GEANT4 – A MONTE CARLO SIMULATION TOOLKIT PART II

F. García

Helsinki Institute of Physics

09.06.2021

Latin American alliance for Capacity buildiNG in Advanced physics

LA-CoNGA physics

Cofinanciado por el programa Erasmus+ de la Unión Europea

OUTLINE

- WHAT IS GEANT4
- MONTECARLO METHOD IN PARTICLE TRANSPORT
- A BIT ABOUT C++
- DEFINITION OF A SIMULATION MODEL
- MAIN COMPONENTS OF A MODEL
- GEANT4 - Hands-on

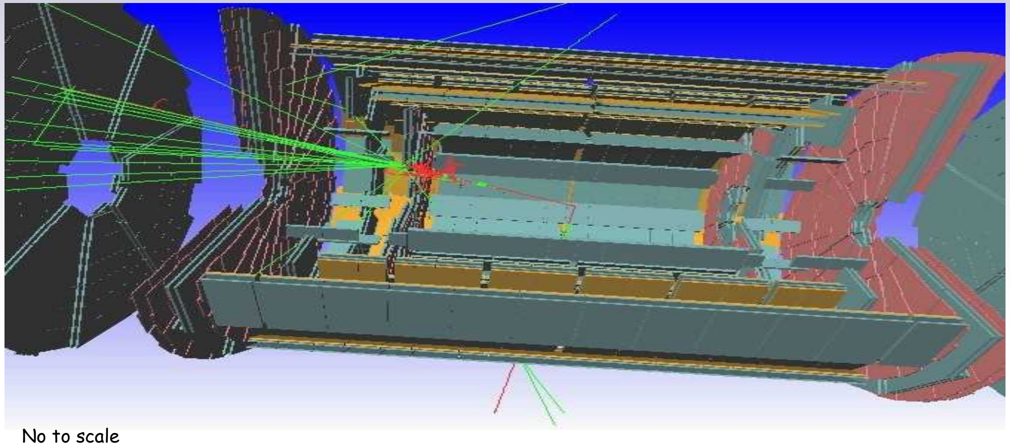
Latin American alliance for Capacity buildiNG in Advanced physics
LA-CoNGA physics

Cofinanciado por el programa Erasmus+ de la Unión Europea

WHAT IS GEANT4

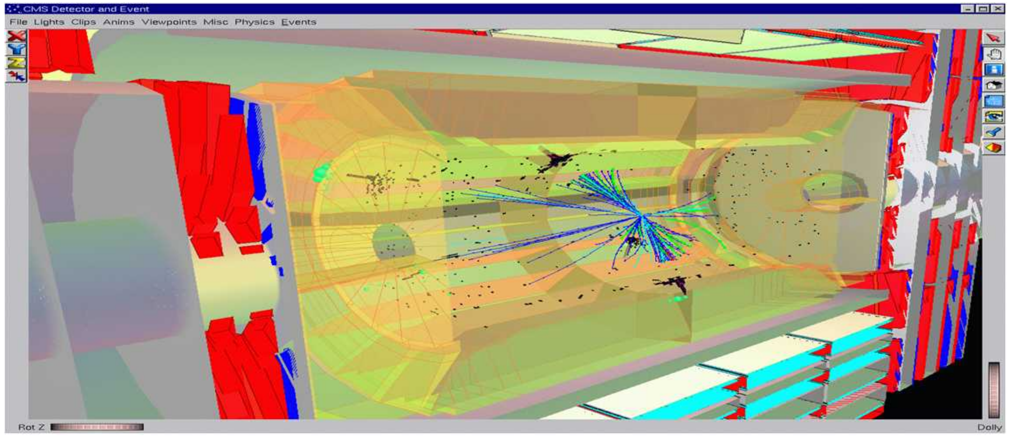
Geant4 is a toolkit for the simulation of the passage of particles through matter. Its areas of application include high energy, nuclear and accelerator physics, as well as studies in medical and space science

ATLAS experiment



No to scale

CMS experiment

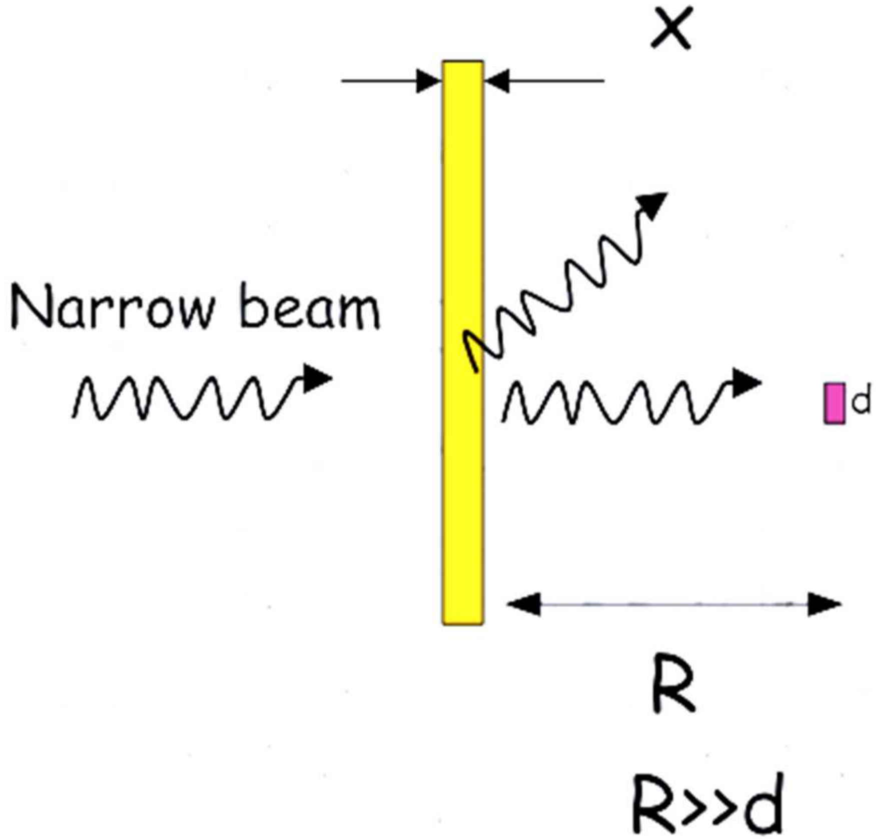


No to scale

$$I\ (x) = I_0 \cdot e^{-\mu x}$$

The probability that normally incident photon will reach the depth **x** in a material slab without interaction is:

$$P(x) = e^{-\mu x}$$



A BIT ABOUT C++

BABAR C++ Course

Paul F. Kunz

Stanford Linear Accelerator Center

No prior knowledge of C assumed

I'm not an expert in C++

Will try to do the dull stuff quickly, then move into OOP and OO design

You need to practice to really learn C++

First two sessions is about the same for C, C++, Objective-C, Java, and C#

BABAR C++ Course 1 Paul F. Kunz

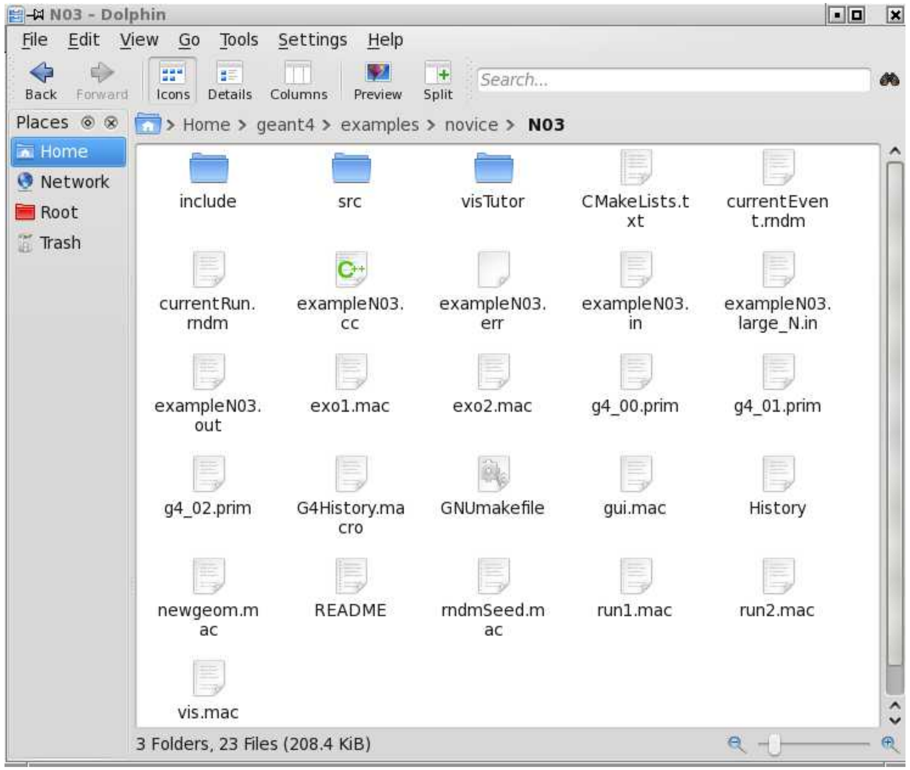
Generate a file .txt

```
Filename: SealedGEMEventAction.cc
.
.
#include "fstream.h"
.
.
void SealedGEMEventAction::EndOfEventAction(const G4Event*
evt)
{
.
.
  G4int evtNb = evt->GetEventID();
  n_hit = DrfHC->entries();
  if (n_hit > 0){
    for(G4int i=0;i<n_hit;i++){
       depEnerg1 += (*DrfHC)[i]->GetDepositEnergy();
    }
  }
  ofstream out1;
  out1.open("EnergyDepositeDriftHits.dat",fstream::app);
  out1 <<  evt << "          " << depEnerg1/keV << G4endl;
  out1.close();
}
```

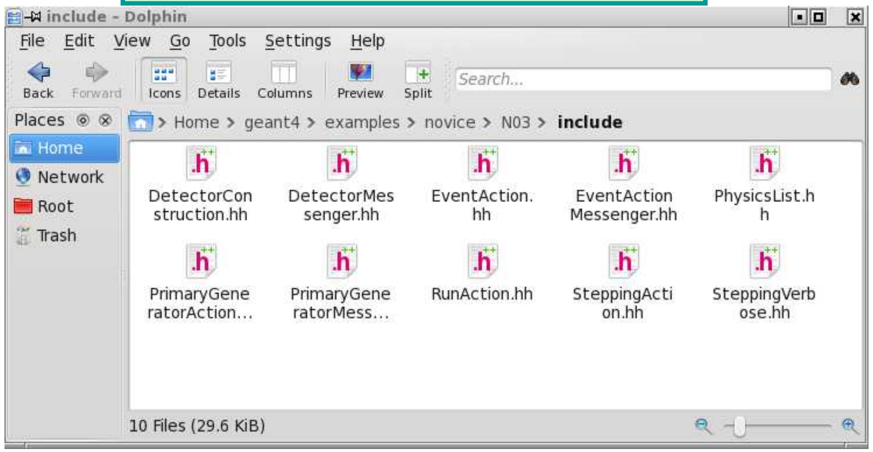
https://cds.cern.ch/record/44808

GEANT4 MODEL STRUCTURE

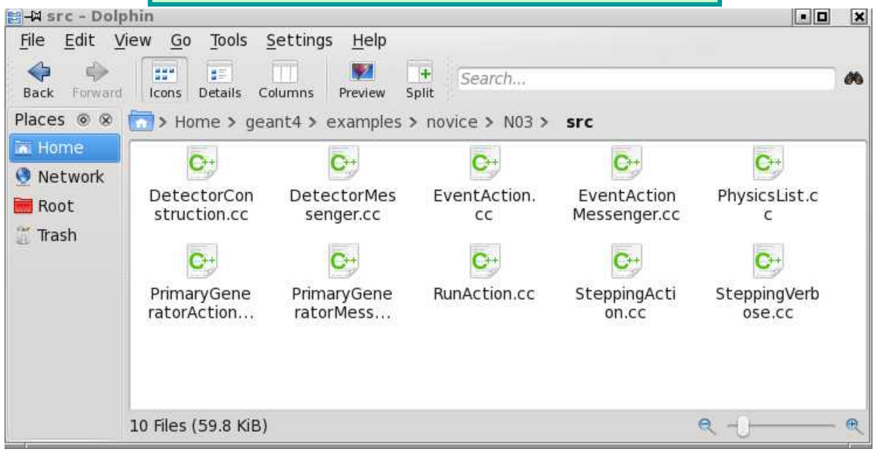
Simulation Model **Main Directory**



Simulation Model **include directory**



Simulation Model **source directory**



DEFINITION OF A SIMULATION MODEL

Simplest example of `main()`

```
#include "G4RunManager.hh"
#include "G4UImanager.hh"

#include "ExG4DetectorConstruction01.hh"
#include "ExG4PhysicsList00.hh"
#include "ExG4PrimaryGeneratorAction01.hh"

int main()
{
  // construct the default run manager
  G4RunManager* runManager = new G4RunManager;

  // set mandatory initialization classes
  runManager->SetUserInitialization(new ExG4DetectorConstruction01);
  runManager->SetUserInitialization(new ExG4PhysicsList00);

  // set mandatory user action class
  runManager->SetUserAction(new ExG4PrimaryGeneratorAction01);

  // initialize G4 kernel
  runManager->Initialize();

  // get the pointer to the UI manager and set verbosities
  G4UImanager* UI = G4UImanager::GetUIpointer();
  UI->ApplyCommand("/run/verbose 1");
  UI->ApplyCommand("/event/verbose 1");
  UI->ApplyCommand("/tracking/verbose 1");

  // start a run
  int numberOfEvent = 3;
  runManager->BeamOn(numberOfEvent);

  // job termination
  delete runManager;
  return 0;
}
```

G4RunManager

The first thing main() must do is create an instance of the G4RunManager class. This is the only manager class in the Geant4 kernel which should be explicitly constructed in the user's main(). It controls the flow of the program and manages the event loop(s) within a run. When G4RunManager is created, the other major manager classes are also created.

They run manager is also responsible for managing initialization procedures, including methods in the user initialization classes. Through these the run manager must be given all the information necessary to build and run the simulation, including:

• how the detector should be constructed

• all the particles and all the physics processes to be simulated

• how the primary particle(s) in an event should be produced and any additional requirements of the simulation.

MAIN COMPONENTS OF A MODEL

Relevant to the Detector Constructor

Create a Solid

```
G4double world_hx = 3.0*m;
G4double world_hy = 1.0*m;
G4double world_hz = 1.0*m;

G4Box* worldBox
   = new G4Box("World", world_hx, world_hy, world_hz);
```

Give Attributes

```
G4LogicalVolume* worldLog
   = new G4LogicalVolume(worldBox, Ar, "World");
```

Place in Space

```
G4double pos_x = -1.0*meter;
G4double pos_y =  0.0*meter;
G4double pos_z =  0.0*meter;

G4VPhysicalVolume* trackerPhys
  = new G4PVPlacement(0,                     // no rotation
              G4ThreeVector(pos_x, pos_y, pos_z),
                                             // translation position
              trackerLog,                    // its logical volume
              "Tracker",                     // its name
              worldLog,                      // its mother (logical) volume
              false,                         // no boolean operations
              0);                            // its copy number
```

Create a Tube

```
G4double innerRadius = 0.*cm;
G4double outerRadius = 60.*cm;
G4double hz = 25.*cm;
G4double startAngle = 0.*deg;
G4double spanningAngle = 360.*deg;

G4Tubs* trackerTube
  = new G4Tubs("Tracker",
               innerRadius,
               outerRadius,
               hz,
               startAngle,
               spanningAngle);
```

MAIN COMPONENTS OF A MODEL (Cont.)

Create a Material e.g. liquid Argon

```
G4double z, a, density;
density = 1.390*g/cm3;
a = 39.95*g/mole;

G4Material* lAr = new G4Material(name="liquidArgon", z=18., a, density);
```

Create a Molecule e.g. H_2O

```
G4double z, a, density;
G4String name, symbol;
G4int ncomponents, natoms;

a = 1.01*g/mole;
G4Element* elH  = new G4Element(name="Hydrogen",symbol="H" , z= 1., a);

a = 16.00*g/mole;
G4Element* elO  = new G4Element(name="Oxygen"  ,symbol="O" , z= 8., a);

density = 1.000*g/cm3;
G4Material* H2O = new G4Material(name="Water",density,ncomponents=2);
H2O->AddElement(elH, natoms=2);
H2O->AddElement(elO, natoms=1);
```

MAIN COMPONENTS OF A MODEL (Cont.)

Create a Material by Fraction of mass e.g. Air

```
G4double z, a, fractionmass, density;
G4String name, symbol;
G4int ncomponents;

a = 14.01*g/mole;
G4Element* elN  = new G4Element(name="Nitrogen",symbol="N" , z= 7., a);

a = 16.00*g/mole;
G4Element* elO  = new G4Element(name="Oxygen"  ,symbol="O" , z= 8., a);

density = 1.290*mg/cm3;
G4Material* Air = new G4Material(name="Air  ",density,ncomponents=2);
Air->AddElement(elN, fractionmass=70*perCent);
Air->AddElement(elO, fractionmass=30*perCent);
```

Create a Material from the G4 base materials – NIST database

```
G4NistManager* man = G4NistManager::Instance();

G4Material* H2O  = man->FindOrBuildMaterial("G4_WATER");
G4Material* Air  = man->FindOrBuildMaterial("G4_AIR");
```

MAIN COMPONENTS OF A MODEL (Cont.)

Create a New Material to the G4 base materials

```
G4double density;

density = 1.05*mg/cm3;
G4Material* water1 = new G4Material("Water_1.05",density,"G4_WATER");

density = 1.03*mg/cm3;
G4NistManager* man = G4NistManager::Instance();
G4Material* water2 = man->BuildMaterialWithNewDensity("Water_1.03","G4_WATER",density);
```

MAIN COMPONENTS OF A MODEL (Cont.)

Particle Definition

Geant4 provides various types of particles for use in simulations:

- ordinary particles, such as electrons, protons, and gammas
- resonant particles with very short lifetimes, such as vector mesons and delta baryons
- nuclei, such as deuteron, alpha, and heavy ions (including hyper-nuclei)
- quarks, di-quarks, and gluon

Relevant to the Physics List

The `G4ParticleDefinition` Class

`G4ParticleDefinition` has properties which characterize individual particles, such as, name, mass, charge, spin, and so on. Most of these properties are "read-only" and can not be changed directly. `G4ParticlePropertyTable` is used to retrieve (load) particle property of `G4ParticleDefinition` into (from) `G4ParticlePropertyData`.

Definition of an Electron

For example, the class `G4Electron` represents the electron and the member `G4Electron::theInstance` points its only object. The pointer to this object is available through the static methods `G4Electron::ElectronDefinition()`. `G4Electron::Definition()`.

MAIN COMPONENTS OF A MODEL (Cont.)

Relevant to the Physics List

Definition of an Ion

```
G4ParticleDefinition* GetIon(  G4int     atomicNumber,
                               G4int     atomicMass,
                               G4double    excitationEnergy);
```

Construct a Proton and a Geantino

Construct a proton and a geantino.

```
void MyPhysicsList::ConstructParticle()
{
  G4Proton::ProtonDefinition();
  G4Geantino::GeantinoDefinition();
}
```

Geant4 particle categories

G4BosonConstructor

G4LeptonConstructor

G4MesonConstructor

G4BarionConstructor

G4IonConstructor

G4ShortlivedConstructor.

MAIN COMPONENTS OF A MODEL (Cont.)

Register Processes for Gammas

Relevant to the Physics List

```
void MyPhysicsList::ConstructProcess()
{
  // Define transportation process
  AddTransportation();
  // electromagnetic processes
  ConstructEM();
}

void MyPhysicsList::ConstructEM()
{
  // Get pointer to G4PhysicsListHelper
  G4PhysicsListHelper* ph = G4PhysicsListHelper::GetPhysicsListHelper();

  //  Get pointer to gamma
  G4ParticleDefinition* particle = G4Gamma::GammaDefinition();

  // Construct and register processes for gamma
  ph->RegisterProcess(new G4PhotoElectricEffect(), particle);
  ph->RegisterProcess(new G4ComptonScattering(), particle);
  ph->RegisterProcess(new G4GammaConversion(), particle);
}
```

Physics Processes

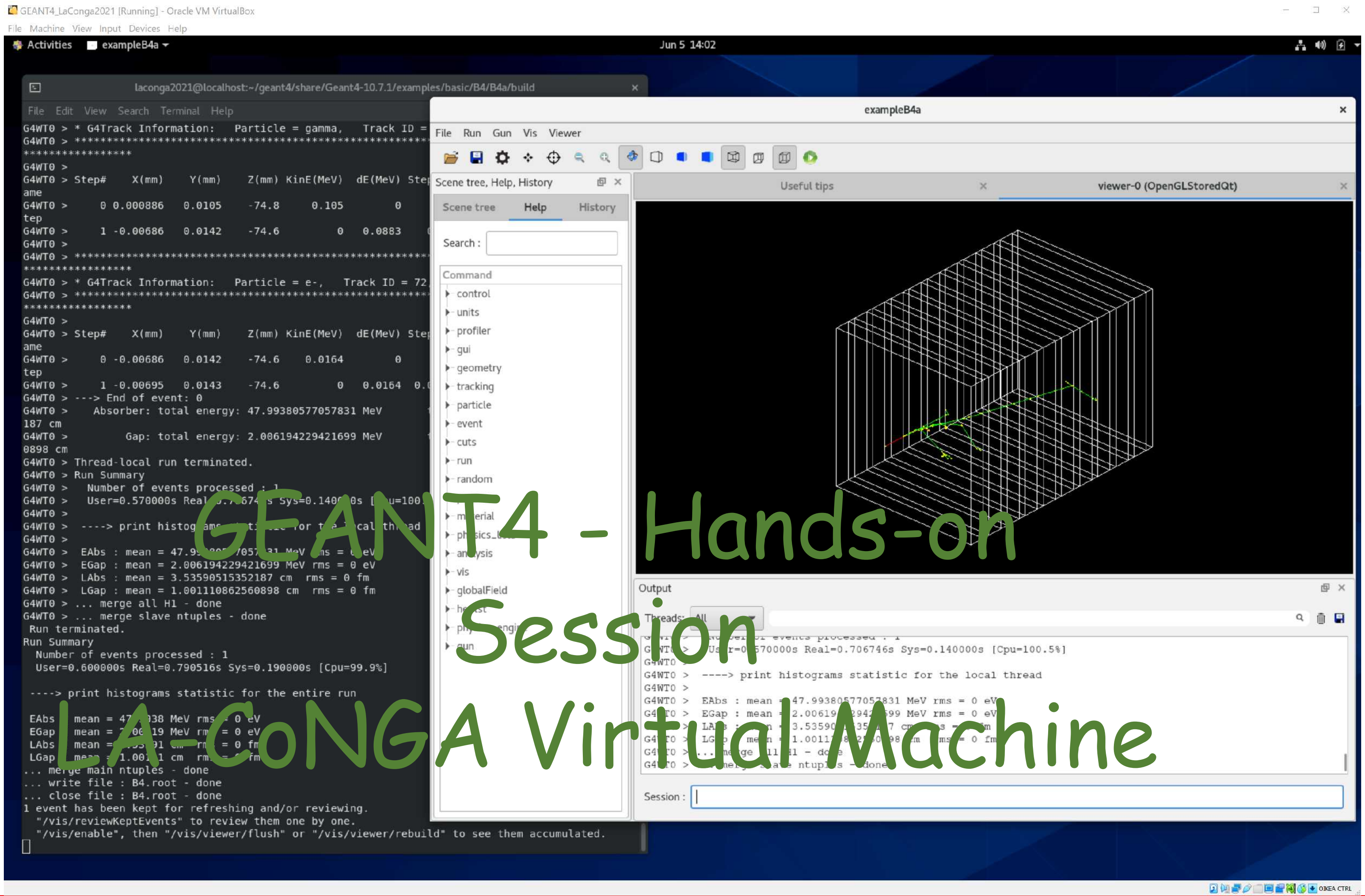
Physics processes describe how particles interact with materials. Geant4 provides seven major categories of processes:

- electromagnetic
- hadronic
- transportation
- decay
- optical
- photolepton_hadron
- parameterisation

MAIN COMPONENTS OF A MODEL (Cont.)

```
#include "ExG4PrimaryGeneratorAction01.hh"

#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

ExG4PrimaryGeneratorAction01::ExG4PrimaryGeneratorAction01(
                                const G4String& particleName,
                                G4double energy,
                                G4ThreeVector position,
                                G4ThreeVector momentumDirection)
  : G4VUserPrimaryGeneratorAction(),
    fParticleGun(0)
{
  G4int nofParticles = 1;
  fParticleGun  = new G4ParticleGun(nofParticles);

  // default particle kinematic
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle
    = particleTable->FindParticle(particleName);
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticleEnergy(energy);
  fParticleGun->SetParticlePosition(position);
  fParticleGun->SetParticleMomentumDirection(momentumDirection);
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

ExG4PrimaryGeneratorAction01::~ExG4PrimaryGeneratorAction01()
{
  delete fParticleGun;
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo......

void ExG4PrimaryGeneratorAction01::GeneratePrimaries(G4Event* anEvent)
{
  // this function is called at the begining of event

  fParticleGun->GeneratePrimaryVertex(anEvent);
}
```

Relevant to the Primary Generation Action



GEANT4 - Hands-on Session

LA-CoNGA Virtual Machine

http://laconga.redclara.net

contacto@laconga.redclara.net

lacongaphysics

Latin American alliance for
Capacity buildiNG in Advanced **physics**

LA-CoNGA **physics**

Cofinanciado por el
programa Erasmus+
de la Unión Europea

El apoyo de la Comisión Europea para la producción de esta publicación no constituye una aprobación del contenido, el cual refleja únicamente las opiniones de los autores, y la Comisión no se hace responsable del uso que pueda hacerse de la información contenida en la misma.