INTRODUCTION TO GEANT4

HAYK HAKOBYAN

UTFSM – 2009

* Object-Oriented Programming basics

* Main requirements for simulation packages

* Introduction to the basic principles of Geant4

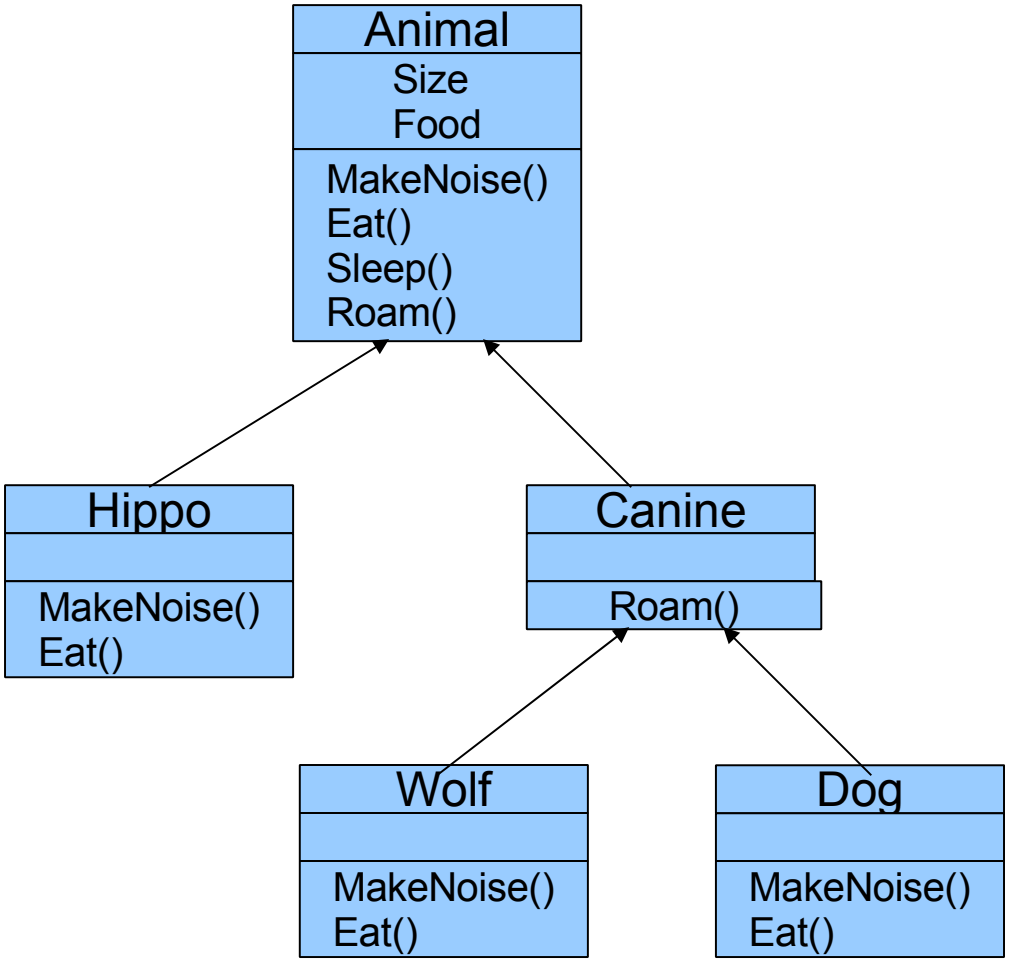
* Geant4 geometry implementation

* Overview on detector response in Geant4

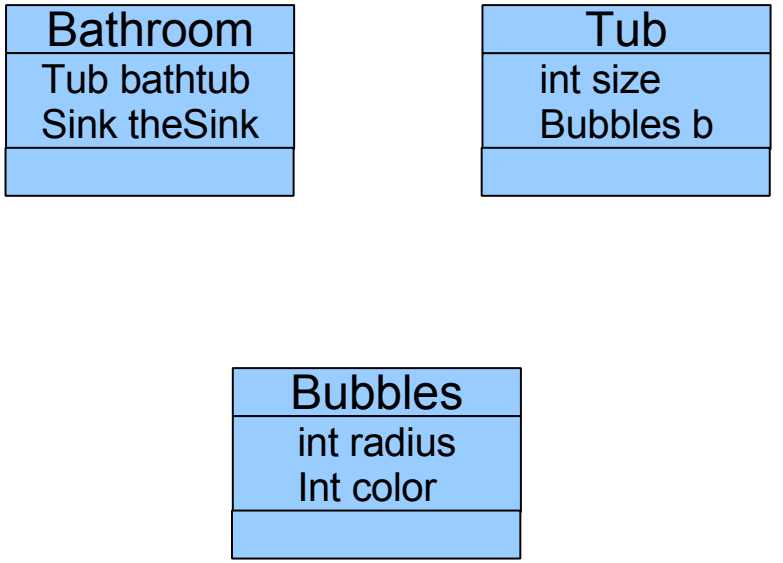
Object-Oriented Programming basics

Object-Oriented Programming (OOP) is a programming paradigm that uses “objects” - data structures consisting of datafields and methods and their interactions to design applications and computer programs. Programming techniques may include features such as information hiding, data abstraction, encapsulation, modularity, polymorphism and inheritance. Classes provide the blueprints to create objects of that class.

IS_A



HAS_A



OOP basic concepts

Objects, Classes – a class defines the abstract characteristics of a thing (object), including the objects attributes (instance variables) and the objects behavior (methods).

Inheritance – generic features common to the group are used to define the foundation (or base class) and then new, more specialized classes (subclasses or derived classes) are formed to inherit attributes and behavior from their parent classes (and can introduce their own).

Encapsulation – hiding of the internal complexity of the object performance and presentation of the certain interface for the external user.

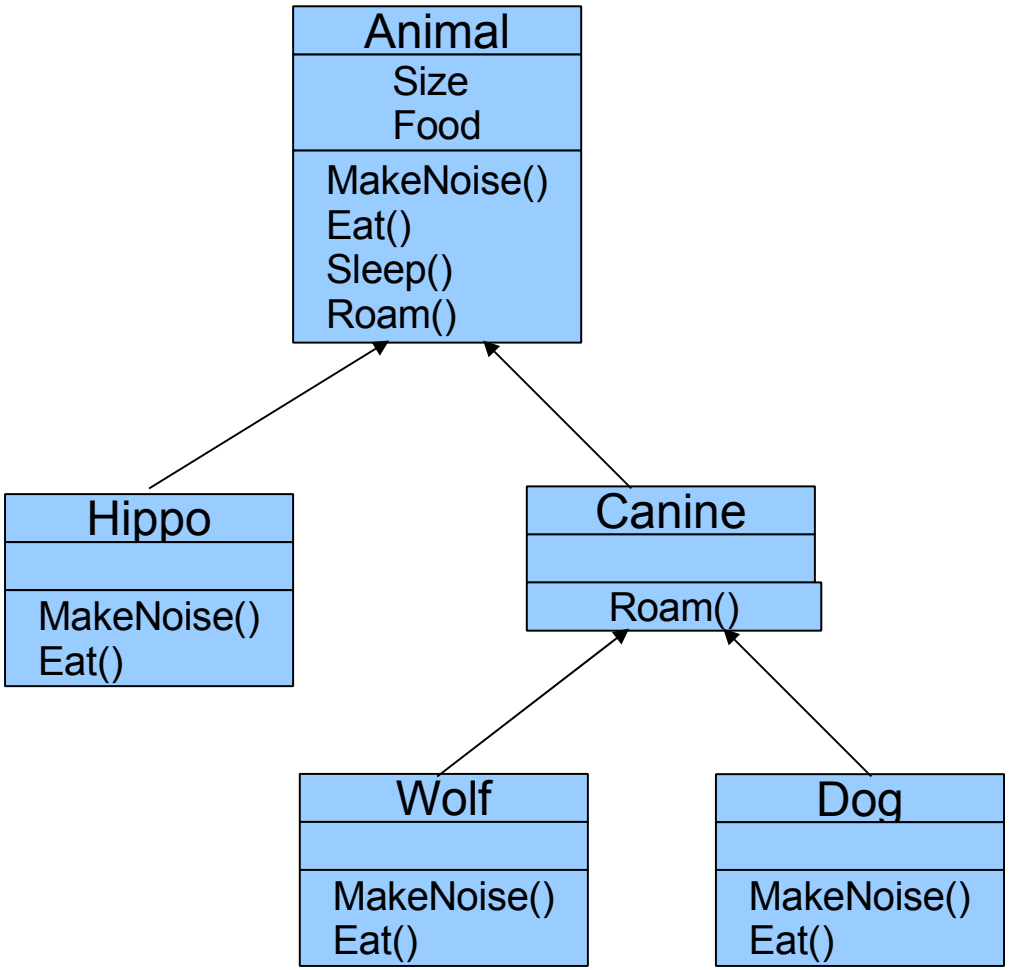
Abstraction – the presentation of the simple concept (or object) to the external world as an interface ensuring that changes on the either sides of this interface don't propagate across it unless unavoidable.

Polymorphism – ability to have one method call work on several different classes of objects, even if these classes need different implementation of the method call.

Object-Oriented – each object knows its own class and which methods manipulate objects in that class.

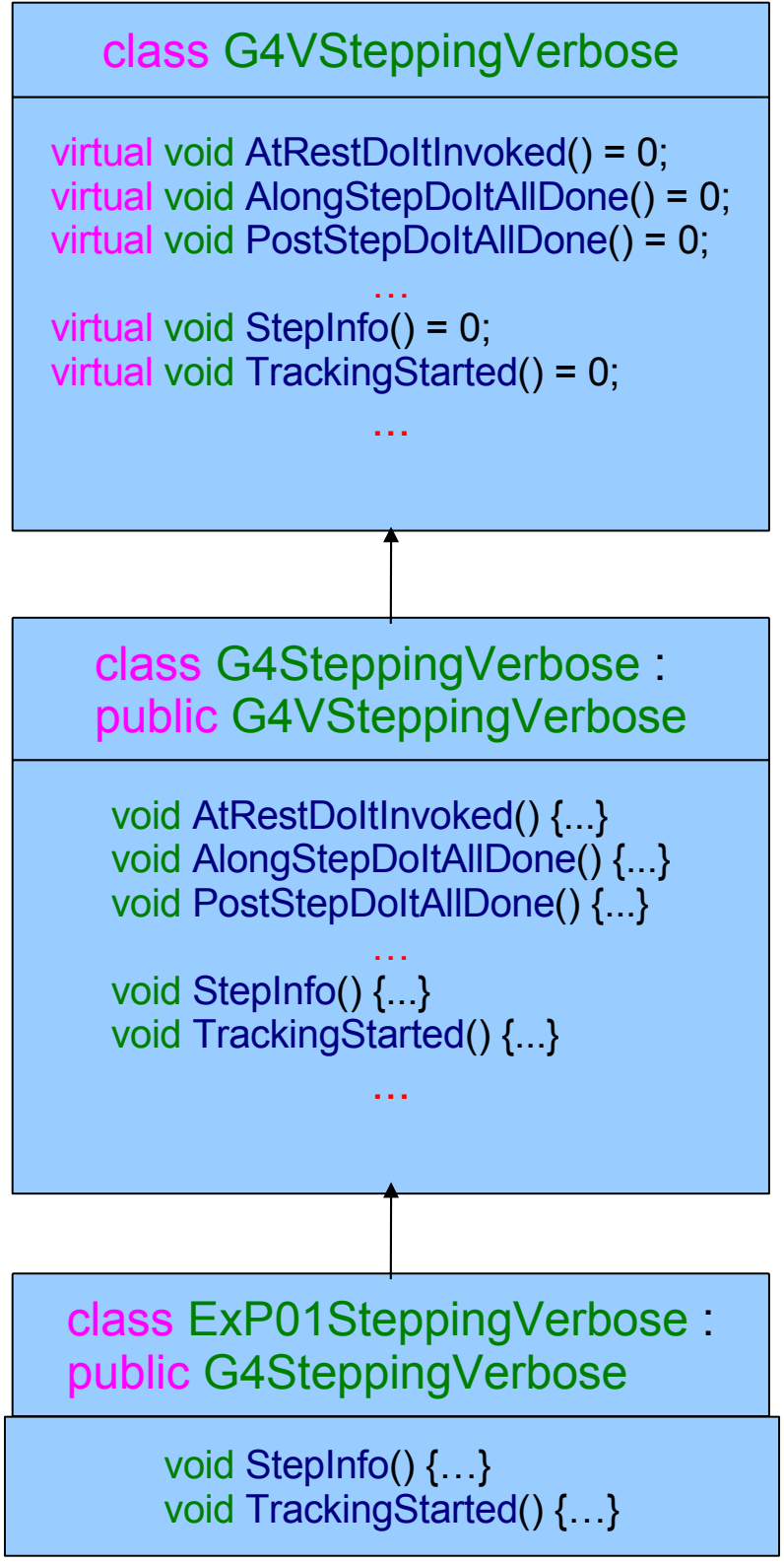
Simple Demonstration of Polymorphism

IS_A




```
int main()
{
// ...
        Animal *animals[3];
        animals[0] = new Dog();
        animals[1] = new Wolf();
        animals[2] = new Hippo();

        for (int I = 0; I < 3; i++) {
                animals[i] → eat();
                animals[i] → roam();
                animals[i] → sleep();
        }
// ...
}
```




```
int main(int argc, char** argv) {
                    …
G4VSteppingVerbose::SetInstance(new
ExP01SteppingVerbose);
                    …
}
```

```
G4SteppingManager :: G4SteppingManager() {
…
#ifdef G4VERBOSE
if(G4VSteppingVerbose::GetInstance()==0) {
     fVerbose = new G4SteppingVerbose();
    G4VSteppingVerbose::SetInstance(fVerbose);
     fVerbose → SetManager(this);
     KillVerbose = true;
}
else {
     fVerbose = G4VSteppingVerbose::GetInstance();
     fVerbose → SetManager(this);
     KillVerbose = false;
}
#endif
…
}
```

Main requirements for simulation packages

Simulation plays a fundamental role in various domains and phases of experimental physics project:

* design of the experimental set-up
* evaluation and definition of the potential physics output of the project
* evaluation of the potential risks of the project
* performance of the experiment
* development, test and optimization of reconstruction and physics analysis software
* contribution to the calculation and validation of physics results

Simulation Components

* simulation of the passage of the particle through the matter
* physics events generators
* electronics response generators
  .etc

(usually the simulation of the complex experiment consists of several of these components interfaced to one another)

After specifying by the user the geometry of the particle detector the software system automatically transports the shot particle into the detector by simulating the particle interactions in matter based on the Monte-Carlo method.

Basic requirements for a simulation system

* Modeling the experimental set-up
* Tracking particles through matter
* Interaction of particles with matter
* Modeling the detector response
* Run and Event Control
* Interface to event generators
* Visualization of the set-up, tracks and hits
* User interface
* Persistency

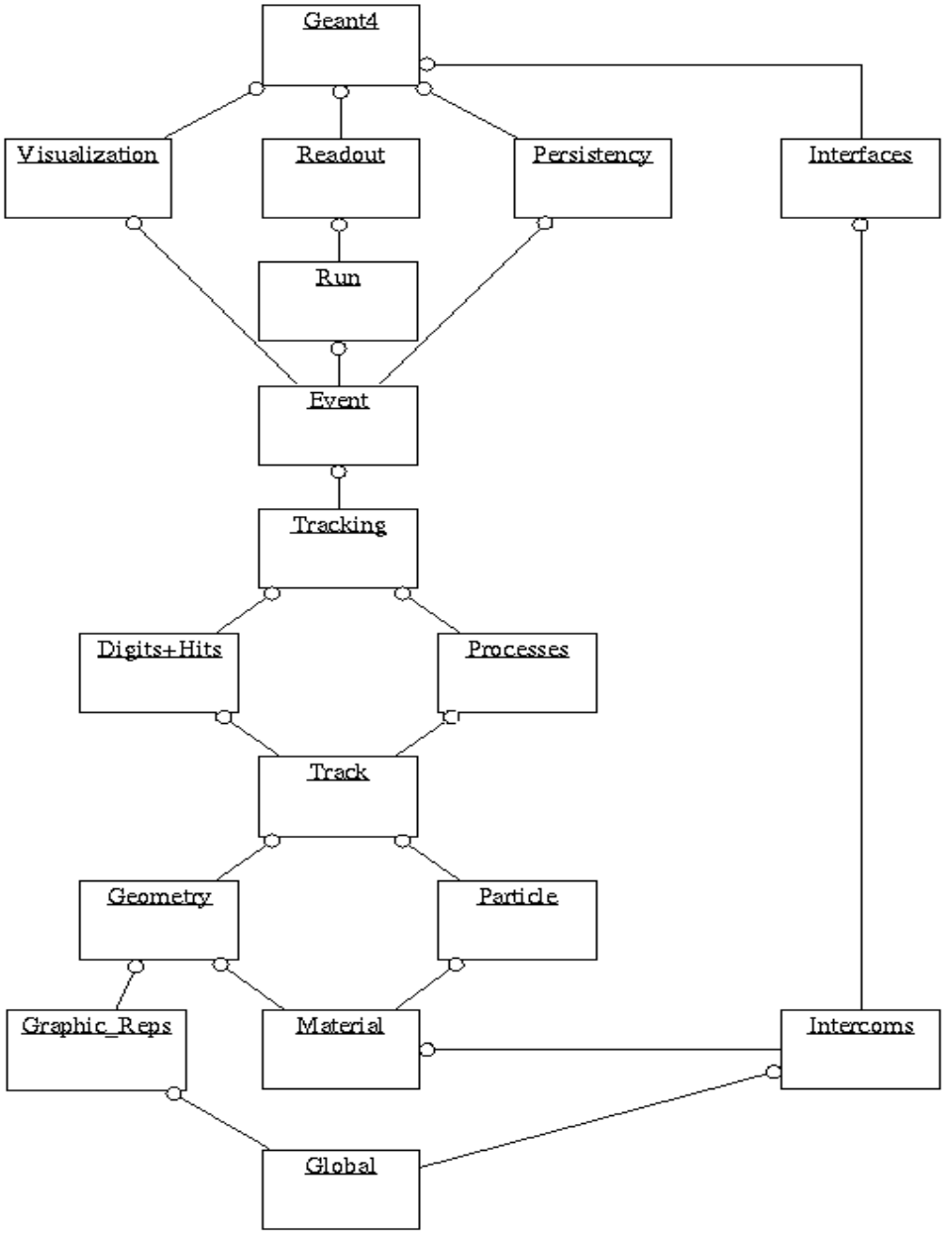


Diagram of Geant4 class categories and their relations. The circle at the end of the straight line means that the class category with the circle uses the category at the other end of the line.

Introduction to the basic principles of Geant4

GEANT4 links

* Collaboration homepage/downloads:
  http://geant4.web.cern.ch

* Full set of documentation is available at
  http://geant4.web.cern.ch/geant4/support/userdocuments.shtml

* Examples for the novices are available at Geant4 working directory:
  novice/extended/advanced categories

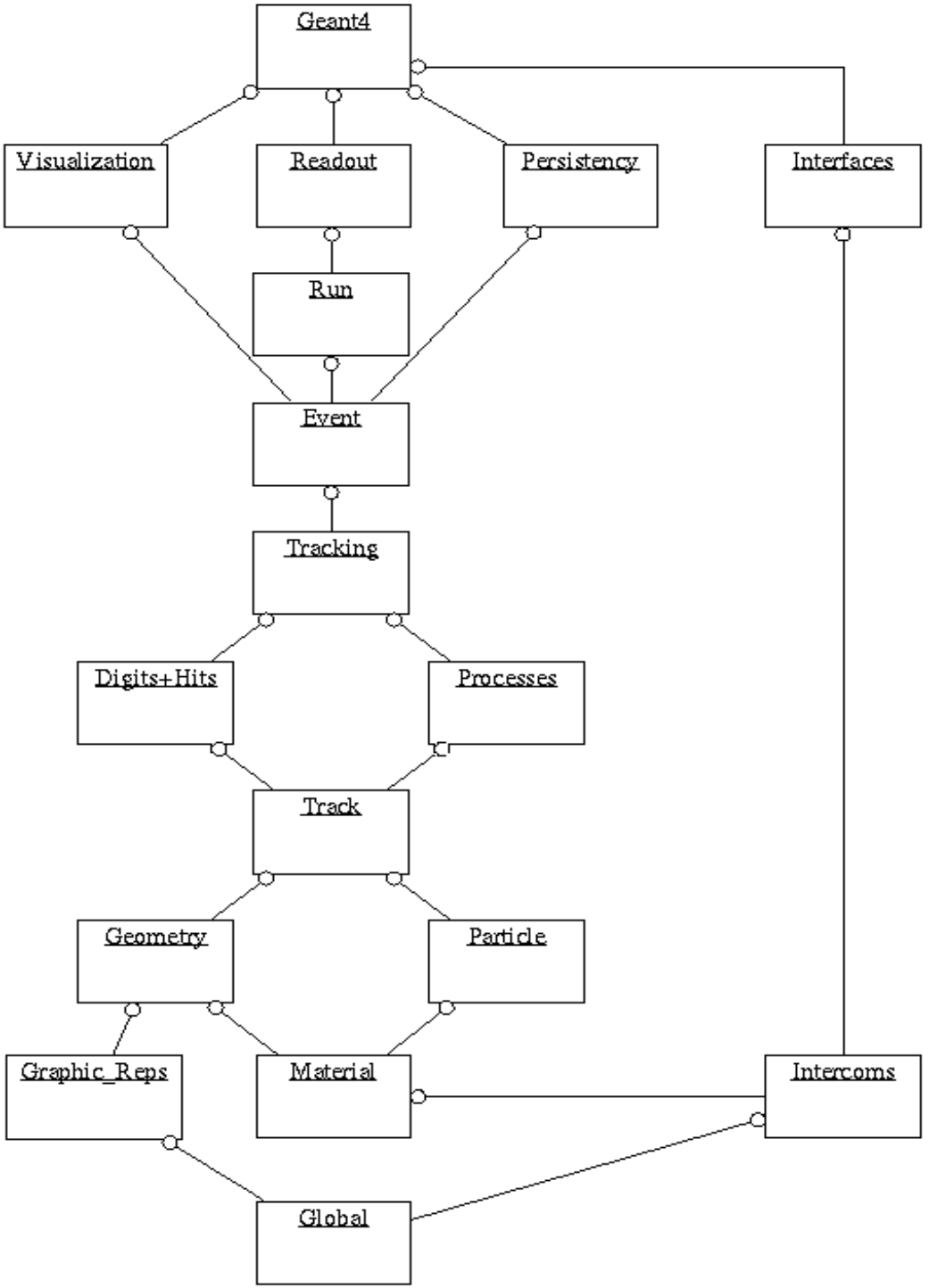
CLHEP – A Class Library for High Energy Physics

* CLHEP is an external library required by Geant4. It is intended to be High Energy Physics (HEP) specific foundation and utility classes such as random generators, 3- and 4-dimensional physics vectors, geometry and linear algebra. CLHEP is structured in a set of packages independent of any external package.

* Web link for CLHEP:
  http://proj-clhep.web.cern.ch

Geant4 comes with Seven visualization drivers:

OpenGL, OpenInventor, HepRep, DAWN, VRML, RayTrace, ASCIITree



geant4/source

global
graphics_reps
materials
geometry
particles
track
digits_hits
processes
tracking
event
run
readout
visualization
persistency
intercoms
Interfaces
parameterisations
physics_lists
error_propagation
g3tog4

GEANT4 Actions

Transports particles step-by-step by taking into account the interactions with materials and external electromagnetic fields until the particle

* loses its kinetic energy to zero,
* disappears by an interaction,
* comes to the end of the simulation volume.

Provides a way for the user to access the transportation process and grab the simulation results

* at the beginning and end of transportation,
* at the end of the each stepping in transportation,
* at the time when the particle is going into the sensitive volume of the detector

Minimal Setup for a GEANT4 Simulation

Geant4 users must define:

* **physics** processes (instantiate particles and processes attached to particles)
* detector **geometry** (material definition)
* primary generator action (**particle gun**)

Auxiliary it might be provided information about:

* decide on actions on different Monte-Carlo levels (**data collection**/**analysis** for particle *step*, *event* and *run*)
* magnetic and electric field
* necessary actions during access to the particle transportation
* necessary actions during the passage of the particle through the sensitive volume of the detector
* etc.

exampleN01.cc (concise version)

```
#include “G4RunManager.hh”
#include “G4UImanager.hh”

#include “ExN01DetectorConstruction.hh”
#include “ExN01PhysicsList.hh”
#include “ExN01PrimaryGeneratorAction.hh”

int main() {

G4RunManager* runManager = new G4RunManager;

G4VUserDetectorConstruction* detector = new ExN01DetectorConstruction;
runManager->SetUserInitialization(detector);

G4UserPhysicsList* physics = new ExN01PhysiucsList;
runManager->SetUserInitialization(physics);

G4VUserPrimaryGeneratorAction* gen_action = new ExN01PrimaryGeneratorAction;
runManager->SetUserAction(gen_action);

runManager->Initialize();

G4int numberOfEvent = 3;
runManager->BeamOn(numberOfEvent);

delete runManager;
return 0;
}
```

RUN

* a run in Geant4 starts with “BeamOn”
* within the run the user cannot change detector geometry and setting of physics processes
* a run is a collection of events which share the same detector conditions

EVENT

* at the begging of the processing an event contains the primary particles. These primaries are pushed into stack
* when the stack is getting empty the processing of event is over
* G4Event class represents an event. It has the following objects at the end of its processing: list of primary vertexes and particles; trajectory container (optional); hit collections; digits collections(optional)

TRACK

* track is a snapshot of a particle
* step is a “delta” information of the track (track is not a collection of steps)
* track is deleted when it goes out of the world volume; it disappears (e.g. decay); it goes down to zero kinetic energy and no “at rest” additional process is required; the user decides to kill it
* a track is made of three layers of class objects: G4Track, G4DynamicPartcile, G4PartcileDefinition

STEP

* step has two points and also “delta” information of the particle (energy loss on the step, time-of-flight spent by step etc.)
* each point knows the volume. In case a step is limited by the volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume.

TRAJECTORY

* Trajectory is a record of the track history. It store some information of all steps done by track as objects of G4TrajectoryPoint class.
* it is advised not to store trajectories for secondary particles generated in a shower because of the memory consumption.
* the user can create his own trajectory class deriving from G4VTrajectory and G4VTrajectoryPoint base classes for storing any additional information useful to the simulation.

List of mandatory and optional classes to be defined by user (not full list)

* G4VUserDetectorConstruction creates and positions different parts of the detector or the experimental setup.

* G4VUserPhysicsList provides particle definitions, physics processes and range cuts to be used throughout the simulation.

* G4VUserPrimaryGeneratorAction generates the primary particles and defines their momentum, direction and energy for all events.

* G4UserRunAction defines actions to be taken in the beginning, end or during runs.

* G4UserEventAction defines actions to be taken in the beginning, end or during events.

* G4UserStackingAction classifies the tracks according to their simulation priorities.

* G4UserTrackingAction allows the user to intervene at specific points during tracking.

* G4UserSteppingAction allows the user to specify actions during the stepping action.

Geant4 geometry implementation

Defining the Detector Geometry

Three conceptual layers:

* G4Solid – shape, size … (G4Box, G4Tubes, …)

* G4LogivcalVolume – daughter physical volumes, material, sensitivity, B/E field, visualization attributes, user limits, etc. (G4Material, G4VisAttributes, G4VSensitiveDetector)

* G4VPhysicalVolume – position, rotation, copy … (G4PVPlacement, G4PVParameterised). Physical volumes of the same type can share a logical volume (placed instances of the logical volume).

Geometrical hierarchy

Mother and daughter volumes: a volume is in its mother volume; position and rotation of the daughter volume is described with respect to the local coordinate system of the mother volume; daughter volumes cannot protrude from the mother volume; one and more volumes can be placed in the mother volume; the logical volume of the mother knows the physical volumes it contains.

World volume is the root volume of the hierarchy. The world volume must be a unique physical volume which fully contains all other volumes.

Overview on detector response in Geant4

Information extraction from the simulation application using Sensitive Detector, Hits and Hits Collection

User can attach to the Logical Volume a Sensitive Detector and can implement the detector to have many functionalities.

To do that:

* Create own sensitive detector (using class inheritance)
* Use Geant4 built-in tools. Primitive scorers.

```
G4VSolid* boxSolid = new G4Box("aBoxSolid", 1.0 * cm, 1.0 * cm, 1.0 * cm);

G4LogicalVolume* boxLog =
                    new G4LogicalVolume(boxSolid, matSilicon, "aBoxLog", 0, 0, 0);

G4VSensitiveDetector* sensitiveBox =
                    new MySensitiveDetector("MyDetector");

G4SDManager* SDManager = G4SDManager::GetSDMPointer();

SDManager → AddNewDetector(sensitiveBox);

boxLog → SetSensitiveDetector(sensitiveBox);
```

Classes implemented by user to implement Sensitive Detectors

* Sensitive Detector: MySensitiveDetector *inherited from* G4VSensitiveDetector
* Hit: MyHit *inherited from* G4VHit
* Hits Collections implemented in G4THitsCollection<hit class> template

Hit is snapshot of the physical interaction of a track or an accumulation of interactions of the tracks in the sensitive regions of the detectors. Hit objects are using the information from the particle steps. One need to create hit class(es) according to own needs.

Hits should be stored in the hit collections which are automatically associated to the G4Event object.

The information associated with hits may be processed in user action classes (G4UserEventAction, G4UserRunAction) to obtain a summary of the event/run.

Hit Collection of the event after its processing

* A G4Event object has a G4HCofThisEvent object at the end of the event processing (if event processing was successful). One can get the pointer to the G4HCofThisEvent object from G4Event.

* The G4HCofThisEvent object stores all hits collections created within the event. In the end of an event, one can process the hits of hits collections contained in G4HCofThisEvent in its own implementation of the EndOfEventAction method of the user event action class.

“Sensitive” elements that record information (hits) needed to simulate detector responses (digitization).

A Hit is created when a particle step is inside a detector, in contrast, a Digit represents the output of a detector:

* ADC/TDC count, trigger signal, etc.
* created from the information of the hits and/or other digits
* the base class of the Digits is the G4VDigi
* there is the G4TDigiCollection template class for storing Digi Collections.
* G4Event has a G4DCofThisEvent object, that is a container class for the digit collections.