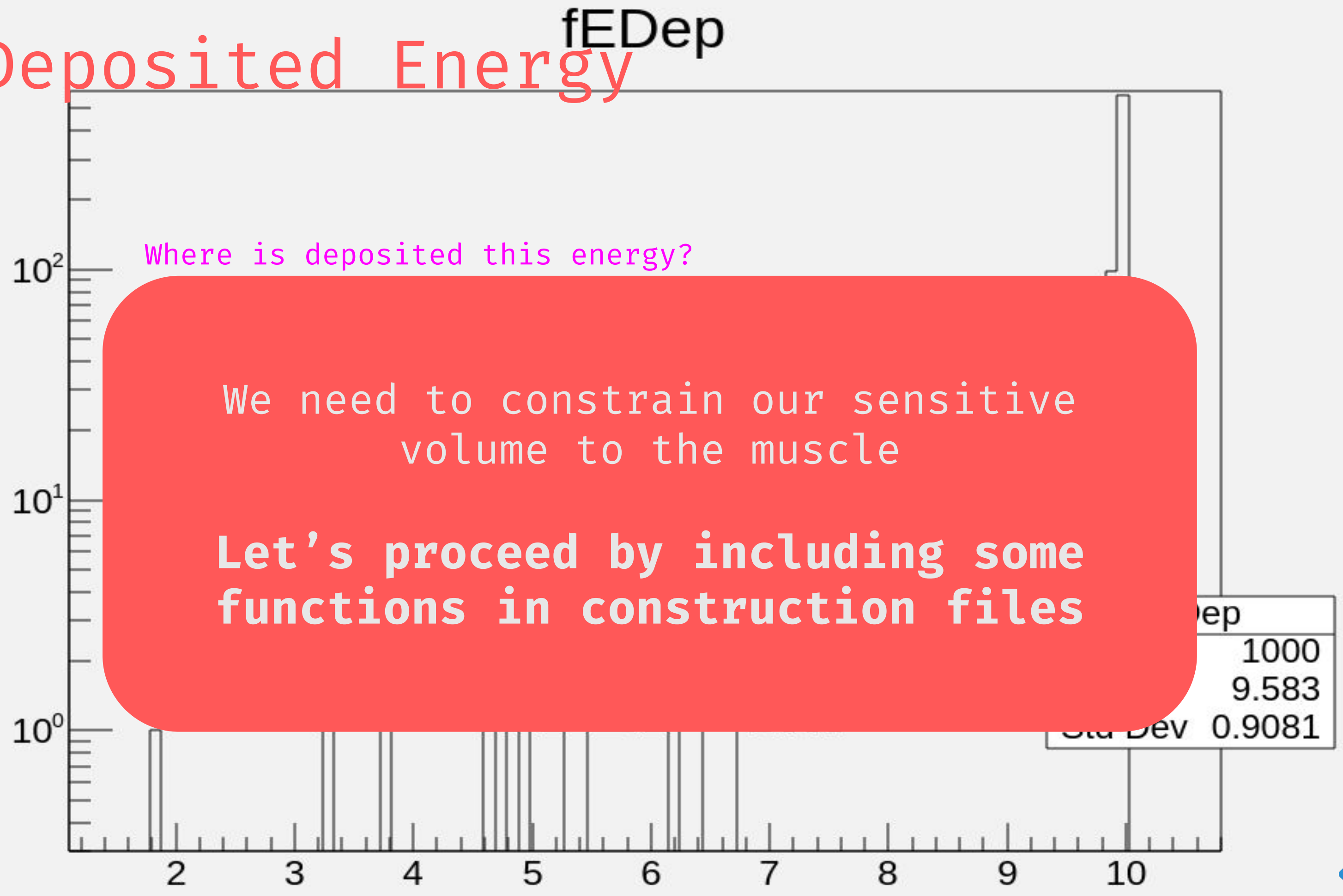


Deposited Energy

Where is deposited this energy?

We need to constrain our sensitive volume to the muscle

Let’s proceed by including some functions in construction files



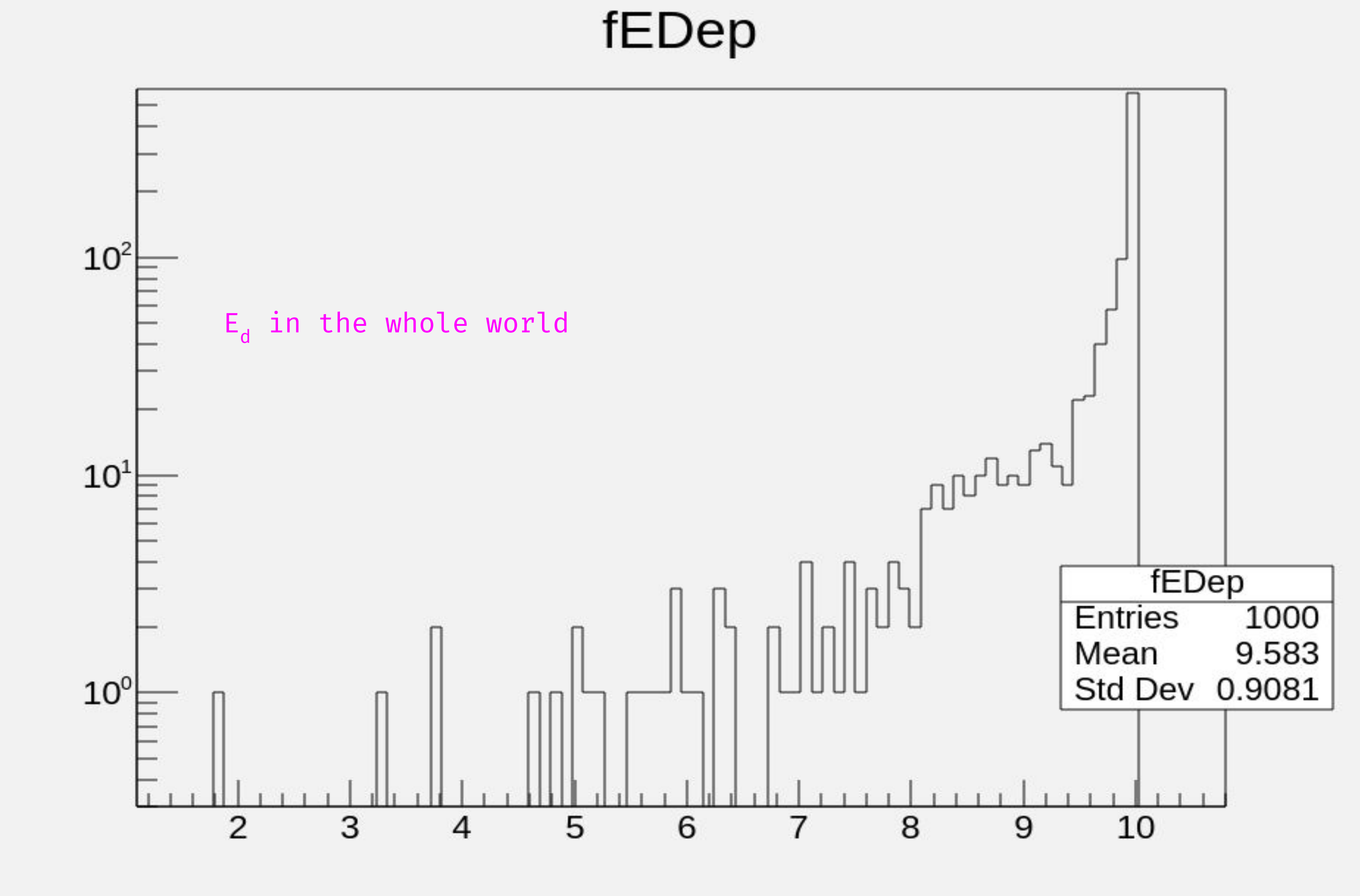
We need to know if the particle is in the sensitive vol

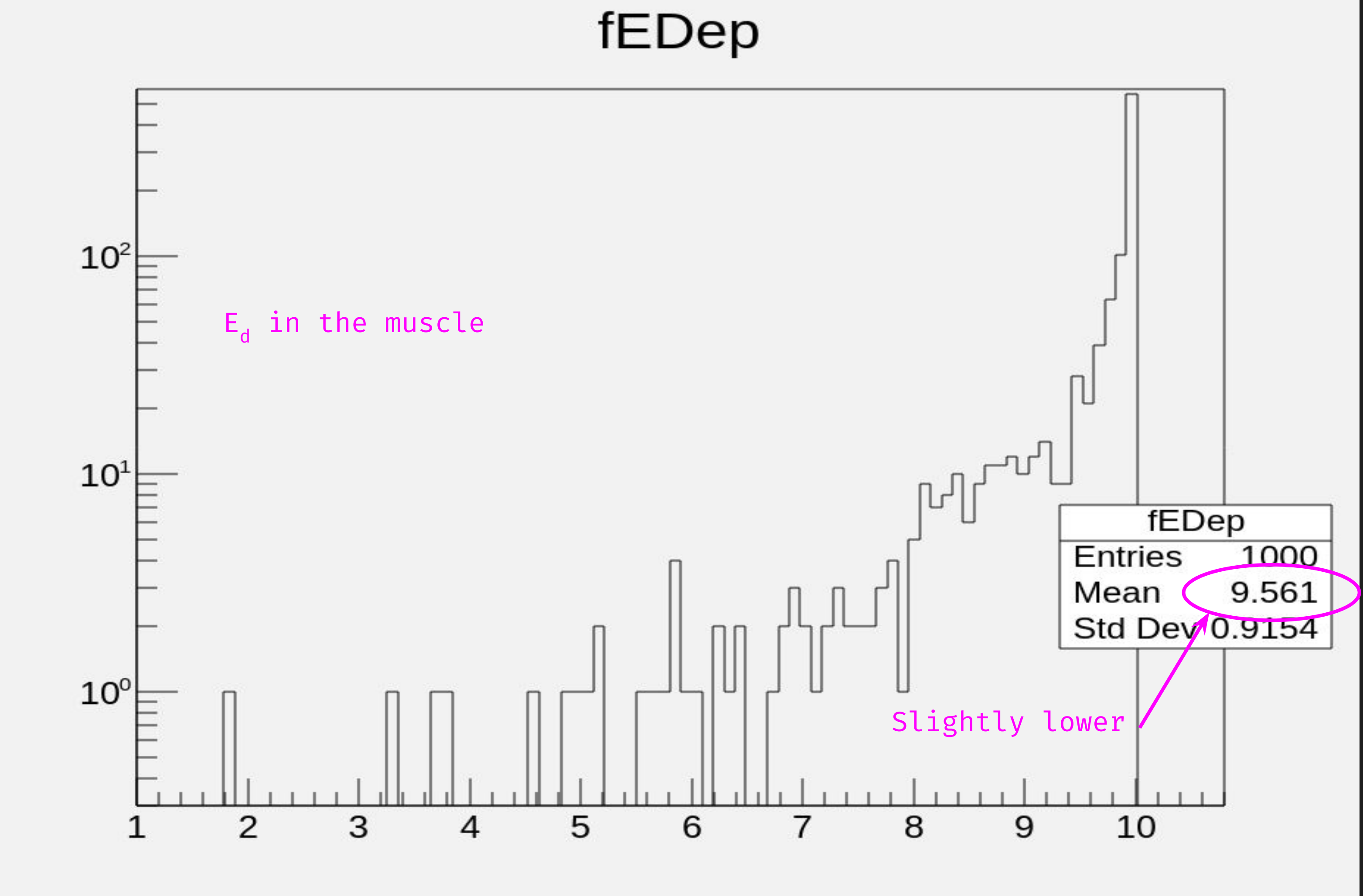
- At 'construction' we need to define our sensitive volume, and we need to build a function to export this logical volume

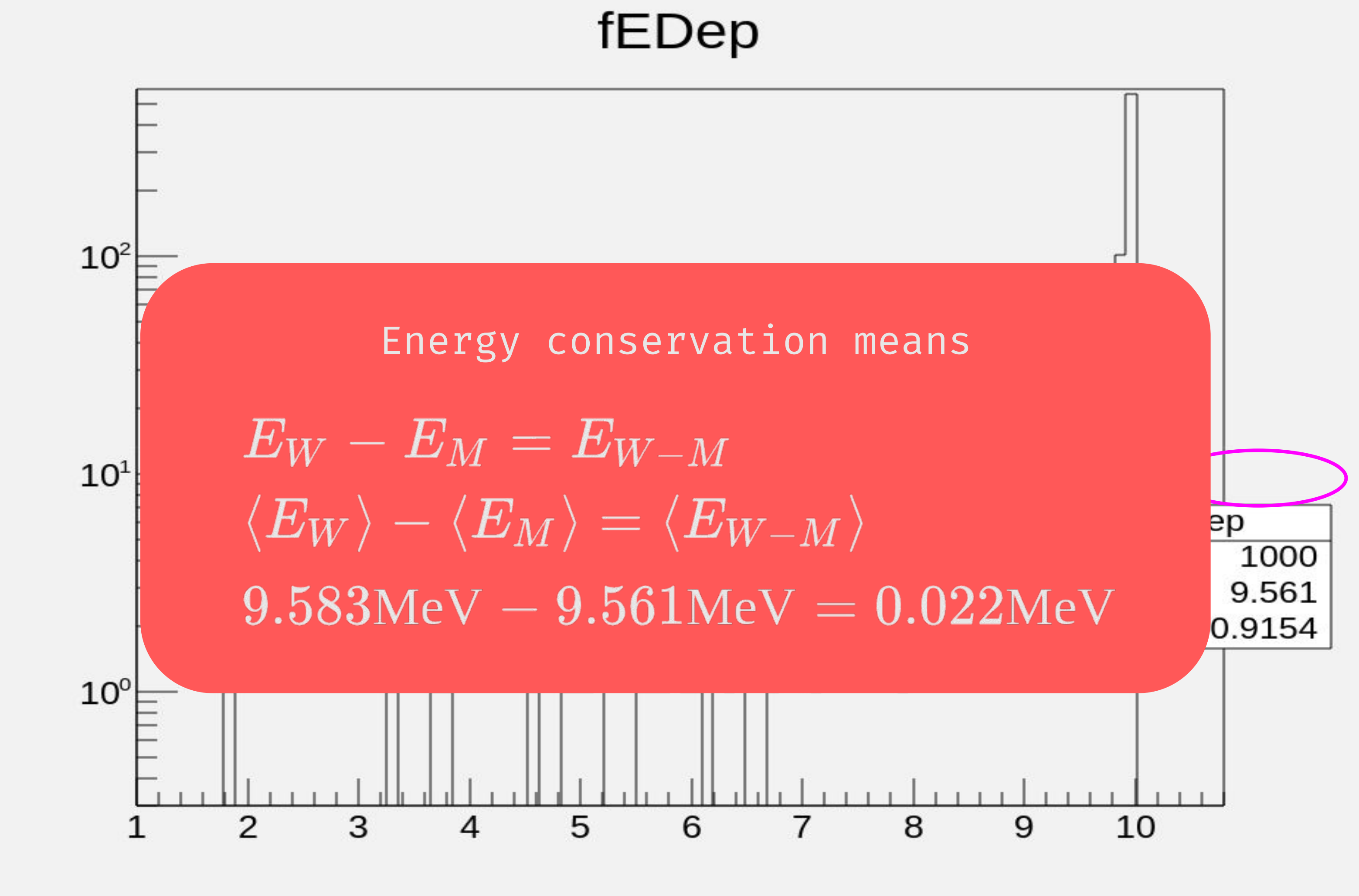
```
// in the class definition
private:
      G4LogicalVolume *fSensitiveVolume;
// in the constructor definition
G4LogicalVolume *GetSensitiveVolume() const { return fSensitiveVolume;}
// define the logical volume in the construction.cc
fSensitiveVolume = logicMuscle;
```

- Now, all the information about the current position of the particle is in stepping

```
// *volume is the LogicalVolume where the particle is located
G4LogicalVolume *volume =
      step->GetPreStepPoint()->GetTouchableHandle()->GetVolume()->GetLogicalVolume();
// we need  to get an object including the volumes we constructed
const MyDetectorConstruction *detectorConstruction = static_cast<const
      MyDetectorConstruction*>(G4RunManager::GetRunManager()->GetUserDetectorConstruction());
// and *fSensitiveVolume is our selected sensitive volume
G4LogicalVolume *fSensitiveVolume = detectorConstruction->GetSensitiveVolume();
```

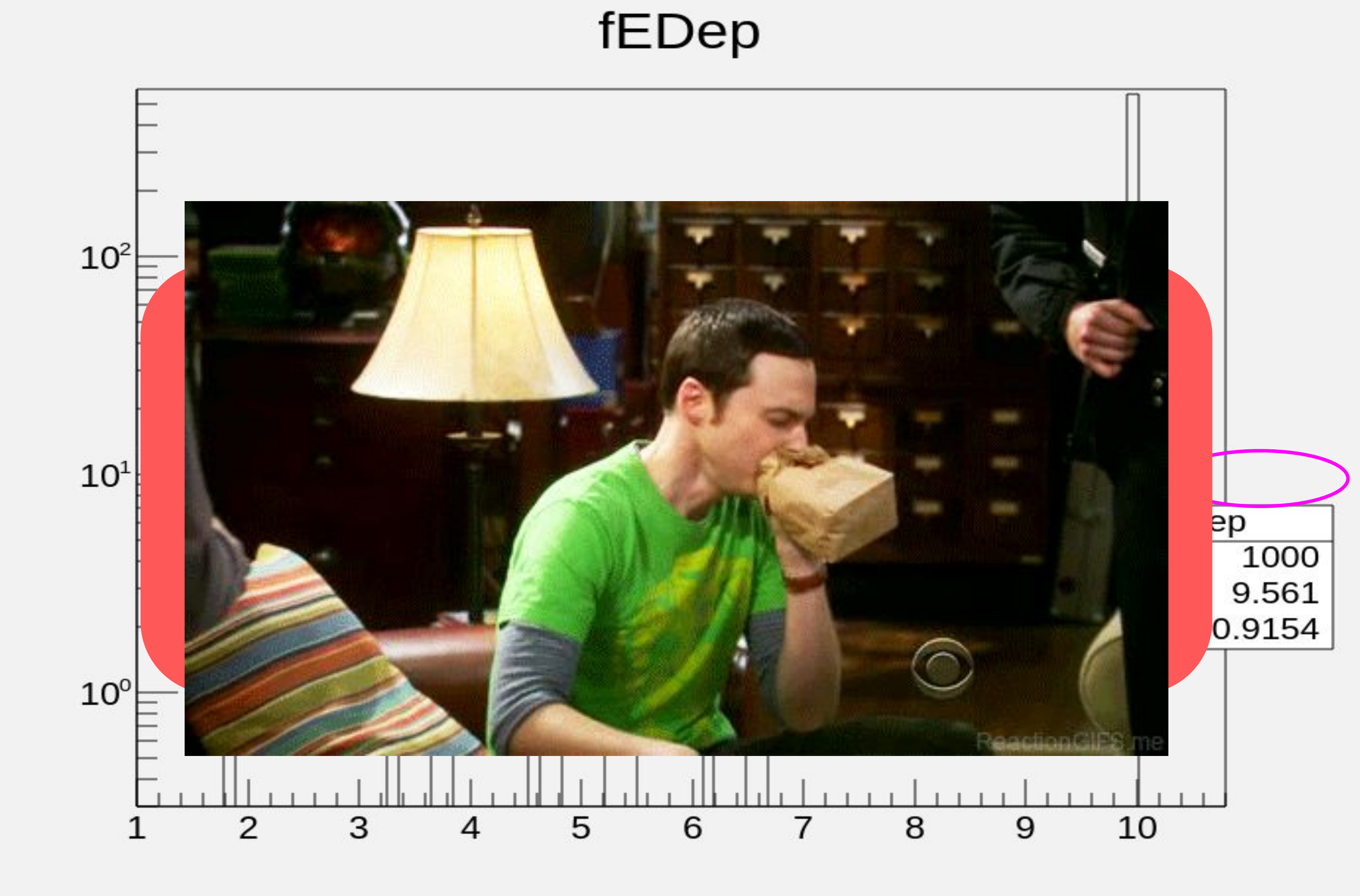

fEDep
E_d in the whole world
fEDep
Entries 1000
Mean 9.583
Std Dev 0.9081
10^2
10^1
10^0
2 3 4 5 6 7 8 9 10

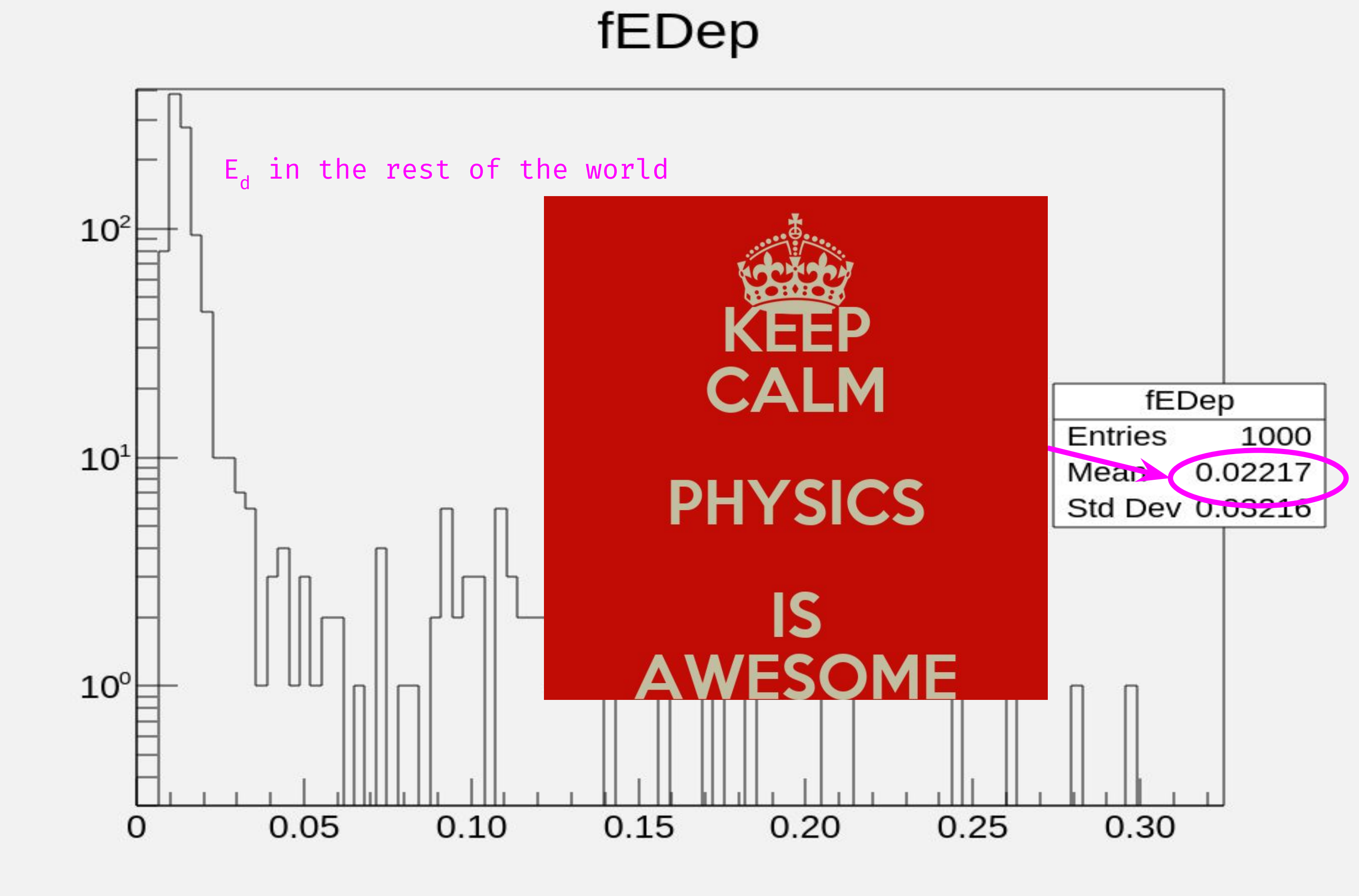

fEDep
10^2
10^1
10^0
1 2 3 4 5 6 7 8 9 10
E_d in the muscle
fEDep
Entries 1000
Mean 9.561
Std Dev 0.9154
Slightly lower



Energy conservation means

$$E_W - E_M = E_{W-M}$$
$$\langle E_W \rangle - \langle E_M \rangle = \langle E_{W-M} \rangle$$
$$9.583\mathrm{MeV} - 9.561\mathrm{MeV} = 0.022\mathrm{MeV}$$


fEDep
10²
10¹
10⁰
1
2
3
4
5
6
7
8
9
10
ep
1000
9.561
0.9154


fEDep
E_d in the rest of the world
KEEP
CALM
PHYSICS
IS
AWESOME
fEDep
Entries 1000
Mean 0.02217
Std Dev 0.03216
10^2
10^1
10^0
0
0.05
0.10
0.15
0.20
0.25
0.30

Success! Now, D=fEDep/sensitiveVolumeMass

- There are several ways to do this. As this is a very beginner course, let's use a practical but no so efficient method
- For other ways, check the 'B1example' included in the Geant4 source directory

- First we need to introduce a new method and attribute at MyEventAction class

```
void GetMass(G4double mass);

G4double fMass;
```

- Then we need to report the mass to the user eventAction (in stepping.cc)

```
fEventAction->GetMass(fMass);
```

- Finally, get the mass at every step (in stepping.cc) (very inefficient!)

```
G4double fMass = fSensitiveVolume->GetMass() / kg; // This is the correct way to deal with units
```

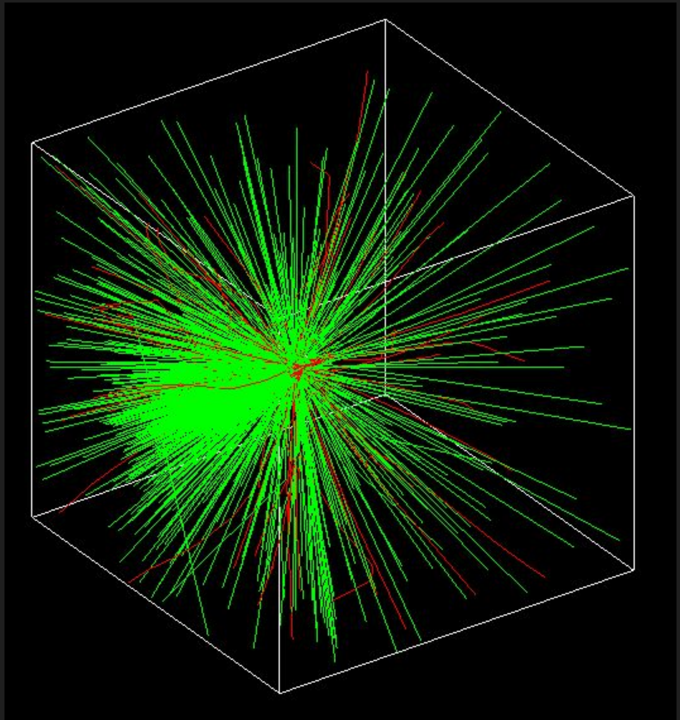
- And just calculate and accumulate the dose at the end of every event (event.cc)

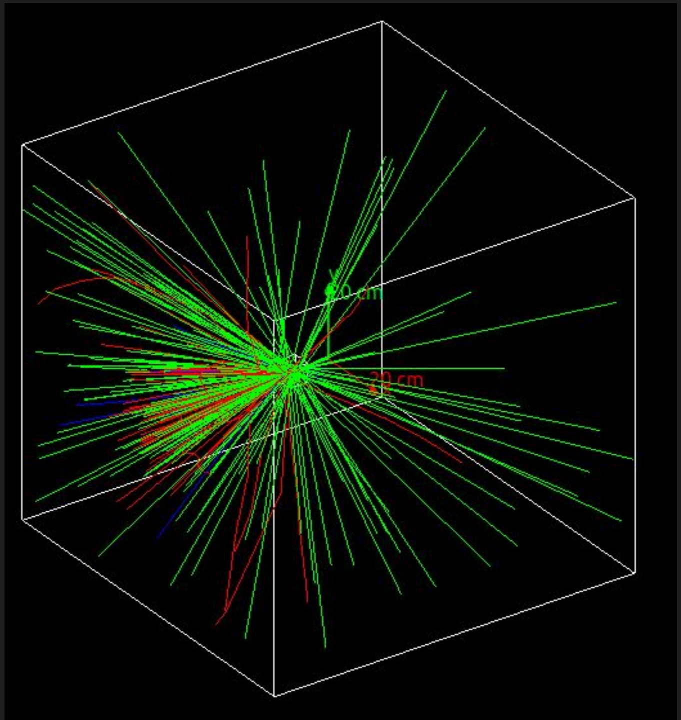
```
G4double fAbsDose = 0.;
if (fMass > 0)
     fAbsDose = (fEDep / joule) / fMass;  //  This is the correct way to deal with units
```

We are done my friends

1000 e^- 10 MeV
Energy deposition: 9560.96 MeV.
Mass: 0.13125 kg.
Dose: 1.16711e-08 Gy. → 11.7 nGy

1000 photons 10 MeV
Energy deposition: 535.392 MeV
Mass: 0.13125 kg
Dose: 6.90615e-10 Gy → 0.690 nGy





Wait! Before you go, remember the shielding?

- How can we evaluate the shielding effect?
- Let's try

Wait! Before you go, remember the shielding?

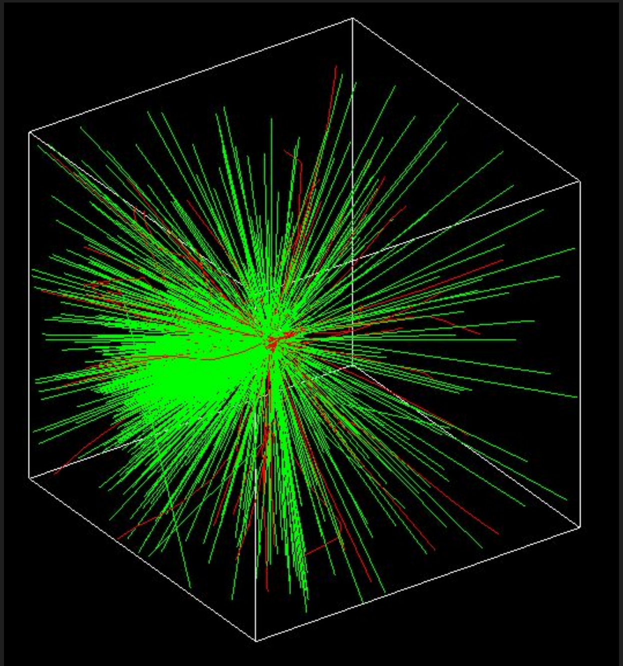
- How can we evaluate the shielding effect?

- Let's give it a try

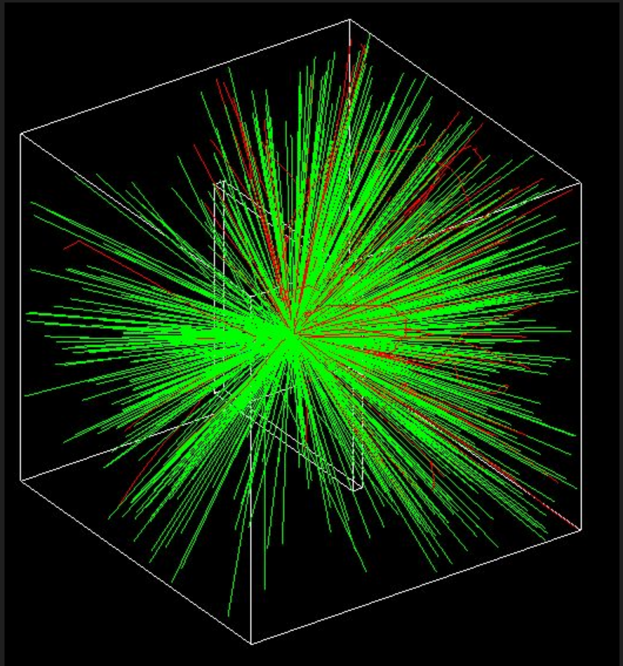
```
G4Material *lead = nist->FindOrBuildMaterial("G4_Pb");

G4Box *solidShield = new G4Box ("solidShield", 0.30*m, 0.30*m, 0.015*m);
G4LogicalVolume *logicShield = new G4LogicalVolume(solidShield, lead, "logicShield");
G4VPhysicalVolume *physShield = new G4PVPlacement(0, G4ThreeVector(0.*m, 0.*m, 0.0375*m),
logicShield, "physShield", logicWorld, false, 0, true);
```

remember the shielding, electrons

1000 e^- 10 MeV
no shielding
Energy deposition: 9560.96 MeV
Mass: 0.13125 kg.
Dose: 1.16711e-08 Gy. → 11.7 nGy

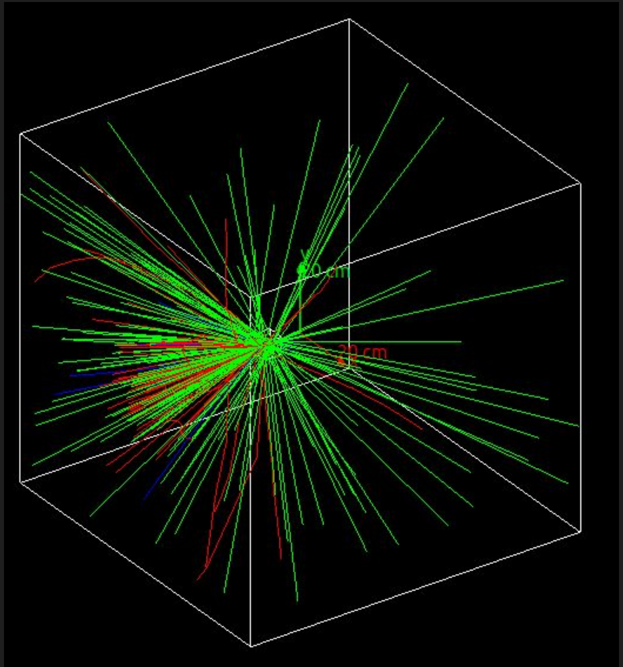


1000 e^- 10 MeV
lead, 3cm
Energy deposition: 28.70 MeV
Mass: 0.13125 kg.
Dose: 3.05302e-11 Gy. → 30.5 pGy
Shield effect: ~1/400

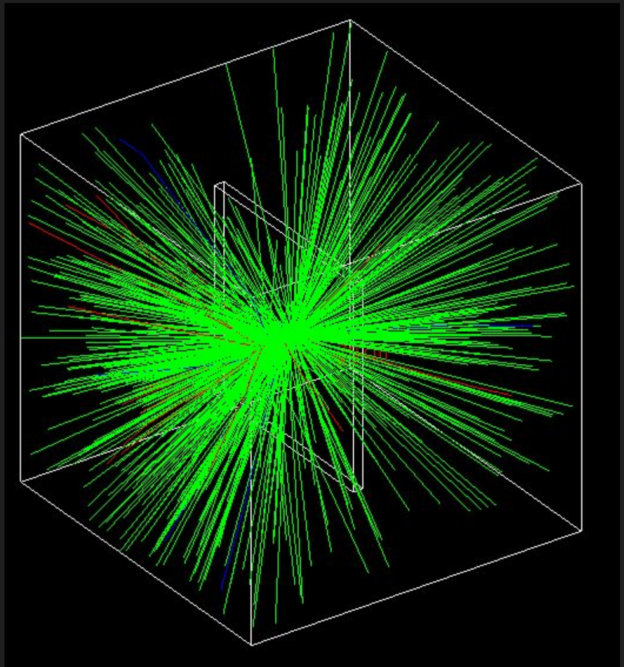


remember the shielding, photons

1000 photons 10 MeV
no shielding
Energy deposition: 535.392 MeV
Mass: 0.13125 kg
Dose: 6.90615e-10 Gy → 690 pGy



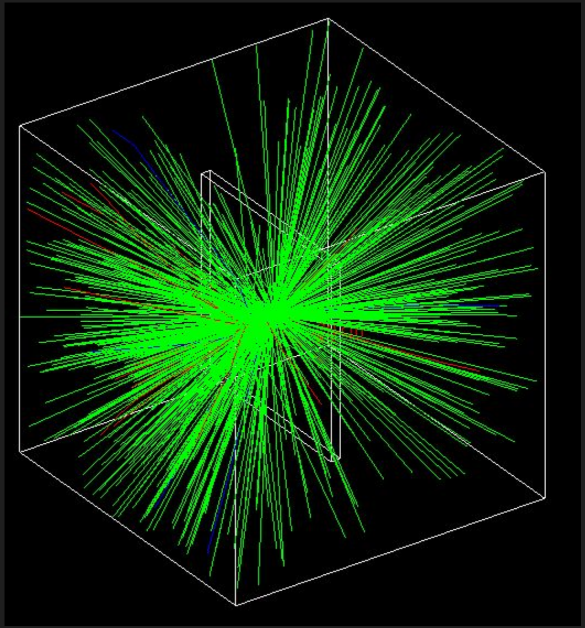
1000 photons 10 MeV
Lead, 3cm
Energy deposition: 168.04 MeV
Mass: 0.13125 kg
Dose: 2.0513e-10 → 205 pGy
Shield effect: 1/3.4



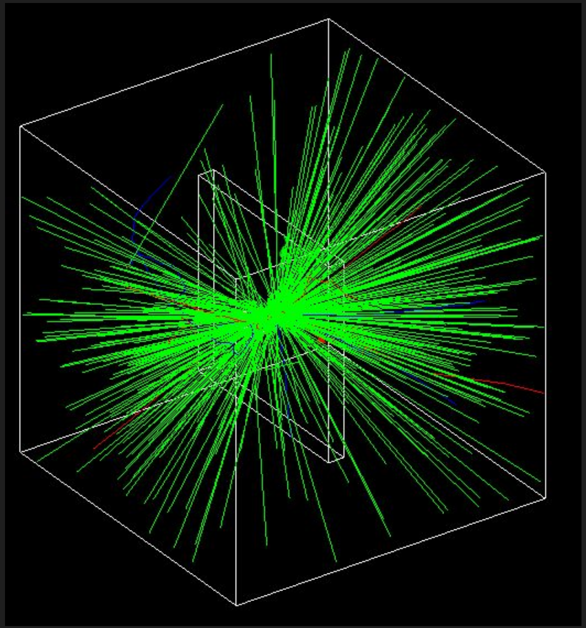
Wait! Before you go, remember the shielding?

Original dose (no shielding): 690 pGy

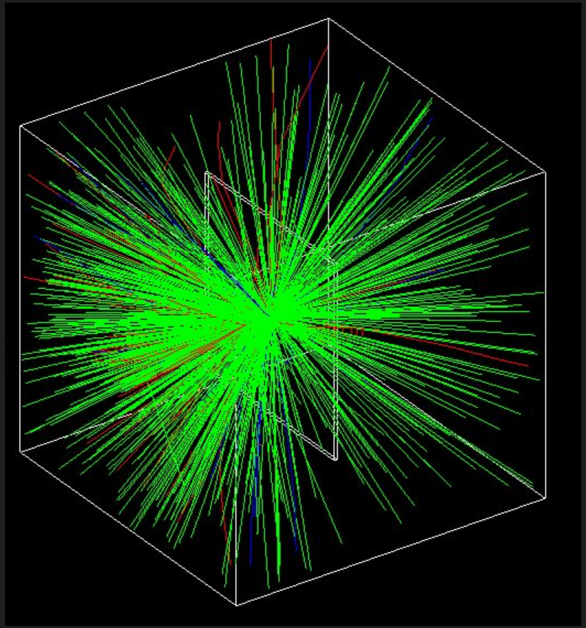
1000 photons 10 MeV
Lead, 3 cm
Energy deposition: 168.042 MeV
Mass: 0.13125 kg
Dose: 2.0513e-10 Gy → 205 pGy



1000 photons 10 MeV
Lead, 5cm
Energy deposition: 62.77 MeV
Mass: 0.13125 kg
Dose: 7.66311e-11 Gy → 76.6 pGy



1000 photons 10 MeV
Lead, 1cm
Energy deposition: 382.918 MeV
Mass: 0.13125 kg
Dose: 4.6743e-10 Gy → 467 pGy



Conclusive remarks, 1

- Together we have developed:
  - a simple, not so efficient, but **complete Geant4 application**
  - it is a powerful tool for performing Geant4 simulations
  - it is a nice **template for building your own applications**

Conclusive remarks, 2

- This course is based on several Geant4 courses available online.

- My recommendation is to read the all the docs and check at least:

    - https://www.youtube.com/playlist?list=PLLybgCU6QCGWgzNYOV0SKen9vqg4KXeVL ← YouTube
    - https://github.com/mnovak42/Geant4-Beginner-Course/tree/master ← GitHub
    - https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/BackupVersions/V10.7/html/index.html ← Official guide for G4 apps developers

I acknowledge Mustafa Schmidt (@physics_matters) and Mihaly Novak for their wonderful resources and courses

Conclusive remarks, 3

I am hoping to continue developing my Geant4 course and surely I will introduce changes in the future.

I will tag this version of the course at GitHub for your future reference (Think in a repository as a movie. A **tag** is picture taken at a certain moment of the development)

Homework, sorry, you also have to work ;-)

- Important homework
  - Play a lot with these codes. Try different geometries, materials, particle beams, energies, and physics lists.
  - Compare the changes and take note of your observations
  - Analyze the effect of different shielding materials, thickness and positions.
  - Try to build composite shieldings (thin layers stacked of different materials)
  - Reproduce the findings on the theoretical courses here: stopping power, particle ranges, mass absorption, …
- Official homework
  - Calculate the energy deposited and the absorbed dose in a 3 cm cube of adipose tissue (G4_ADIPOSE_TISSUE_ICRP) when irradiated with a 5,000 gamma beam with E=7 MeV.
  - Calculate the effect on the energy deposited and the absorbed dose in the same tissue when using a 2cm thick lead shield (G4_Pb).

Take home messages

- Geant4 is a wonderful toolsuite for simulate the interaction of radiation with matter.

- IMHO, G4 analysis tools are not so good, so it I recommend to produce root/csv outputs, and analyze them in root/python correspondingly

- IMHO, developing a new expertise require:

1. A simple but functional example
2. Understand the jargon

this course

3. Google (or Bard/ChatGPT)

freely available

4. **Eager to learn**

it depends on you

It was nice to share this course with you.


```
// "Geant4 for Beginners: a crash course" ended here, so
delete course;
```

Thanks for participate!
Hope to see you soon!

@asoreyh