Interaction with the Geant4 ke

GAP Cirrone, Luciano Pandola
INFN-LNS

The main ingredients

Optional user classes - 1

- Five concrete base classes whose virtual member functions the user may override to gain control of the simulation at various stages
  - G4User**Run**Action
  - G4User**Event**Action
  - G4User**Tracking**Action
  - G4User**Stacking**Action
  - G4User**Stepping**Action

e.g. actions to be done at the beginning and end of each event

- Each member function of the base classes has a dummy implementation (**not** purely virtual)
  - Empty implementation: does nothing

Optional user classes - 2

- The user may implement the member functions he desires in his/her derived classes
  - E.g. one may want to perform some action at each tracking step
- Objects of user action classes must be **registered** to the `G4(MT)RunManager` via the ActionInitialization

```
runManager->SetUserAction(new
  MyActionInitialization);
```

Geant4 terminology: an overview

- The following keywords are often used in Geant4
  - **Run, Event, Track, Step**
  - **Processes**: At Rest, Along Step, Post Step
  - **Cut** (or production threshold)

The Run (G4Run)

- As an analogy with a real experiment, a run of Geant4 starts with '**Beam On**'
- Within a run, the User **cannot change**
  - The detector setup
  - The physics setting (processes, models)
- A Run is a collection of events with the same detector and physics conditions
- At the beginning of a Run, geometry is optimised for navigation and cross section tables are (re)calculated
- The G4RunManager class manages the processing of each Run, represented by:
  - **G4Run** class
  - **G4UserRunAction** for an optional User hook

The Event (G4Event)

- An Event is the basic unit of simulation in Geant4
- At the beginning of processing, primary tracks are generated and they are pushed into a stack
- A track is popped up from the stack one-by-one **and 'tracked'**
  - Secondary tracks are also pushed into the stack
  - When the stack gets empty, the processing of the event is completed
- **`G4Event`** class represents an event. At the end of a successful event it has:
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as outputs)
- **`G4EventManager`** class manages the event
- **`G4UserEventAction`** is the optional User hook

The Step (G4Step)

- `G4Step` represents a step in the particle propagation
- A G4Step object stores transient information of the step
  - In the tracking algorithm, G4Step is updated each time a process is invoked
- You can extract information from a step after the step is completed
  - Both, the **`ProcessHits()`** method of your sensitive detector and **`UserSteppingAction()`** of your step action class file get the pointer of G4Step
  - Typically , you may retrieve information in these functions (for example fill histograms in Stepping action)

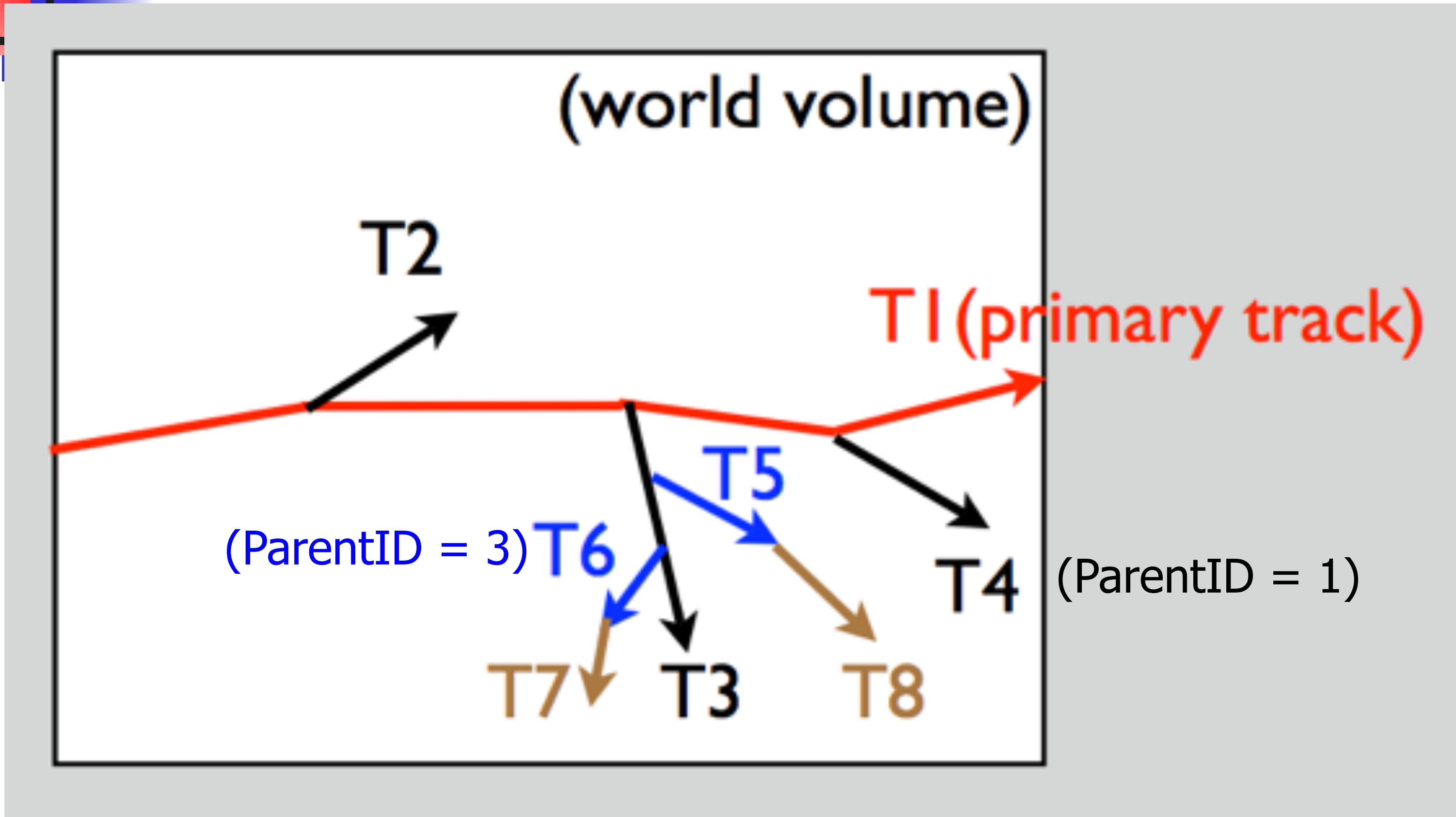
The Track (G4Track)

- The Track is a **snapshot of a particle** and it is represented by the **`G4Track`** class
  - It keeps ‘current’ information of the particle (i.e. energy, momentum, position, polarization, ..)
  - It is updated after every step
- The track object is deleted when
  - It goes outside the world volume
  - It disappears in an interaction (decay, inelastic scattering)
  - It is slowed down to zero kinetic energy and there are no 'AtRest' processes
  - It is manually killed by the user
- No track object **persists** at the end of the event
- **`G4TrackingManager`** class manages the tracking
- **`G4UserTrackingAction`** is the optional User hook

Run, Event and Tracks

- **One Run consists of**
  - Event #1 (track #1, track #2, ...)
  - Event #2 (track #1, track #2, ...)
  - ....
  - Event #N (track #1, track #2, ...)

Example of an Event and Tracks



- Tracking order follows **'last in first out'** rule:
  T1 -> T4 -> T3 -> T6 -> T7 -> T5 -> T8 -> T2

```
*****************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 87,   Parent ID = 1
*****************************************************************************************
```

```
*********************************************************************************************************
* G4Track Information:   Particle = e-,   Track ID = 1,   Parent ID = 0
*********************************************************************************************************
Step#     X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
0       711      711    -0.275      5.98        0        0         0 acceleratorBox initStep
1       707      707     -0.26      5.98 1.43e-25     4.99      4.99     targetB Transportation
2       707      707     -0.26      5.98 3.91e-05  3.8e-05      4.99     targetB msc
3       707      707    -0.247      5.85    0.127    0.101      5.09     targetB msc
4       707      707     -0.25      5.76    0.091    0.101      5.19     targetB msc
5       707      707    -0.258      5.62    0.145    0.101      5.29     targetB msc
6       707      707    -0.254       5.5    0.117    0.101      5.39     targetB msc
7       707      707    -0.231       5.4    0.104    0.101      5.49     targetB msc
8       707      707     -0.21      5.24    0.156    0.101      5.59     targetB msc
9       707      707    -0.186         5    0.237    0.101      5.69     targetB msc
10       707      706    -0.167      4.93   0.0761    0.101      5.79     targetB msc
11       707      706     -0.13       4.8    0.125    0.101      5.89     targetB msc
12       706      706    -0.108      4.71   0.0928    0.101      5.99     targetB msc
13       706      706    -0.106      4.63   0.0789    0.101      6.09     targetB msc
14       706      706   -0.0934      4.53   0.0981    0.101      6.19     targetB msc
15       706      706   -0.0775      4.44   0.0882    0.101      6.29     targetB msc
16       706      706   -0.0806      4.36   0.0796    0.101      6.39     targetB msc
17       706      706   -0.0749       4.2    0.162    0.101       6.5     targetB msc
18       706      706   -0.0805      4.09     0.11    0.101       6.6     targetB msc
19       706      707   -0.0897         4   0.0959    0.101       6.7     targetB msc
20       706      707    -0.125      3.89    0.104    0.101       6.8     targetB msc
21       706      707    -0.152      3.79    0.106    0.101       6.9     targetB msc
22       706      707    -0.189      3.68    0.111    0.101         7     targetB msc
23       706      707     -0.24      3.56    0.119    0.101       7.1     targetB msc
24       706      707    -0.312      3.41    0.149    0.101       7.2     targetB msc
25       706      707    -0.391      3.33   0.0804    0.101       7.3     targetB msc
26       706      707    -0.467      3.26   0.0665    0.101       7.4     targetB msc
27       705      707    -0.547      3.15    0.108    0.101       7.5     targetB msc
28       705      707    -0.627      3.04    0.112    0.101       7.6     targetB msc
29       705      707    -0.708      2.94   0.0994    0.101       7.7     targetB msc
30       705      707    -0.776      2.87   0.0747    0.101       7.8     targetB msc
31       705      707    -0.805      2.78   0.0913    0.101       7.9     targetB msc
```

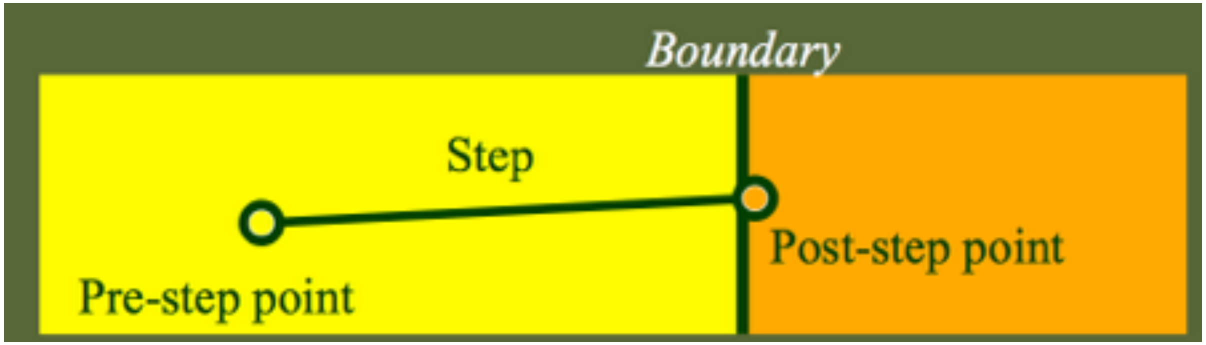
```
Step#     X(mm)     Y(mm)     Z(mm) KinE(MeV)  dE(MeV) StepLeng TrackLeng  NextVolume ProcName
    0 -1.87e+03       6.1      5.41   0.00138        0        0         0 physicalTreatmentRoom initStep
    1 -1.87e+03      6.11      5.39  0.000253  0.00112   0.0481    0.0481 physicalTreatmentRoom msc
    2 -1.87e+03      6.12      5.39         0 0.000253   0.0088    0.0569 physicalTreatmentRoom eIoni
```

Example: retrieving information from tracks

```
// retrieving information from tracks (given the G4Track object "track"):

if(track -> GetTrackID() != 1)  {
    G4cout << "Particle is a secondary" << G4endl;

    // Note in this context, that primary hadrons might loose their identity
    if(track -> GetParentID() == 1)
                    G4cout << "But parent was a primary" << G4endl;

    G4VProcess* creatorProcess = track -> GetCreatorProcess();

    if(creatorProcess -> GetProcessName()  == "LowEnergyIoni") {
          G4cout << "Particle was created by the Low-Energy " <<
                    << "Ionization process" << G4endl;
    }
}
```

The Step in Geant4

- The G4Step has the information about the **two points** (pre-step and post-step) and the **'delta'** information of a particle (energy loss on the step, ...)
- Each point knows the volume (and the 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



- **`G4SteppingManager`** class manages processing a step; a 'step' in represented by the **`G4Step`** class
- **`G4UserSteppingAction`** is the optional User hook

The G4Step object

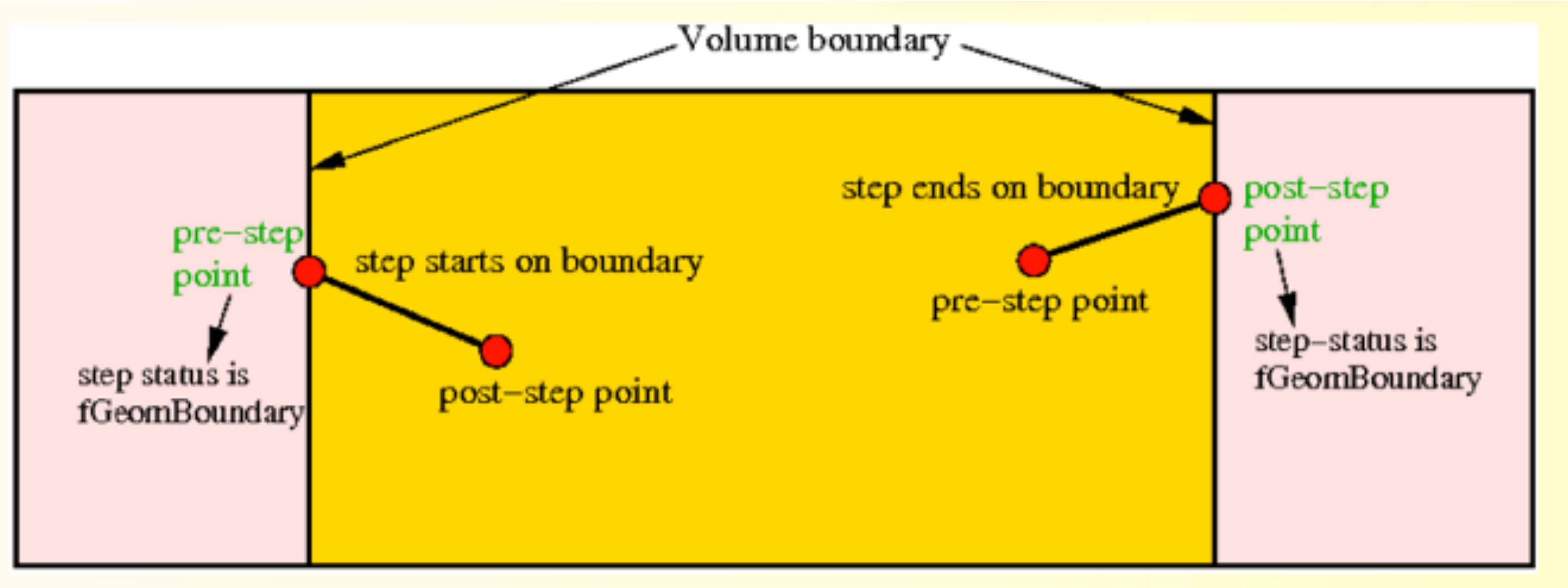
- A **`G4Step`** object contains
  - The two endpoints (pre and post step) so one has access to the volumes containing these endpoints
  - Changes in particle properties between the points
    - Difference of particle energy, momentum, ...
    - Energy deposition on step, step length, time-of-flight, ...
  - A pointer to the associated **`G4Track`** object
- **`G4Step`** provides many Get methods to access these information or object istances
  - **`G4StepPoint* GetPreStepPoint(), ......`**

The geometry boundary

- To check, if a step ends on a boundary, one may compare if the physical volume of pre and post-step points are equal
- One can also use the **step status**
  - Step Status provides information about the process that restricted the step length
  - It is attached to the step points: the pre has the status of the previous step, the post of the current step
  - If the status of POST is "**`fGeometryBoundary`**" the step ends on a volume boundary (does not apply to word volume)
  - To check if a step starts on a volume boundary you can also use the step status of the PRE-step point

Step concept and boundaries

Illustration of step starting and ending on boundaries



Geant4 terminology: an overview

	Object	Description
Run	G4Run	Largest unit of simulation, that consist of a sequence of events: If a defined number of events was processed a run is finished.
Event	G4Event	Basic simulation unit in Geant4: If a defined number of primary tracks and all resulting secondary tracks were processed an event is over.
Track	G4Track	A track is NOT a collection of steps: It is a snapshot of the status of a particle after a step was completed (but it does NOT record previous steps). A track is deleted, if the particle leaves world, has zero kinetic energy, ...
Step	G4Step	Represents a particle step in the simulation and includes two points (pre-step point and post-step point).

Example of usage of the hook user classes - 1

- **`G4UserRunAction`**
  - Has two methods (**`BeginOfRunAction()`** and **`EndOfRunAction()`**) and can be used e.g. to initialise, analyse and store histogram
  - Everything User want to know at this stage
- **`G4UserEventAction`**
  - Has two methods (**`BeginOfEventAction()`** and **`EndOfEventAction()`**)
  - One can apply an event selection, for example
  - Access the hit-collection and perform the event analysis

Example of usage of the hook user classes - 2

- **`G4UserStakingAction`**
  - Classify priority of tracks
- **`G4UserTrackingAction`**
  - Has two methods (**`PreUserTrakingAction()`** **`and PostUserTrackinAction()`**)
  - For example used to decide if trajectories should be stored
- **`G4UserSteppingAction`**
  - Has a method which is invoked at the end of a step

Part II: Retrieving information from steps and tracks

Example:
check if step is on boundaries

```
// in the source file of your user step action class:

#include "G4Step.hh"

UserStepAction::UserSteppingAction(const G4Step* step) {

  G4StepPoint* preStepPoint = step -> GetPreStepPoint();
  G4StepPoint* postStepPoint = step -> GetPostStepPoint();

  // Use the GetStepStatus() method of G4StepPoint to get the status of the
  // current step (contained in post-step point) or the previous step
  // (contained in pre-step point):
  if(preStepPoint -> GetStepStatus() == fGeomBoundary) {

      G4cout << "Step starts on geometry boundary" << G4endl;
  }
  if(postStepPoint -> GetStepStatus() == fGeomBoundary) {

      G4cout << "Step ends on geometry boundary" << G4endl;

      // You can retrieve the material of the next volume through the
      // post-step point :
      G4Material* nextMaterial = step -> GetPostStepPoint()->GetMaterial();
  }
}
```

Example: step information in SD

```
// in source file of your sensitive detector:


MySensitiveDetector::ProcessHits(G4Step* step,
                                 G4TouchableHistory*) {

  // Total energy deposition on the step (= energy deposited by energy loss
  //  process and energy of secondaries that were not created since their
  // energy was < Cut):
  G4double energyDeposit = step -> GetTotalEnergyDeposit();

  // Difference of energy , position and momentum of particle between pre-
  // and post-step point
  G4double deltaEnergy = step -> GetDeltaEnergy();
  G4ThreeVector deltaPosition = step -> GetDeltaPosition();
  G4double deltaMomentum = step -> GetDeltaMomentum();

  // Step length
  G4double stepLength = step -> GetStepLength();

}
```

Something more about tracks

- After each step the track can change its state
- The status can be (in red can only be set by the User)

Track Status	Description
fAlive	The particle is continued to be tracked
fStopButAlive	Kin. Energy = 0, but AtRest process will occur
fStopAndKill	Track has lost identity (has reached world boundary, decayed, ...), Secondaries will be tracked
fKillTrackAndSecondaries	Track and its secondary tracks are killed
fSuspend	Track and its secondary tracks are suspended (pushed to stack)
fPostponeToNextEvent	Track but NOT secondary tracks are postponed to the next event (secondaries are tracked in current event)

Particles in Geant4

- A particle in general has the following **three sets of properties**:
  - Position/geometrical info
    - **`G4Track`** class (representing a particle to be tracked)
  - Dynamic properties: momentum, energy, spin,..
    - **`G4DynamicParticle`** class
  - Static properties: rest mass, charge, life time
    - **`G4ParticleDefinition`** class
- All the **`G4DynamicParticle`** objects of the same kind of particle share the same **`G4ParticleDefinition`**

Particles in Geant4

Class	What does it represent?	What does it contain?
G4Track	Represents a particle that travels in space and time	Information relevant to tracking the particle, e.g. position, time, step,..., and *dynamic information*
G4DynamicParticle	Represents a particle that is subject to interactions with matter	Dynamic information, e.g. particle momentum, kinetic energy, ..., and *static information*
G4ParticleDefinition	Defines a physical particle	Static information, e.g. particle mass, charge, ... Also physics processes relevant to the particle

Examples: particle information from step/track

```
#include "G4ParticleDefinition.hh"
#include "G4DynamicParticle.hh"
#include "G4Step.hh"
#include "G4Track.hh"

// Retrieve from the current step the track  (after PostStepDoIt of step is
// completed):
G4Track* track = step -> GetTrack();

 // From the track you can obtain the pointer to the dynamic particle:
const G4DynamicParticle* dynParticle = track -> GetDynamicParticle();

 // From the dynamic particle, retrieve the particle definition:
G4ParticleDefinition* particle = dynParticle -> GetDefinition();

 // The dynamic particle class contains e.g. the kinetic energy after the step:
G4double kinEnergy = dynParticle -> GetKineticEnergy();

 // From the particle definition class you can retrieve static information like
 // the particle name:
G4String particleName = particle -> GetParticleName();

 G4cout << particleName << ": kinetic energy of "
          << kinEnergy/MeV << " MeV"
          << G4endl;
```

Write an ASCII file

1. Add to the include list of your class the <fstream> header file
   - This will allow to use the C++ libraries for stream on file
2. Put into the class declaration (file .hh) an ofstream (=output file stream) object (or pointer):

   ```
   std::ofstream myFile;
   ```

   In this way, the file object will be visible in all methods of the class
3. Open the file, in the class constructor, or into a specific method:

   myFile.open("filename.out", std::ios::trunc);

   - To append data to an existing file, you must specify std::ios::app

```
  std::ofstream myFile("Data.out", std::ios::app);
 myFile  <<      eKin                << '\t' << "    "
         <<    EventID               <<   '\t'<< "    "
          <<   PreStepX               << '\t' << "    "
          <<   PreStepY               << '\t' << "    "
         <<   PreStepZ               << '\t' << "    "
         << G4endl;
```

- This could be for instance the EndOfEventAction() of the G4UserEventAction user class or in the UserSteppingAction class

Data analysis in Geant4

Basic classes for data analysis have recently been implemented in Geant4 (g4analysis)

- ✦Support for histograms and ntuples
- ✦Output in ROOT, XML, HBOOK and CSV (ASCII)

The resulting files can be opened and analyzed by tools such as: Gnuplot, Excel, OpenOffice, Matlab, Origin, ROOT, PAW,...

Appropriate only for easy/quick analysis: for advanced tasks, the user must write his/her own code and to use an external analysis tool

Native Geant4 analysis classes

- A basic analysis interface is available in Geant4 for histograms (1D and 2D) and ntuples
  - ➡Make life easier because they are MT-compliant (no need to worry about the interference of threads)
- Unique interface to support different output formats ROOT, AIDA XML, CSV and HBOOK
  - ➡Code is the same, just change one line to switch from one to an other
- Everything done via the public analysis interface G4AnalysisManager
  - ➡Singleton class: Instance()
  - ➡UI commands available for creating histograms at run-time and setting their properties

- Selection of output format is hidden in a user-defined .hh file
- All the rest of the code unchanged
  - ➡Unique interface

```
#ifndef MyAnalysis_h
#define MyAnalysis_h 1

#include "g4root.hh"
//#include "g4xml.hh"
//#include "g4csv.hh"  // can be used only
with ntuples

#endif
```

Open file and book histograms

```
#include "MyAnalysis.hh"

void MyRunAction::BeginOfRunAction(const G4Run* run)
{
  // Get analysis manager
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->SetVerboseLevel(1);
  man->SetFirstHistoId(1);

  // Creating histograms
  man->CreateH1("h","Title", 100, 0., 800*MeV);
  man->CreateH1("hh","Title",100,0.,10*MeV);

 // Open an output file
  man->OpenFile("myoutput");
}
```

Fill histograms and close

```
#include "MyAnalysis.hh"
void MyEventAction::EndOfEventAction(const G4Run* aRun)
{
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->FillH1(1, fEnergyAbs);
  man->FillH1(2, fEnergyGap);
}
void MyRunAction::EndOfRunAction(const G4Run* aRun)
{
  G4AnalysisManager* man = G4AnalysisManager::Instance();
  man->Write();
  man->CloseFile();
}
MyRunAction::~MyRunAction()
{
  delete G4AnalysisManager::Instance();
}
```

The End