Universidade Federal do ABC

FAPESP

ufscar

Introduction to GEANT4

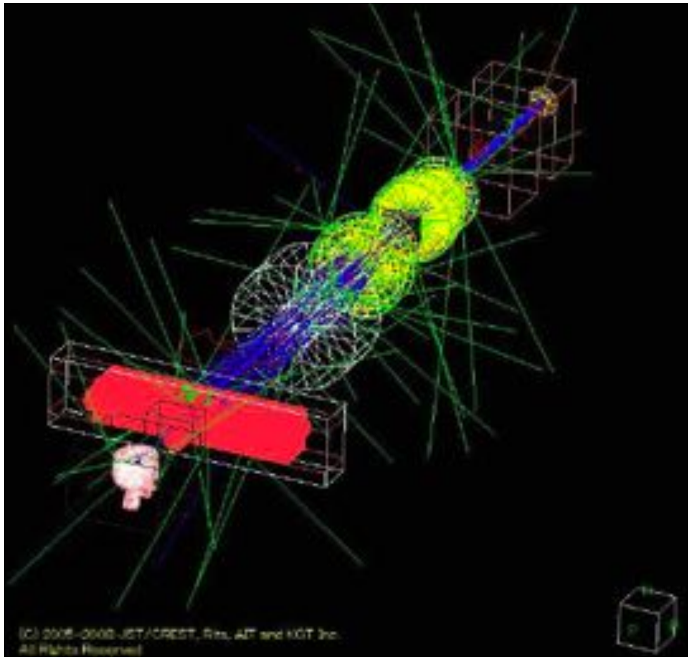
Laura Paulucci and Franciole Marinho

material adapted from M. Asai / P. Gumplinger / G. Santin / J. McCormick

Third UK-Latin America Workshop on Advanced Light Collection Systems

What is GEANT4?

- a general purpose Monte Carlo simulation tool for elementary particles passing through and interacting with matter
- wide variety of uses: high energy and nuclear physics, space engineering, medical applications, material science, radiation protection…
- Provides:
  - Geometry and navigation
  - Physics processes
  - Scoring
  - GUI and Visualization drivers
  - Extensive user guide documents and examples



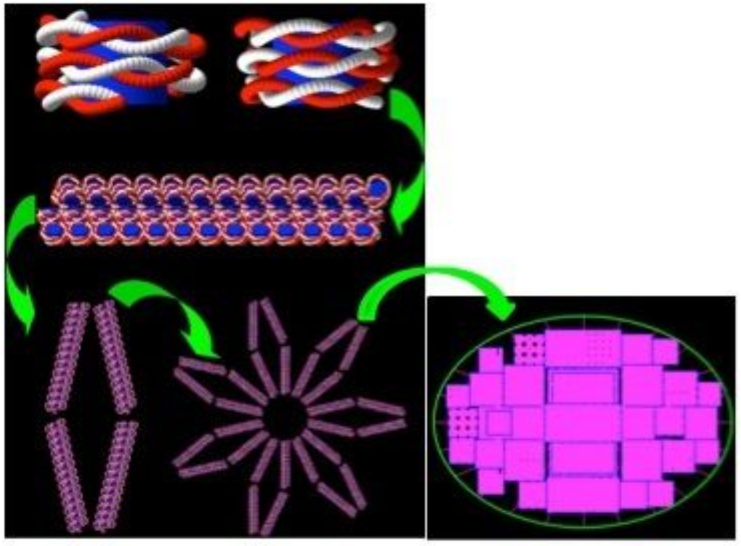
GEANT4 Basics

- To run, you have to build an application:
  - Define your geometrical setup (Material, volume…)
  - Define physics to get involved (Particles, physics processes/models, Production thresholds)
  - Define how an event starts (Primary track generation)
  - Extract information useful to you
- You may also want to
  - Visualize geometry, trajectories and physics output
  - Utilize (Graphical) User Interface
  - Define your own UI commands
  - ...

Geometry

- Rich collection of shapes
  - CSG (Constructed Solid Geometry), Boolean operation, Tessellated solid, etc.
  - The user can easily extend

- Describing a setup as hierarchy or ‘flat’ structure
  - Describing setups up to billions of volumes
  - Tools for creating & checking complex structures
  - Interface to CAD
- Geometry models can be ‘dynamic’
  - Changing the setup at run-time, e.g. “moving objects”



Physical processes

- Electromagnetic
- Hadronic and nuclear
- Photon/lepton---hadron
- Optical photon
- Decay
- Shower parameterization
- Event biasing techniques
- And you can plug-in more
- Sets of alternative physics models → user can freely choose appropriate models according to the type of his/her application (e.g. accuracy vs. speed)

Terminology

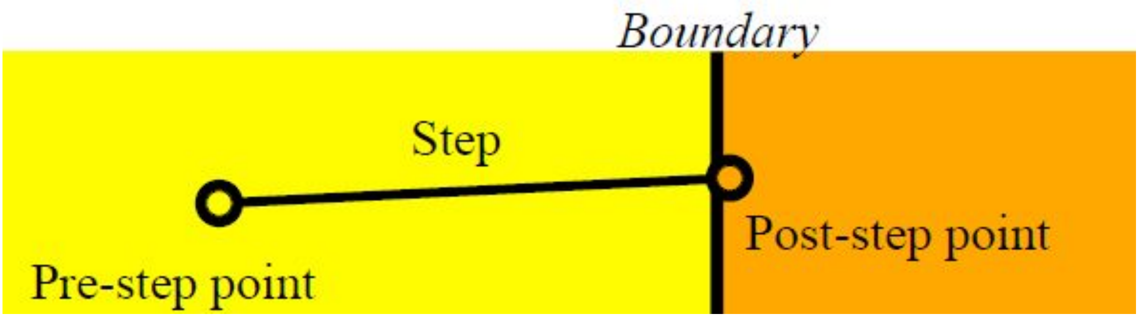
- Run, event, track, step, step point
- Track ↔ trajectory, step ↔ trajectory point
- Process
  - At rest, along step, post step
- Cut = production threshold
- Sensitive detector, score, hit, hits collection

What is an event?

- Basic unit of simulation in Geant4 → G4Event class
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as output)
- At beginning of processing, primary tracks are generated which are pushed into a stack
- Track is popped up from the stack one by one and “tracked”. Resulting secondary tracks are pushed into the stack
  - This “tracking” lasts as long as the stack has a track
  - When the stack is empty, processing of one event is over
- G4EventManager class manages processing an event
- G4UserEventAction is the optional user hook

What is a step?

- Has two points and information of a particle (energy loss on the step, time-of-flight spent by the step, etc.) → G4Step class
- Each point knows the volume (and material)
  - In case a step is limited by a volume boundary, the end point physically stands on the boundary → allows simulation of boundary processes (transition radiation or refraction...)
- G4SteppingManager class manages processing a step
- G4UserSteppingAction is the optional user hook



What is a track?

- Snapshot of a particle → G4Track class
  - It has physical quantities of current instance only. It does not record previous quantities
  - Step is a "delta" information to a track. Track is not a collection of steps, but is being updated by steps
- Track object is deleted when
  - it goes out of the world volume,
  - it disappears (by e.g. decay, inelastic scattering),
  - it goes down to zero kinetic energy and no "AtRest" additional process is required, or
  - the user decides to kill it artificially

What is a track?

- No track object persists at the end of event
  - For the record of tracks, use trajectory class objects
- G4TrackingManager manages processing a track
- G4UserTrackingAction is the optional user hook

What about trajectory and trajectory point?

- No track object persists at the end of event
- G4Trajectory is the class which copies some of G4Track information
- G4TrajectoryPoint is the class which copies some of G4Step information
- G4Trajectory has a vector of G4TrajectoryPoint
- At the end of event processing, G4Event has a collection of G4Trajectory objects
- Given G4Trajectory and G4TrajectoryPoint objects persist till the end of an event, you should be careful not to store too many trajectories (e.g. high energy EM shower tracks)

What is a run?

- Collection of events which share the same detector and physics conditions
  - Consists of one event loop
  - Starts with "Beam On"
- Within a run, the user cannot change
  - detector setup
  - settings of physics processes
- G4RunManager class manages processing a run, a run is represented by G4Run class or a user-defined class derived from G4Run
- G4UserRunAction is the optional user hook

Processes

- Each particle has its own list of applicable processes
- At each step, all processes listed are invoked to get proposed physical interaction lengths
- The process which requires the shortest interaction length limits the step
- Each process has one or combination of the following natures
  - AtRest • e.g. muon decay at rest
  - AlongStep (a.k.a. continuous process) • e.g. Cerenkov process
  - PostStep (a.k.a. discrete process) • e.g. decay on the fly

Cuts and storing info

- A Cut in Geant4 is a production threshold
- Not tracking cut, which does not exist in Geant4 as default
  - All tracks are traced down to zero kinetic energy
- Geant4 does proper physics simulation “silently” → You have to do something to extract useful information
  - Built-in scoring commands
    - Most commonly-used physics quantities are available

Storing info

- Use scorers in the tracking volume
    - Create scores for each event
    - Create own Run class to accumulate scores
  - Use user hooks :
    - G4UserEventAction, G4UserRunAction to get event /run summary
    - G4UserTrackingAction, G4UserSteppingAction,etc. → full access to almost all information, but do-it-yourself

Building an application

- main()
- Initialization classes
  - G4VUserDetectorConstruction
  - G4VUserPhysicsList
  - G4VUserActionInitialization
- Action classes: Invoked during an event loop
  - G4VUserPrimaryGeneratorAction
  - G4UserRunAction
  - G4UserEventAction
  - G4UserStackingAction
  - G4UserTrackingAction
  - G4UserSteppingAction

MANDATORY CLASSES!

How Geant4 runs

- Initialization
  - Construction of material and geometry
  - Construction of particles, physics processes and calculation of cross-section tables

- “Beam-On” = “Run”
  - Close geometry --> Optimize geometry
  - Event Loop

    ---> More than one runs with different geometrical configurations

Environment variables

- You need to set following environment variables to compile, link and run Geant4-based simulation.
  - Mandatory variables
    - G4SYSTEM – OS (e.g. Linux-g++)
    - G4INSTALL – base directory of Geant4
    - G4WORKDIR – your temporary work space
    - CLHEP_BASE_DIR – base directory of CLHEP
  - Variable for physics processes
    - G4LEVELGAMMADATA – directory of PhotonEvaporation data
  - Additional variables for GUI/Vis/Analysis

Main program

- Must
  - Construct G4RunManager (or derived class)
  - Set user mandatory classes to RunManager
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserPrimaryGeneratorAction

- Can
  - define VisManager, (G)UI session, optional user action classes, and/or your persistency manager

Select (G)UI

- In *main*(): construct a G4UIsession concrete class provided by Geant4 and invoke its *sessionStart*() method (according to your computer environments)
- Geant4 provides
  - G4UIterminal -- C-shell like character terminal
  - G4GAG -- Tcl/Tk or Java PVM based GUI
  - G4Wo -- Opacs
  - G4UIBatch -- Batch job with macro file

Visualization

- Derive your own concrete class from G4VVisManager according to your computer environments.
- Geant4 provides interfaces to graphics drivers
  - DAWN
  - WIRED
  - RayTracer -- Ray tracing by Geant4 tracking
  - OPACS
  - OpenGL
  - OpenInventor
  - VRML

Describing the detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class
- In the virtual method *Construct*()
  - Construct all necessary materials
  - Construct volumes of your detector geometry
  - Construct your sensitive detector classes and set them to the detector volumes
- Visualization attributes of your detector elements are optional

Selecting physics processes

- No default particles or processes
  - Even for the particle transportation, you have to define it explicitly
- Derive your own concrete class from G4VUserPhysicsList abstract base class
  - Define all necessary particles
  - Define all necessary processes and assign them to proper particles
  - Define cut-off ranges

Generating primary event

- Derive your concrete class from G4VUserPrimaryGeneratorAction abstract base class
- Pass a G4Event object to one or more primary generator concrete class objects which generate primary vertices and primary particles
- Geant4 provides three generators:
  - G4ParticleGun
  - G4HEPEvtInterface → Interface to /hepevt/ common block via ascii file
  - Interface to HepMC

Let's take a look at an example

- You may need to setup environment variables before hand
  - source /opt/geant4/share/Geant4-10.4.2/geant4make/geant4make.sh
- Get the "g4workshop_example/" directory in your account
- Create a working directory "g4workshop_build" and do
  - cmake -DGeant4_DIR=$G4COMP ../g4workshop_example
  - make -j
  - and cross fingers
- Before running identify key elements in the code
  - For instance, look in g4workshop_example/src/
  - DetectorConstruction.cc, PhysicsList.cc, PrimaryGeneratorAction.cc

Geometry description

```
#include "DetectorConstruction.hh"
#include "G4PhysicalConstants.hh"
#include "G4SystemOfUnits.hh"
#include "G4NistManager.hh"
```

```
void DetectorConstruction::DefineMaterials()
{
  G4NistManager * man = G4NistManager::Instance();
  G4Material* env_mat = man->FindOrBuildMaterial("G4_lAr");
  fDefaultMaterial = env_mat;
  G4cout << G4endl << *(G4Material::GetMaterialTable()) << G4endl;
}
```

Geometry description

```
G4VPhysicalVolume* DetectorConstruction::ConstructLine()
{
  // WORLD
  fWorldSizeXY  = 10*m;  fWorldSizeZ   = 10*m;

  fSolidWorld = new G4Box("World",fWorldSizeXY/2,fWorldSizeXY/2,
                               fWorldSizeZ/2);

  fLogicWorld = new G4LogicalVolume(fSolidWorld,fDefaultMaterial,
                                         "World");

  fPhysiWorld = new G4PVPlacement(0,G4ThreeVector(),"World",
                                       fLogicWorld,NULL,false,0);

      return fPhysiWorld;
}
```

Physics processes

```
void PhysicsList::ConstructEM()
{
  auto theParticleIterator=GetParticleIterator();
  theParticleIterator->reset();
  while( (*theParticleIterator)() ){
    G4ParticleDefinition* particle = theParticleIterator->value();
    G4ProcessManager* pmanager = particle->GetProcessManager();
    G4String particleName = particle->GetParticleName();

    if (particleName == "gamma") {
      // Construct processes for gamma
      pmanager->AddDiscreteProcess(new G4GammaConversion());
      pmanager->AddDiscreteProcess(new G4ComptonScattering());
      pmanager->AddDiscreteProcess(new G4PhotoElectricEffect());

    } else if (particleName == "e-") {
      // Construct processes for electron
      pmanager->AddProcess(new G4eMultipleScattering(),-1, 1, 1);
      pmanager->AddProcess(new G4eIonisation(),        -1, 2, 2);
      pmanager->AddProcess(new G4eBremsstrahlung(),    -1, 3, 3);
    }...
```

Primary generator

```
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  G4int i=0;    G4double x0,y0,z0,theta,phi;    G4double test;
  x0=y0=z0=0.0;

  phi = CLHEP::twopi*CLHEP::RandFlat::shoot(0.0,1.0)*rad;
  while(i==0){
     theta = CLHEP::RandFlat::shoot(0.0,CLHEP::pi)*rad;
     test = CLHEP::RandFlat::shoot(0.0,1.0);
     if(sin(theta)>test)i++;
  }

  //Particle direction
  G4double kx, ky, kz;
  kx=cos(phi)*sin(theta);
  ky=sin(phi)*sin(theta);
  kz=cos(theta);

  G4ThreeVector dir_vec (kx,ky,kz);
  ...
```

Primary generator

```
...
  //Polarization
  G4ThreeVector polar = Polarisation(dir_vec);

  fParticleGun->SetParticleEnergy(4*GeV);
  fParticleGun->SetParticleMomentumDirection(dir_vec);
  fParticleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));
  //fParticleGun->SetParticlePolarization(polar);

  G4ParticleDefinition* particle=
    G4ParticleTable::GetParticleTable()->FindParticle("mu-");

//G4ParticleTable::GetParticleTable()->FindParticle("opticalphot
on");

  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->GeneratePrimaryVertex(anEvent);

}
```

Running the application

- Now open g4workshop.cc

```
int main(int argc,char** argv) {
 // Choose the Random engine and random seed with system time
 // Construct the default run manager

 // Set mandatory user initialization classes
 DetectorConstruction* detector = new DetectorConstruction;
 runManager->SetUserInitialization(detector);

 PhysicsList* physics = new PhysicsList();
 runManager->SetUserInitialization(physics);
 // User action initialization

 runManager->SetUserInitialization(new ActionInitialization(detector));

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

 // Set visual and interface managers...
}
```

Running the application

- Go back to terminal on "g4workshop_build/" and do "./g4workshop"
- You should get on terminal description of:
  - Materials composition and characteristics
  - Physics processes per particle type
    - Optical, electromagnetic and hadronic
  - and list of interactions (ocurred only)

4 GeV μ