

Montecarlo Methods for Medical Physics

Francesco Longo
(francesco.longo@ts.infn.it)

Summary of the Course

- Part1 (Monday 30.05)
 - General (and brief) introduction to Monte Carlo methods
 - Montecarlo methods in Medical Physics
- Part2 (Monday 30.05)
 - Introduction to the Geant4 toolkit
- Part3
 - Fundamentals of a Geant4 application (Tuesday 31.05)
 - Geometry, Physics, Particle Flux, Scoring needs (Tuesday 31.05 - Today)
- Laboratory (Next weeks)
 - Realisation of an example relevant to Medical Physics

Simulation basics

Geant4 simulation toolkit

- Modeling the experimental set-up
- Tracking particles through matter
- Interaction of particles with matter
- Modeling the detector response
- Run and event control
- **Accessory utilities** (*random number generators, PDG particle information, physical constants, system of units etc.*)
 - User interface
 - Interface to event generators
 - Visualisation (*of the set-up, tracks, hits etc.*)
 - Persistency
 - Analysis

Physics Lists

OO technology

- Openness to **extension** and **evolution**
new implementations can be added w/o changing the existing code
- Robustness and ease of **maintenance**
protocols and well defined dependencies minimize coupling

Strategic vision

Toolkit

- A set of compatible components
- each component is **specialised** for a specific functionality
 - each component can be **refined** independently to a great detail
 - components can be **integrated** at any degree of complexity
 - it is easy to provide (and use) **alternative** components
 - the user application can be **customised** as needed

To use Geant4, you have to...

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
 - 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
 - etc.

User classes

- **main()**
 - Geant4 does not provide *main()*.

Note : classes written in **yellow** are mandatory.
- Initialization classes
 - Use G4RunManager::**SetUserInitialization()** to define.
 - Invoked at the initialization
 - **G4VUserDetectorConstruction**
 - **G4VUserPhysicsList** ←
- Action classes
 - Use G4RunManager::**SetUserAction()** to define.
 - Invoked during an event loop
 - **G4VUserPrimaryGeneratorAction**
 - G4UserRunAction
 - G4UserEventAction
 - G4UserStackingAction
 - G4UserTrackingAction
 - G4UserSteppingAction

Physics in Geant4

- It is rather unrealistic to develop a uniform physics model to cover wide variety of particles and/or wide energy range.
- Much wider coverage of physics comes from mixture of theory-driven, parameterized, and empirical formulae. Thanks to polymorphism mechanism, both cross-sections and models (final state generation) can be combined in arbitrary manners into one particular process.
- Geant4 offers
 - EM processes
 - Hadronic processes
 - Photon/lepton-hadron processes
 - Optical photon processes
 - Decay processes
 - Shower parameterization
 - Event biasing techniques
 - And you can plug-in more

Physics Processes Provided by Geant4

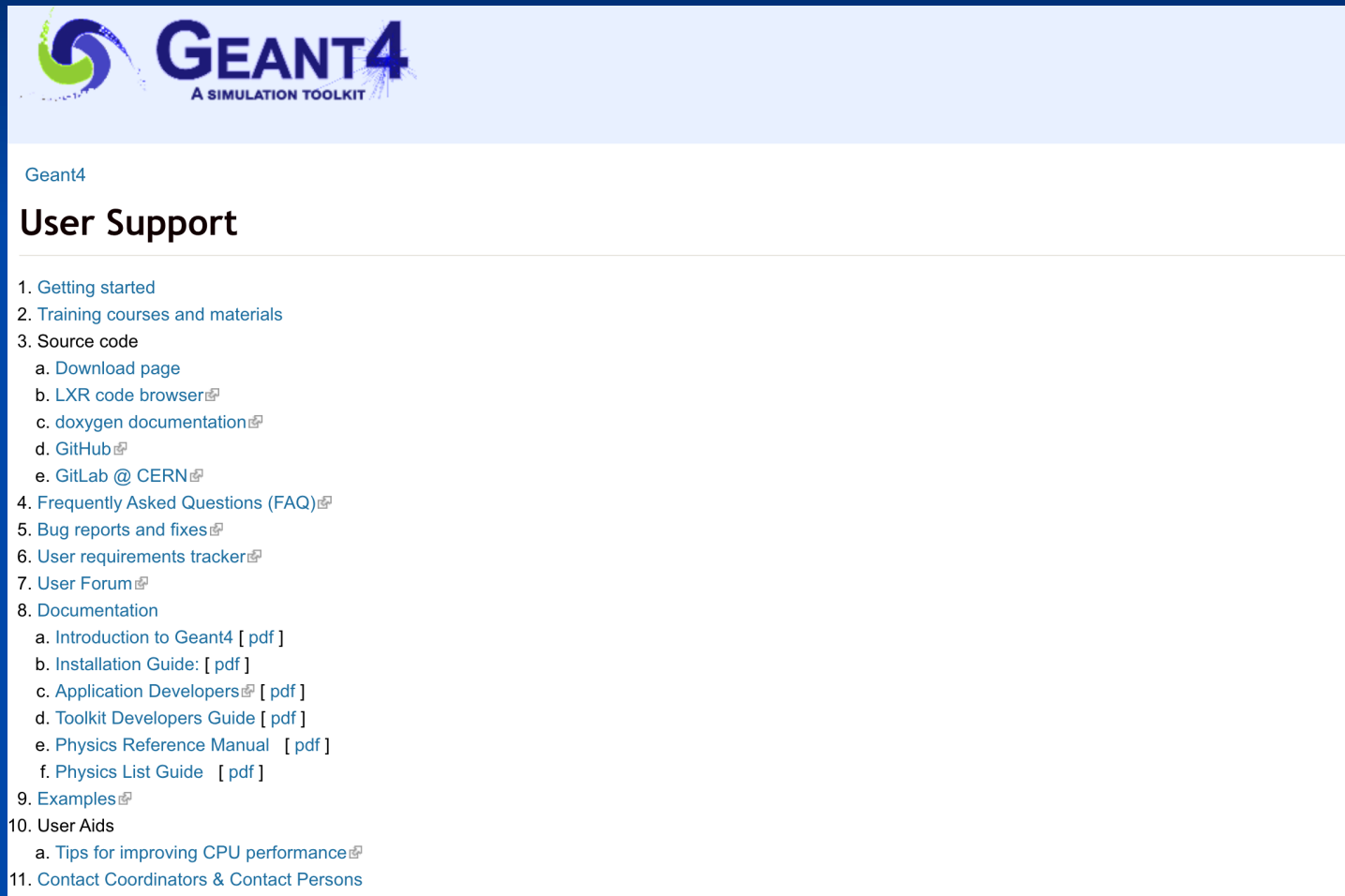
- EM physics
 - ☒ “standard” processes valid from ~ 1 keV to \sim PeV
 - ☒ “low-energy” Livermore/ Penelope valid from 250 eV to \sim PeV
 - ☒ optical photons
- Weak physics
 - ☒ decay of subatomic particles
 - ☒ radioactive decay of nuclei
- Hadronic physics
 - ☒ pure hadronic processes valid from 0 to ~ 100 TeV
 - ☒ γ^- , μ^- -nuclear valid from 10 MeV to \sim TeV
- Parameterized or “fast simulation” physics

Pre-packaged Physics Lists (2)

- Originally referred to as “hadronic physics lists” but include electromagnetic physics already
- Can be found on the Geant4 web page at
 - [PhysicsList Guide](#)
- Caveats:
 - these lists are provided as a “best guess” of the physics needed in a given case
 - The user is responsible for validating the physics for his own application and adding (or subtracting) the appropriate physics
 - “Trust, but verify.”
 - they are intended as starting points or templates

G4 home page

- <https://geant4.web.cern.ch/>



The screenshot shows the top part of the Geant4 website. At the top left is the Geant4 logo, which consists of a stylized green and blue swirl followed by the text 'GEANT4' in a bold, blue, sans-serif font. Below 'GEANT4' is the tagline 'A SIMULATION TOOLKIT' in a smaller, blue, sans-serif font. Below the logo is a light blue horizontal bar with the text 'Geant4' in a small, blue, sans-serif font. Below this bar is a white horizontal bar with the text 'User Support' in a bold, black, sans-serif font. Below the 'User Support' bar is a list of 11 items, each with a small blue icon to its right. The items are: 1. Getting started; 2. Training courses and materials; 3. Source code; 4. Frequently Asked Questions (FAQ); 5. Bug reports and fixes; 6. User requirements tracker; 7. User Forum; 8. Documentation; 9. Examples; 10. User Aids; 11. Contact Coordinators & Contact Persons.

Geant4

User Support

1. Getting started
2. Training courses and materials
3. Source code
 - a. Download page
 - b. LXR code browser
 - c. doxygen documentation
 - d. GitHub
 - e. GitLab @ CERN
4. Frequently Asked Questions (FAQ)
5. Bug reports and fixes
6. User requirements tracker
7. User Forum
8. Documentation
 - a. Introduction to Geant4 [pdf]
 - b. Installation Guide: [pdf]
 - c. Application Developers [pdf]
 - d. Toolkit Developers Guide [pdf]
 - e. Physics Reference Manual [pdf]
 - f. Physics List Guide [pdf]
9. Examples
10. User Aids
 - a. Tips for improving CPU performance
11. Contact Coordinators & Contact Persons

Hands On

Work on Medical Physics Example

- Check the Example documentation or the source code.
 - Find the geometrical info
 - Find the physics list
 - Find the particle source mechanism
 - Find the particle scoring mechanism
- Start designing your application ...

Particle Generation

G4VUserPrimaryGeneratorAction

- This class is one of mandatory user classes to **control the generation** of primaries.
 - This class itself **should NOT** generate primaries but **invoke** `GeneratePrimaryVertex()` method of primary generator(s) to make primaries.
- Constructor
 - Instantiate primary generator(s)
 - Set default values to it(them)
- **GeneratePrimaries()** method
 - Randomize particle-by-particle value(s)
 - Set these values to primary generator(s)
 - Never use hard-coded UI commands
 - Invoke **GeneratePrimaryVertex()** method of primary generator(s)

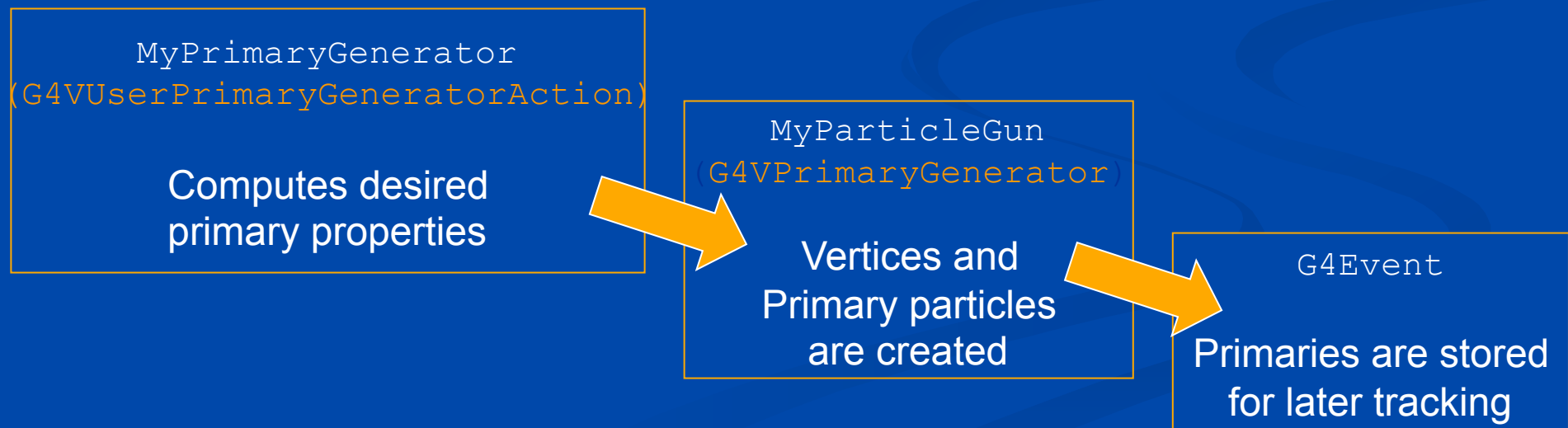
G4ParticleGun

- Concrete implementations of G4VPrimaryGenerator
 - A good example for experiment-specific primary generator implementation
- It shoots one primary particle of a certain energy from a certain point at a certain time to a certain direction.
 - Various set methods are available
 - Intercoms commands are also available for setting initial values
- One of most frequently asked questions is :

I want “particle shotgun”, “particle machinegun”, etc.
- Instead of implementing such a fancy weapon, in your implementation of UserPrimaryGeneratorAction, you can
 - Shoot random numbers in arbitrary distribution
 - Use set methods of G4ParticleGun
 - Use G4ParticleGun as many times as you want
 - Use any other primary generators as many times as you want to make overlapping events

Primary vertices and primary particles

- Primary vertices and primary particles are stored in G4Event in advance to processing an event.
 - `G4PrimaryVertex` and `G4PrimaryParticle` classes
 - These classes don't have any dependency to `G4ParticleDefinition` nor `G4Track`.
 - They will become "primary tracks" only at Begin-of-Event phase and put into a "stack"



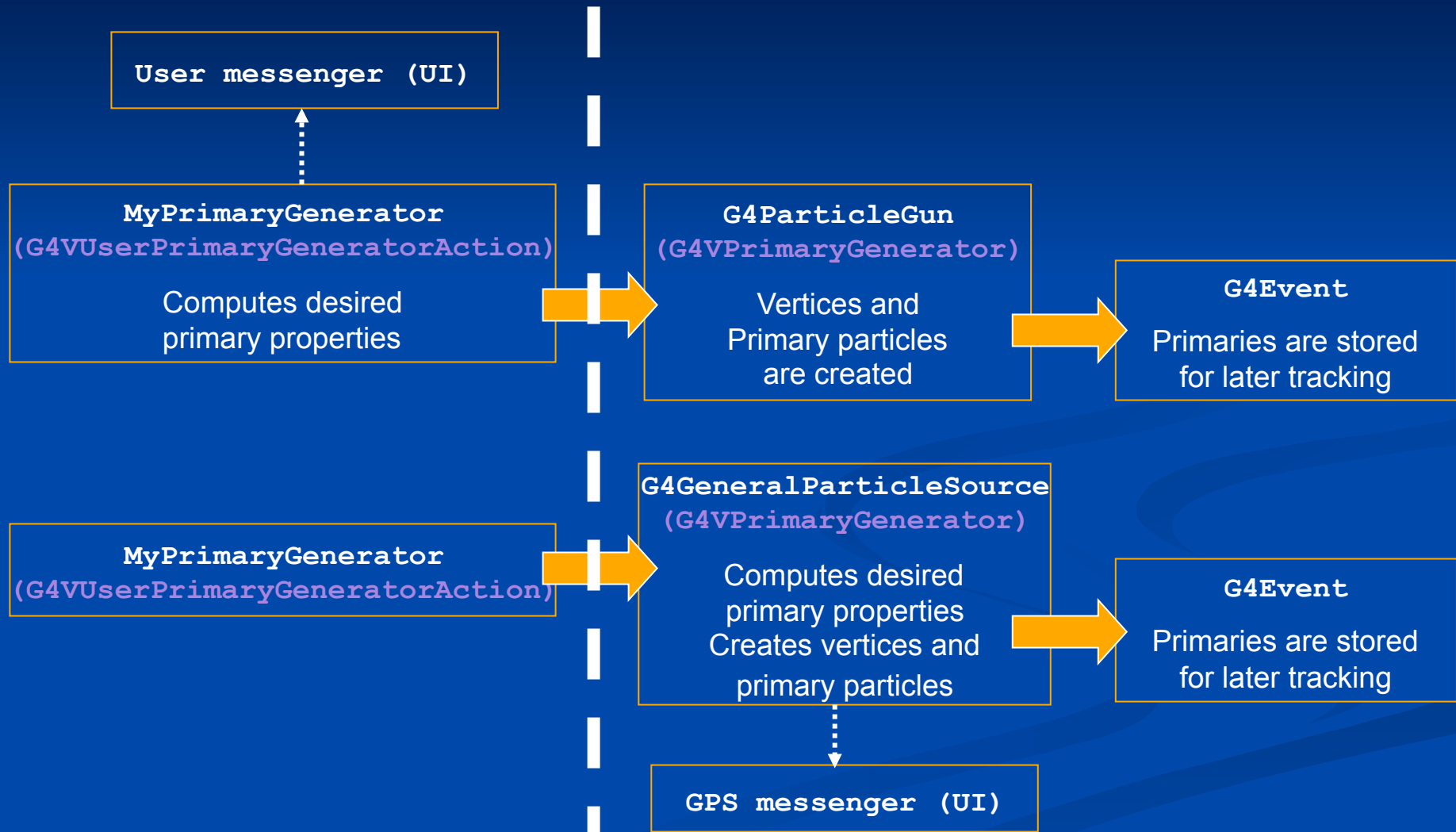
Motivation for GPS

- After first simple tutorial trials, modelling sources in realistic set-up soon requires relatively more complex sources
- G4ParticleGun can be used in most cases (as in the series of examples during this tutorial), but
 - users still needs to code (C++) almost every change and
 - add related UI commands for interactive control
- Requirements for advanced primary particle modelling are often common to many users in different communities
 - E.g. uniform vertex distribution on a surface, isotropic generation, energy spectrum,...

What is GPS?

- The General Particle Source (GPS) offers as **pre-defined** many common options for particle generation (energy, angular and spatial distributions)
 - GPS is a concrete implementation of G4VPrimaryGenerator (as G4ParticleGun but more advanced)
 - G4 class name: G4GeneralParticleSource (in the event category)
- User cases: space radiation environment, medical physics, accelerator (fixed target)
- First development (2000) University of Southampton (ESA contract), maintained and upgraded now mainly by QinetiQ and ESA

GPS vs G4ParticleGun



The main program

To use Geant4, you have to...

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
 - 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
 - etc.

User Classes needs

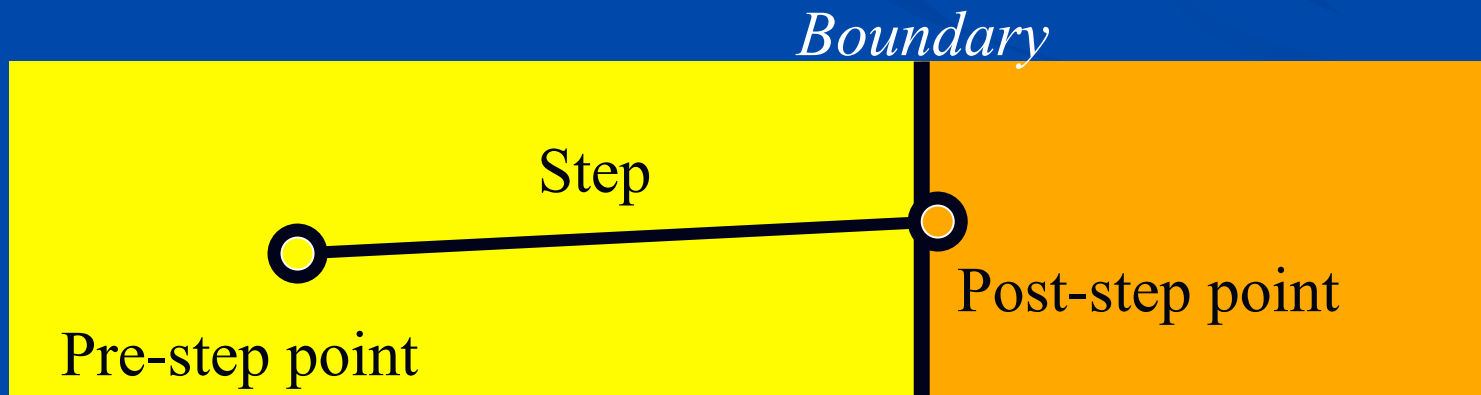
- Define material and geometry
 - ➔ G4VUserDetectorConstruction
- Select appropriate particles and processes and define production threshold(s)
 - ➔ G4VUserPhysicsList
- Define the way of primary particle generation
 - ➔ G4VUserPrimaryGeneratorAction
- **Define the way to extract useful information from Geant4**
 - ➔ G4UserSteppingAction, G4UserTrackingAction, etc.
 - ➔ G4VUserDetectorConstruction, G4UserEventAction, G4Run, G4UserRunAction
 - ➔ G4SensitiveDetector, G4VHit, G4VHitsCollection
 - ➔ G4PrimitiveScorers

Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
 - You have to add a bit of code to **extract information useful to you**.
- There are three ways:
 - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
 - You have an access to almost all information
 - Straight-forward, but do-it-yourself
 - Use Geant4 scoring functionality
 - Assign **G4VSensitiveDetector** to a volume
 - **Hits collection** is automatically stored in G4Event object, and automatically accumulated if **user-defined Run** object is used.
 - Use **Geant4 native scorers** to get specified quantities (dose, energy release, flux, path length, etc.)

Step in Geant4

- Step has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
- 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, and it **logically belongs to the next volume**.
 - Because one step knows materials of two volumes, boundary processes such as transition radiation or refraction could be simulated.
- **G4SteppingManager** class manages processing a step, a step is represented by **G4Step** class.
- **G4UserSteppingAction** is the optional user hook.



Geant4 Basic Examples

Basic Examples

- The set of basic examples is oriented to "novice" users and covering many basic general use-cases typical of an "application"-oriented kind of development.
- **Example B1**
 - Simple geometry with a few solids
 - Geometry with simple placements (G4PVPlacement)
 - **Scoring total dose in a selected volume user action classes**
 - Geant4 physics list (QBBC)
- **Example B2**
 - Simplified tracker geometry with global constant magnetic field
 - Geometry with simple placements (G4PVPlacement) and parametrisation (G4PVParametrisation)
 - **Scoring within tracker via G4 sensitive detector and hits**
 - Geant4 physics list (FTFP_BERT) with step limiter
 - Started from novice/N02 example
- **Example B3**
 - Schematic Positron Emitted Tomography system
 - Geometry with simple placements with rotation (G4PVPlacement)
 - Radioactive source
 - **Scoring within Crystals via G4 scorers**
 - Modular physics list built via builders provided in Geant4

Basic Examples

- The set of basic examples is oriented to "novice" users and covering many basic general use-cases typical of an "application"-oriented kind of development.
- **Example B4**
 - Simplified calorimeter with layers of two materials
 - Geometry with replica (G4PVReplica)
 - Scoring within layers in four ways: via user actions (a), via user own object (b), via G4 sensitive detector and hits (c) and via scorers (d)
 - Geant4 physics list (FTFP_BERT)
 - Histograms (1D) and ntuple saved in the output file
 - Started from novice/N03 example
- **Example B5**
 - A double-arm spectrometer with wire chambers, hodoscopes and calorimeters with a local constant magnetic field
 - Geometry with placements with rotation, replicas and parameterisation
 - Scoring within wire chambers, hodoscopes and calorimeters via G4 sensitive detector and hits
 - Geant4 physics list (FTFP_BERT) with step limiter
 - UI commands defined using G4GenericMessenger
 - Histograms (1D) and ntuple saved in the output file
 - Started from extended/analysis/A01

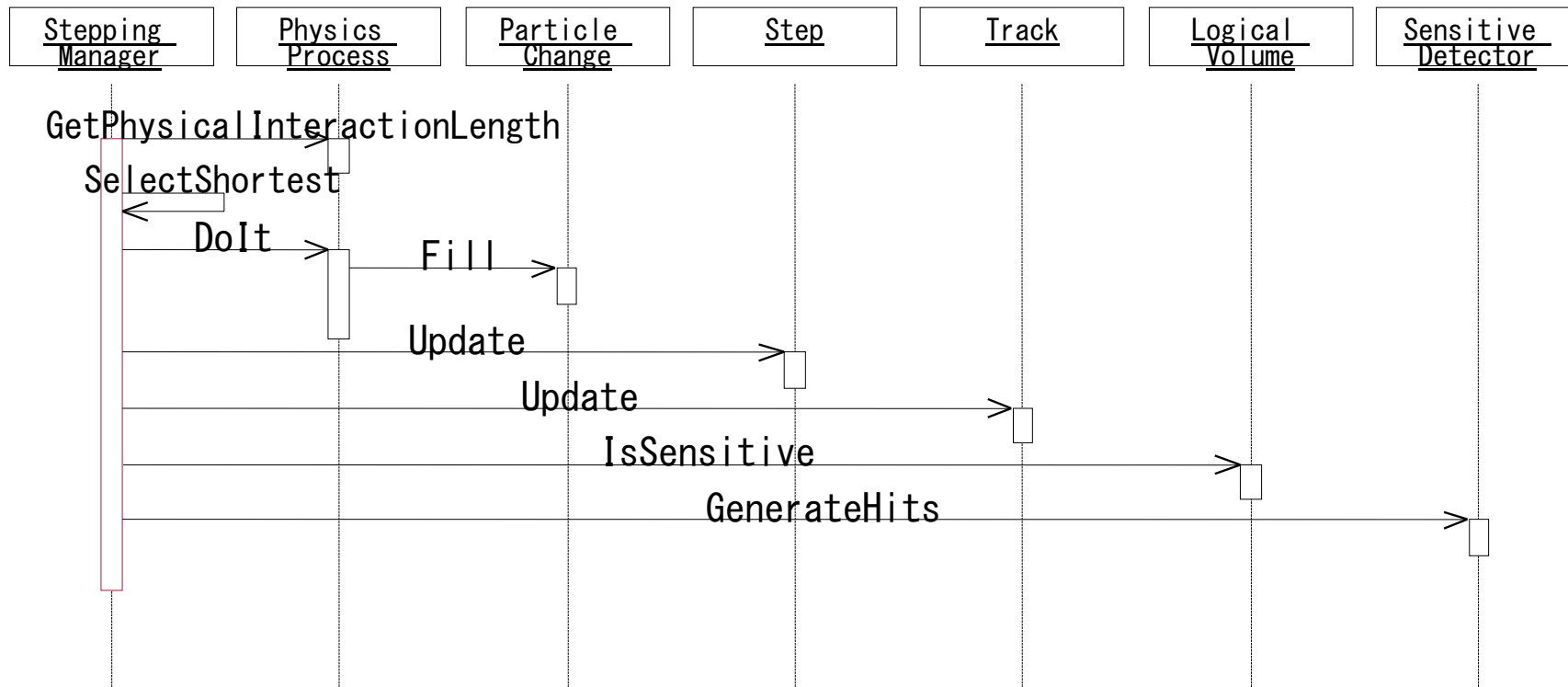
Scoring in Geant4

Extract useful information

- Given geometry, physics and primary track generation, Geant4 does proper physics simulation “silently”.
 - You have to add a bit of code to **extract information useful to you**.
- There are three ways:
 - Use user hooks (G4UserTrackingAction, G4UserSteppingAction, etc.)
 - You have full access to almost all information
 - Straight-forward, but do-it-yourself
 - Use Geant4 scoring functionality
 - Assign **G4VSensitiveDetector** to a volume
 - **Hit** is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive (or interested) part of your detector.
 - **Hits collection** is automatically stored in G4Event object, and automatically accumulated if **user-defined Run** object is used.
 - Use user hooks (G4UserEventAction, G4UserRunAction) to get event / run summary
 - Using the **G4 primitive scorers**

Sensitive detector

- A **G4VSensitiveDetector** object can be assigned to **G4LogicalVolume**.
- In case a step takes place in a logical volume that has a **G4VSensitiveDetector** object, this **G4VSensitiveDetector** is invoked with the **current G4Step** object.
 - You can implement your own sensitive detector classes, or use scorer classes provided by Geant4.



Defining a sensitive detector

- Basic strategy

```
G4LogicalVolume* myLogCalor = .....;  
  
G4VSensitiveDetector* pSensitivePart =  
    new MyDetector("/mydet");  
  
G4SDManager* SDMan = G4SDManager::GetSDMpointer();  
SDMan->AddNewDetector(pSensitivePart);  
myLogCalor->SetSensitiveDetector(pSensitivePart);
```

- Each detector **object** must have a unique name.
 - Some logical volumes can share one detector object.
 - More than one detector objects can be made from one detector class **with different detector name**.
 - One logical volume cannot have more than one detector objects. But, one detector object can generate more than one kinds of hits.
 - e.g. a double-sided silicon micro-strip detector can generate hits for each side separately.

Detector Sensitivities

March 2009

Geant4 Tutorial Introduction
F.Longo

Sensitive detector and hit

Sensitive detector and Hit

- Each Logical Volume can have a pointer to a sensitive detector.
 - Then this volume becomes **sensitive**.
- Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector.
- A sensitive detector creates hit(s) using the information given in G4Step object. The user has to provide his/her own implementation of the detector response.
- Hit objects, which are still the user's class objects, are collected in a G4Event object at the end of an event.

Hit class

Hit class

- Hit is a user-defined class derived from **G4VHit**.
- You can store various types information by implementing your own concrete Hit class. For example:
 - Position and time of the step
 - Momentum and energy of the track
 - Energy deposition of the step
 - Geometrical information
 - or any combination of above
- Hit objects of a concrete hit class must be stored in a dedicated collection which is instantiated from **G4THitsCollection template class**.
- The collection will be associated to a G4Event object via **G4HCofThisEvent**.
- Hits collections are accessible
 - through G4Event at the end of event.
 - to be used for analyzing an event
 - through G4SDManager during processing an event.
 - to be used for event filtering.

Implementation of Hit class

```
#include "G4VHit.hh"
class MyHit : public G4VHit
{
    public:
        MyHit(some_arguments);
        virtual ~MyHit();
        virtual void Draw();
        virtual void Print();
    private:
        // some data members
    public:
        // some set/get methods
};

#include "G4THitsCollection.hh"
typedef G4THitsCollection<MyHit> MyHitsCollection;
```

Sensitive detector class

Sensitive Detector class

- Sensitive detector is a user-defined class derived from G4VSensitiveDetector.

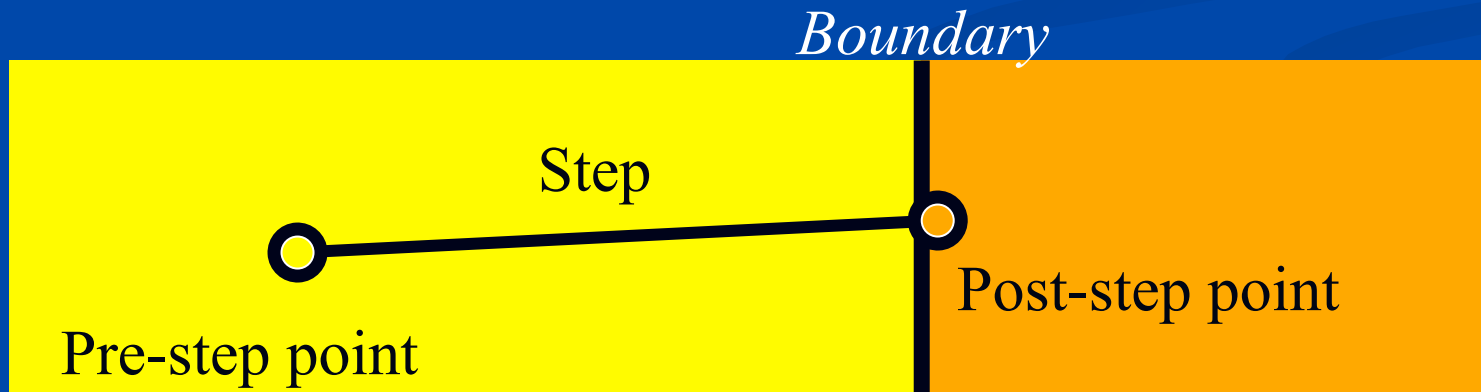
```
#include "G4VSensitiveDetector.hh"
#include "MyHit.hh"
class G4Step;
class G4HCofThisEvent;
class MyDetector : public G4VSensitiveDetector
{
public:
    MyDetector(G4String name);
    virtual ~MyDetector();
    virtual void Initialize(G4HCofThisEvent*HCE);
    virtual G4bool ProcessHits(G4Step*aStep,
                               G4TouchableHistory*ROhist);
    virtual void EndOfEvent(G4HCofThisEvent*HCE);
private:
    MyHitsCollection * hitsCollection;
    G4int collectionID;
};
```

Sensitive detector

- A **tracker** detector typically generates a **hit for every single step of every single (charged) track**.
 - A tracker hit typically contains
 - Position and time
 - Energy deposition of the step
 - Track ID
- A **calorimeter** detector typically generates a hit for every cell, and **accumulates energy deposition in each cell for all steps of all tracks**.
 - A calorimeter hit typically contains
 - Sum of deposited energy
 - Cell ID
- You can instantiate more than one objects for one sensitive detector class. Each object should have its unique detector name.
 - For example, each of two sets of detectors can have their dedicated sensitive detector objects. But, the functionalities of them are exactly the same to each other so that they can share the same class. See [examples/extended/analysis/A01](#) as an example.

Step

- Step has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
- 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, and it **logically belongs to the next volume**.
- Note that you must get the volume information from the “PreStepPoint”.



Implementation of Sensitive Detector

```
MyDetector::MyDetector(G4String detector_name)
                :G4VSensitiveDetector(detector_name),
                collectionID(-1)
{
    collectionName.insert("collection_name");
}
```

- In the constructor, define the name of the hits collection which is handled by this sensitive detector
- In case your sensitive detector generates more than one kinds of hits (e.g. anode and cathode hits separately), define all collection names.

Implementation of Sensitive Detector

```
void MyDetector::Initialize(G4HCofThisEvent*HCE)
{
    if(collectionID<0) collectionID = GetCollectionID(0);
    hitsCollection = new MyHitsCollection
        (SensitiveDetectorName, collectionName[0]);
    HCE->AddHitsCollection(collectionID, hitsCollection);
}
```

- Initialize() method is invoked **at the beginning of each event**.
- Get the unique ID number for this collection.
 - GetCollectionID() is a heavy operation. It should not be used for every events.
 - GetCollectionID() is available **after** this sensitive detector object is constructed and registered to G4SDManager. Thus, this method **cannot** be invoked in the constructor of this detector class.
- Instantiate hits collection(s) and attach it/them to **G4HCofThisEvent** object given in the argument.
- In case of calorimeter-type detector, you may also want to instantiate hits for all calorimeter cells with zero energy depositions, and insert them to the collection.

Implementation of Sensitive Detector

```
G4bool MyDetector::ProcessHits
(G4Step*aStep,G4TouchableHistory*ROhist)
{
  MyHit* aHit = new MyHit();
  ...
  // some set methods
  ...
  hitsCollection->insert(aHit);
  return true;
}
```

- This ProcessHits() method is invoked **for every steps** in the volume(s) where this sensitive detector is assigned.
- In this method, generate a hit corresponding to the current step (for tracking detector), or accumulate the energy deposition of the current step to the existing hit object where the current step belongs to (for calorimeter detector).
- Don't forget to collect geometry information (e.g. copy number) from **"PreStepPoint"**.
- Currently, returning boolean value is not used.

Implementation of Sensitive Detector

```
void MyDetector::EndOfEvent(G4HCofThisEvent*HCE)
{;}
```

- This method is invoked at the end of processing an event.
 - It is invoked even if the event is aborted.
 - It is invoked before UserEndOfEventAction.

**Sensitive detector
vs.
primitive scorer**

G4MultiFunctionalDetector

- **G4MultiFunctionalDetector** is a concrete class derived from G4VSensitiveDetector. It should be set to a logical volume as a kind of sensitive detector.
- It takes arbitrary number of **G4VPrimitiveSensitivity** classes. By registering G4VPrimitiveSensitivity classes, you can define the scoring detector of your need.
 - Each G4VPrimitiveSensitivity class accumulates **one physics quantity** for each physical volume.
 - For example, G4PSDoseScorer (a concrete class of G4VPrimitiveSensitivity provided by Geant4) accumulates dose for each cell.
- By using G4MultiFunctionalDetector and provided concrete G4VPrimitiveSensitivity classes, you are freed from implementing sensitive detector and hit classes.

Sensitive detector vs. primitive scorer

Sensitive detector

- You have to implement your own detector and hit classes.
- One hit class can contain many quantities. A hit can be made for each individual step, or accumulate quantities.
- Basically one hits collection is made per one detector.
- Hits collection is relatively compact.

Primitive scorer

- Many scorers are provided by Geant4. You can add your own.
- Each scorer accumulates one quantity for an event.
- G4MultiFunctionalDetector creates many collections (maps), i.e. one collection per one scorer.
- Keys of maps are redundant for scorers of same volume.

I would suggest to :

- ▶ Use primitive scorers
 - ▶ if you are **not** interested in recording each individual step **but** accumulating some physics quantities for an event or a run, and
 - ▶ if you do **not** have to have too many scorers.
- ▶ Otherwise, consider implementing your own sensitive detector.

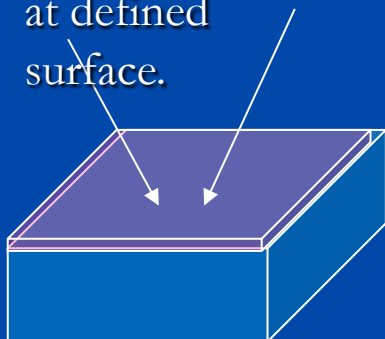
Primitive scorers

List of provided primitive scorers

- Concrete Primitive Scorers (See Application Developers Guide 4.4.6)
 - Track length
 - G4PSTrackLength, G4PSPassageTrackLength
 - Deposited energy
 - G4PSEnergyDeposit, G4PSDoseDeposit, G4PSChargeDeposit
 - Current/Flux
 - G4PSFlatSurfaceCurrent, G4PSSphereSurfaceCurrent, G4PSPassageCurrent, G4PSFlatSurfaceFlux, G4PSCellFlux, G4PSPassageCellFlux
 - Others
 - G4PSMinKinEAtGeneration, G4PSNofSecondary, G4PSNofStep

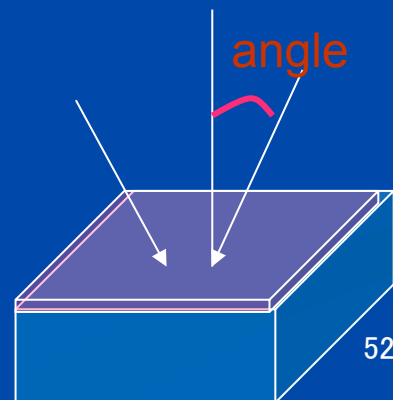
SurfaceCurrent :

Count number of injecting particles at defined surface.



SurfaceFlux :

Sum up $1/\cos(\text{angle})$ of injecting particles at defined surface

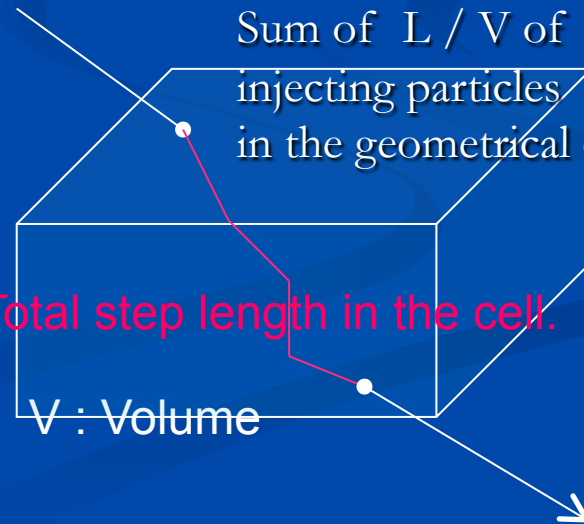


CellFlux :

Sum of L / V of injecting particles in the geometrical cell.

L : Total step length in the cell.

V : Volume



For example...

```
MyDetectorConstruction::Construct()
{ ... G4LogicalVolume* myCellLog = new G4LogicalVolume(...);
  G4VPhysicalVolume* myCellPhys = new G4PVParametrised(...);
  G4MultiFunctionalDetector* myScorer = new
    G4MultiFunctionalDetector("myCellScorer");
  G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);
  myCellLog->SetSensitiveDetector(myScorer);
  G4VPrimitiveSensitivity* totalSurfFlux = new
    G4PSFlatSurfaceFlux("TotalSurfFlux");
  myScorer->Register(totalSurfFlux);
  G4VPrimitiveSensitivity* totalDose = new G4PSDoseDeposit("TotalDose");
  myScorer->Register(totalDose);
}
```

No need of implementing sensitive detector !

Filter class

List of provided filter classes

- G4SDChargedFilter, G4SDNeutralFilter
 - Accept only charged/neutral tracks, respectively
- G4SDKineticEnergyFilter
 - Accepts tracks within the defined range of kinetic energy
- G4SDParticleFilter
 - Accepts tracks of registered particle types
- G4SDParticleWithEnergyFilter
 - Accepts tracks of registered particle types within the defined range of kinetic energy
- G4VSDFilter
 - Abstract base class which you can use to make your own filter

```
class G4VSDFilter
{
    public:
        G4VSDFilter(G4String name);
        virtual ~G4VSDFilter();
    public:
        virtual G4bool Accept(const G4Step*) const = 0;

```

...

For example...

```
MyDetectorConstruction::Construct()
```

```
{ ... G4LogicalVolume* myCellLog = new G4LogicalVolume(...);  
      G4VPhysicalVolume* myCellPhys = new G4PVParametrised(...);  
      G4MultiFunctionalDetector* myScorer = new G4MultiFunctionalDetector("myCellScorer");  
      G4SDManager::GetSDMpointer()->AddNewDetector(myScorer);  
      myCellLog->SetSensitiveDetector(myScorer);  
      G4VPrimitiveSensitivity* totalSurfFlux = new G4PSFlatSurfaceFlux("TotalSurfFlux");  
      myScorer->Register(totalSurfFlux);  
      G4VPrimitiveSensitivity* protonSurfFlux = new G4PSFlatSurfaceFlux("ProtonSurfFlux");  
      G4VSDFilter* protonFilter = new G4SDParticleFilter("protonFilter");  
      protonFilter->Add("proton");  
      protonSurfFlux->SetFilter(protonFilter);  
      myScorer->Register(protonSurfFlux);  
}
```


Work on Medical Physics Example

- Check the Example documentation or the source code.
 - Find the geometrical info
 - Find the physics list
 - Find the particle source mechanism
 - Find the particle scoring mechanism
- Start designing your application ...