Geant4: a user perspective

SLAC - August 2000
INFN Glast
Trieste-Udine Group
Contents

- A brief overview of Geant4
- Basic structures
- Some details
- A user (limited) experience
- Planned activities for Udine-Trieste
Geant4 Overview

- Lots of components
  - Geometry & Materials
  - Physics
  - Graphics drivers
  - others ..

- True OO design
  - A simplified initial approach..
  - .. can be later refined
  - Modular and quite extendable

Glast Udine-Trieste
Geant4 Overview (cont’d)

- Lots of particles and processes
- Highly customizable Physics
  - You can decide what physics
  - You can insert new physics
- User Support
  - Supported by CERN, IT division
  - Lots of (potential) users
  - Lots of (enthusiastic) developers
  - 2 pub. releases in a year, 1 dev. release each month
Geant4 Overview (cont’d)

- Various applications
  - HEP
  - Medical
  - Space

- Space applications are a G4 collaboration goals
  - ESA
  - We are now G4 collaboration member (Udine-Trieste)
    » Agile investigations
    » Glast investigations

Glast Udine-Trieste
Geant4 Class Categories

- Run and Event
- Tracking and Track
- Geometry
- Particle and Matter
- Physics
- Hits and Digitazion
- Visualization
- Interfaces
A run is a collection of events that shares the same physics and the same detector geometry

- An event start with the generation of an ensemble of particles (primaries)
  - each particle is followed along its track
    - each track is divided in steps
    - for each step the physical processes that depends on both particles and materials are simulated
  - new particle with their track can be generated
  - particles can deposit energy in materials; if they belong to a sensitive detector an hit is recorded
The basic structures of a GEANT4 simulation (cont’d)

- The simulation is managed by the only mandatory explicit class in the main(), i.e. the G4RunManager
- The user must give two type of classes and register them to the manager (see later)
  - Initialization Classes
  - Action Classes
There are three mandatory classes to be registered to the RunManager, derived from the following ones (abstract classes):

- Initialization
  - G4VUserDetectorConstruction (Geometry)
  - G4VUserPhysicsList (Physics and Particles)
- Action
  - G4VUserPrimaryGeneratorAction (Primary vertex)
The basic structures of a GEANT4 simulation (cont’d)

- Other possible Action Classes
  - G4UserRunAction
  - G4UserEventAction
  - G4UserTrackingAction
  - G4UserSteppingAction

Glast Udine-Trieste
The basic structures of a GEANT4 simulation (cont’d)

- User interface is managed by G4UIManager
  - It can be customized by the user

- The User Interface can be
  - hard coded
  - interactive
  - with macro

- The graphics is managed by a G4VisManager derived class
The basic structures of a GEANT4 simulation (cont’d)

An example (just some pieces of the main())

........
// Construct the default run manager
G4RunManager* runManager = new G4RunManager;
// set mandatory initialization classes
runManager->SetUserInitialization(new BTEMDetectorConstruction);
runManager->SetUserInitialization(new BTEMPhysicsList);
.....
// visualization manager
G4VisManager* visManager = new BTEMVisManager;
visManager->Initialize();
.....

Glast Udine-Trieste
The basic structures of a GEANT4 simulation (cont’d)

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

......

// set optional user action classes
runManager->SetUserAction(new BTEmRunAction);
runManager->SetUserAction(new BTEMEventAction);

......

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

......

Glast Udine-Trieste
The basic structures of a GEANT4 simulation (cont’d)

To summarize

- **GeneratorAction**
  - generation of primary vertices (In GeneratePrimaries method)

- **DetectorConstruction**
  - definitions of materials (In Construct method)
  - definition of the geometry (In Construct method)

- **PhysicsList**
  - definition of particles (In ConstructParticle method)
  - registration of physical processes (In ConstructProcess method)
A stripped version (you can see also some glimpses of the UIManager; the same things can be done throught member function of particleGun)

.....
G4int n_particle = 1;
particleGun = new G4ParticleGun(n_particle);
G4UImanager* UI = G4UImanager::GetUIpointer();
UI->ApplyCommand("/gun/particle e-");
UI->ApplyCommand("/gun/energy 1.0 GeV");
UI->ApplyCommand("/gun/position -0.01 0.05 0.19 m");
UI->ApplyCommand("/gun/direction 1.0 0.0 0.0");

.....
G4int i = anEvent->GetEventID();
particleGun->GeneratePrimaryVertex(anEvent);
Some details: particlegun

- Particlegun is very easy but quite generic
- For more sophisticated primary vertices generation the user has to supply its own models
- There is an interface via ASCII files to some existing HEP generator (like Pythia)
There are lots of possible methods to define materials; usually they are defined in the Construct method of DetectorConstruction class

**Elements**

\[ a = 1.01\text{g/mole}; \]

G4Element* H = new G4Element(name="Hydrogen",symbol="H", z = 1., a);

**Isotopes**

G4Isotope* U5 = new G4Isotope(name="U235", iz=92, n=235, a=235.01*g/mole);
G4Isotope* U8 = new G4Isotope(name="U238", iz=92, n=238, a=238.03*g/mole);
G4Element* U = new G4Element(name="enriched Uranium", symbol="U", ncomponents=2);
U->AddIsotope(U5, abundance= 90.*perCent);
U->AddIsotope(U8, abundance= 10.*perCent);
Some details: Materials (cont’d)

- New materials
  - density = 1.390*g/cm³;
  - a = 39.95*g/mole;
  - G4Material* lAr = new G4Material(name="liquidArgon", z=18., a, density);

- Material from elements (chem. molecule)
  - density = 1.000*g/cm³;
  - G4Material* H2O = new G4Material(name="Water", density, ncomponents=2);
  - H2O->AddElement(H, natoms=2);
  - H2O->AddElement(O, natoms=1);

- Materials from elements (fract. masses)
  - density = 1.290*mg/cm³;
  - G4Material* Air = new G4Material(name="Air", density, ncomponents=2);
  - Air->AddElement(N, fractionmass=0.7);
  - Air->AddElement(O, fractionmass=0.3);
Some details: Materials (cont’d)

- Mixture of mixtures
  
density = 0.200*g/cm3;
G4Material* Aerog = new G4Material(name="Aerogel", density, ncomponents=3);
  Aerog->AddMaterial(SiO2, fractionmass=62.5*perCent);
  Aerog->AddMaterial(H2O , fractionmass=37.4*perCent);
  Aerog->AddElement (C   , fractionmass= 0.1*perCent);
Some details: Geometry

- Defined in the DetectorConstruction class; it is based on three elements:
  - Solids
  - Logical Volumes
  - Physical Volumes

- A useful feature: Readout Geometry
  - virtual parallel geometry
  - give access to geometrical information of hits (volumes hierarchies)
  - useful in simulate channels responses

Glast Udine-Trieste
Some details: **Geometry** (cont’d)

- **Solids**
  - Primitive ones (box, tube etc.)
  - BREP ones (also via STEP interchange file)
  - Booleans operations

- **Logical volumes**
  - **Solid** + **material** without positional information
  - Hierarchy of volumes, visualization attributes and sensitive detectors relies on logical volumes

*Glast Udine-Trieste*
Some details: **Geometry (cont’d)**

- Physical volumes
  - positional information with respect to a mother logical volume (rotation, translation)
  - three possibilities for placement:
    » single positioned copy
    » replicas
      • multiple copies of a volume along an axis
    » parameterized repeated volumes
      • user defined parameterization
      • cannot allow for daughters volumes; useful only for leaf nodes in the hierarchy
Some details: Geometry (cont’d)

G4Box* SiliconLadder_box
    = new G4Box("SiliconLadder_box",silicon_x/2,silicon_y/2,silicon_z/2);

G4LogicalVolume* SiliconLadder_log
    = new G4LogicalVolume(SiliconLadder_log,Si,"SiliconLadder_log",0,0,0);

G4VPhysicalVolume* SiliconLadder_phys
    = new G4PVPlacement(0,
        G4ThreeVector(siliconPos_x, siliconPos_y, siliconPos_z),
        SiliconLadder_log,"SiliconLadder_phys",experimentalHall_log,false,0);
Some details: Physics

- **Particles**
  - Normally defined in the PhysicsList class
  - Lots of particle (stable, long life, short life, optical photons, nuclei and more)
  - Various implementation
    » Singleton: unique static object
    » On-the-fly creation (ions)
    » Dynamic creation by processes
Some details: Physics (cont’d)

Processes

- Processes directly linked to particles
- Activable and disactivable
- Lots of physics
  » Electromagnetic Standard
  » Low Energy Electromagnetic (100 Gev)
  » Hadronic
    • Lots of models
Some stripped code from PhysicsList

......
G4Electron::ElectronDefinition();
G4Positron::PositronDefinition();
G4Gamma::GammaDefinition();
......
while( (*theParticleIterator)() ){
  G4ParticleDefinition* particle = theParticleIterator->value();
  G4ProcessManager* pmanager = particle->GetProcessManager();
  G4String particleName = particle->GetParticleName();
  if (particleName == "gamma") {
    //gamma
    pmanager->AddDiscreteProcess(new G4PhotoElectricEffect());
    pmanager->AddDiscreteProcess(new G4ComptonScattering());
  }
  ......
Some details: Hits and Digi

- All hits in Geant4 come from G4VHit
- It’s up to the user to define the content of an Hit
- A collection of hits is associated to every event in the simulation
- Digi are almost the same (G4VDigi)
Some details: Sensitive Detector

- Class derived from G4VSensitiveDetector
- All sensitive detectors (logical volumes) are registered with the singleton G4SDManager (accessed by G4SDManager::GetSDMpointer)
- Three methods:
  - ProcessHits()
  - Initialize()
  - EndOfEvent()
Some details: EventAction

- Gives access to events with two methods
  - BeginOfEventAction(const G4Event*)
  - EndOfEventAction(const G4Event*)

- Useful to work with hits collection associated to an event, for example to dump to a file or an histogram

- Similar for SteppingAction, TrackAction or RunAction

Glast Udine-Trieste
A user experience

A day work starting from an example in the standard distribution. Not optimized

*Glast Udine-Trieste*
A user experience (cont’d)

4x4 Geometry; another day from the single tower (again it’s not optimized).
A user experience (cont’d)

Energy deposition of positron in 400 micron of Silicon; a day work starting from scratch (with some cut-and-paste from examples).

Now we have energy deposition and energy loss for various particles and materials.

It is a (bad) example on how to extract hit information in Geant4.
Planned activities

- Continue to study G4 (Hits and Digits, fastMC)
- XML interface
  - Geometry design and specification
  - Incoming flux
- Developing of a Glast simulation
  - Beam Test comparison (hit distributions along a tower?)
- General space application example + LowEnergy EM unit-tests for next release of GEANT4