Geant4: A Simulation toolkit

O. Stézowski and M. Pinto

With many thanks to the Geant4 community !!!!
The roadmap of the week

W1: installation / running a G4 application

W2: Primary generator, GPS, physics list

W3: Geometries!

W4: Sensitive detectors / user’s actions

NOW, HOW does it really work?

w1: 3:00, Monday
w2: 3:00, Tuesday
w3: 4:30, Wednesday
w4: 3:00, Thursday
W4: Sensitive detectors / user’s actions

- General aspects
- User’s actions
- Sensitive detector
To extract information from G4

Given geometry, physics and primary track generation, G4 does proper physics simulation “silently”

👉 You have to add a bit of code **to extract information useful to you**

There are two ways:

❶ Use user hooks (**G4UserTrackingAction**, **G4UserSteppingAction**, etc.)
   - You have full access to almost all information
   - Straight-forward, but do-it-yourself

❷ Use Geant4 scoring functionality
   - Assign **G4VSensitiveDetector** to a volume i.e. make it a detector!
   - It is based on **Hits**, 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

+ **G4Digi, G4DigitizerModule** ... the electronic chain could be emulated!
Mapping in Geant4: touchable

As any real detector, a mapping is required to know where things happen

- Done using the `copy#` which is set at the `G4VPhysicalVolume` level

```
aphysBox = new G4PVPlacement(
    0,         
    G4ThreeVector(X_C, Y_C, Z_C),
    alogicBox,
    "PBlueCube",
    logicWorld,
    false,
    0); // THIS IS the copy number!
```

The `copy#` is configurable by the user [collaboration policy]

Note: `G4PVDivision, ...` deal internally with `copy #`,
- to see whether or not it fits the user’s requirements
Mapping in Geant4: touchable

There are many different possibilities

The particle goes through $2 \rightarrow 7 \rightarrow 10$

a unique copy# per detectors

The particle goes through $0:2 \rightarrow 1:1 \rightarrow 1:4$

One has to deal with sequences of numbers. For that G4 provides 

G4TouchableHistory
W4: Sensitive detectors / user’s actions

- General aspects
- User’s actions
- Sensitive detector
The user’s application

Building an application requires to put together 3 mandatory bricks*

the detector construction - the description of the physics - the primary generator

+ many other hooks
but not mandatory

```c++
class ARedSphereConstruction : public G4VUserDetectorConstruction {
// the virtual method to be implemented by the user
virtual G4VPhysicalVolume* Construct();
};

class AGammaGun : public G4VUserPrimaryGeneratorAction {
// the virtual method to be implemented by the user
virtual void GeneratePrimaries(G4Event* anEvent);
};

class AnElectroMagneticPhysicsList: public G4VUserPhysicsList {
// the virtual method to be implemented by the user
void ConstructParticle();
// the virtual method to be implemented by the user
void ConstructProcess();
// the virtual method to be implemented by the user
void SetCuts();
};
```

// The User's main program to control / run simulations
int main(int argc, char** argv)
{
    // Construct the run manager, necessary for G4 kernel to control everything
    G4RunManager *theRunManager = new G4RunManager();

    // now set the others user actions
    theRunManager->SetUserAction(new MyRunAction());
    theRunManager->SetUserAction(new MyEventAction());
    theRunManager->SetUserAction(new MyTrackingAction());
    theRunManager->SetUserAction(new MySteppingAction());

    return 0;
}
G4UserXXXAction should be defined by the user
Where are they called, what is provided by Geant4 ...

Where are they called in the G4 loop?

Start run # 1:
Start event # i
Start track # j
Start step # k

Stop step # k
Stop track # j
Stop event # i
Stop run # 1

User's hooks to extract information
User's hooks to extract information

G4UserXXXAction should be defined by the user
Where are they called, what is provided by Geant4 ...

class MyRunAction : public G4UserRunAction
{
    public:
    virtual void BeginOfRunAction(const G4Run *therun);
    virtual void EndOfRunAction(const G4Run *therun);
};
User’s hooks to extract information

G4UserXXXAction should be defined by the user
Where are they called, what is provided by Geant4 ...

Where are they called?
Start run # 1:
Start event # i
Start track # j
Start step # k
Stop step # k
Stop track # j
Stop event # i
Stop run # 1

class MyRunAction : public G4UserRunAction {
    public:
    virtual void BeginOfRunAction(const G4Run *therun);
    virtual void EndOfRunAction(const G4Run *therun);
};

class MyEventAction : public G4UserEventAction {
public:
    virtual void BeginOfEventAction(const G4Event *event); 
    virtual void EndOfEventAction(const G4Event *event);
};
User's hooks to extract information

G4UserXXXAction should be defined by the user
Where are they called, what is provided by Geant4 ...

Where are they called in the G4 loop?
User's hooks to extract information

Start run # 1:
Start event # i
Start track # j
Start step # k
Stop step # k
Stop track # j
Stop event # i
Stop run # 1

class MyRunAction : public G4UserRunAction
{
    public:
    virtual void BeginOfRunAction(const G4Run *therun);
    virtual void EndOfRunAction(const G4Run *therun);
};

class MyEventAction : public G4UserEventAction
{
    public:
    virtual void BeginOfEventAction(const G4Event *event);
    virtual void EndOfEventAction(const G4Event *event);
};

class MyTrackingAction : public G4UserTrackingAction
{
    public:
    virtual void PreUserTrackingAction(G4Track *track);
    virtual void PostUserTrackingAction(G4Track *track);
};
G4UserXXXAction should be defined by the user
Where are they called, what is provided by Geant4 ...

class MyRunAction : public G4UserRunAction
{
public:
virtual void BeginOfRunAction(const G4Run *therun);
virtual void EndOfRunAction(const G4Run *therun);
};

class MyEventAction : public G4UserEventAction
{
public:
virtual void BeginOfEventAction(const G4Event *event);
virtual void EndOfEventAction(const G4Event *event);
};

class MyTrackingAction : public G4UserTrackingAction
{
public:
virtual void PreUserTrackingAction(G4Track *track);
virtual void PostUserTrackingAction(G4Track *track);
};

class MySteppingAction : public G4UserSteppingAction
{
public:
virtual void UserSteppingAction(const G4Step *step);
};
The user’s application

TODO List

Write your Run, Event, Tracking and Stepping Action. Run adding one by one.

In the RunAction, print out:
  begin - it is starting [and the world’s name]
  end   - the run ID and the number of simulated events
In the EventAction, print out:
  begin - it starts every N (=10) events
  end   - it ends + the number of primary vertexes
In the TrackingAction, print out:
  begin - the track ID, the parent ID and what is tracked
  end   - the current volume name, total and kinetic energy, velocity
In the SteppingAction, print out:
  the step length if it is the first step in a volume
W4: Sensitive detectors / user’s actions

- General aspects
- User’s actions
- Sensitive detector
Sensitive detector - principle

Goal: avoid going into tracking/stepping details for the user

Mechanism:
1. attach to a volume a sensitivity
2. at any step in it keep the required information in a hit
3. push it on a stack [in a collection]
4. the list of hits is available in the G4Event

⇒ retrieve info at the EndOfEventAction
A sensor is added to the SD manager & a logical volume

```
// Now add a blue cube to the world
asolidBox = new G4Box("BlueCube", Side/2., Side/2., Side/2.);
alogicBox =
    new G4LogicalVolume(asolidBox, CubeMaterial, "LBlueCube", 0, 0, 0);

// the cube is blue
visatt = new G4VisAttributes( G4Colour(0.0, 0.0, 1.0) );
visatt->SetBranchVisibility(true);
alogicBox->SetBranchVisibility(visatt);
// and is a sensitive detector of type MySD
MySD *sd = new MySD(SDname="/MySD");
G4SDManager* SDman = G4SDManager::GetSDMpointer();
SDman->AddNewDetector(sd);
alogicBox->SetBranchSensitiveDetector(sd);

aphysiBox = new G4PVPlacement( ... )
```

keep:
energy deposited, detectorID ...

HitsCollection

G4Event

Sensitivity detector, implementation
A sensor is added to the SD manager & a logical volume

```cpp
// Now add a blue cube to the world
asolidBox = new G4Box("BlueCube", Side/2., Side/2., Side/2.);
alogicBox =
    new G4LogicalVolume(asolidBox, CubeMaterial, "LBlueCube", 0, 0, 0);

// the cube is blue
visatt = new G4VisAttributes( G4Colour(0.0, 0.0, 1.0) );
alogicBox->SetVisAttributes(visatt);
alogicBox->SetSensitiveDetector(sd);
// and is a sensitive detector of type MySD
MySD *sd = new MySD(SDname="/MySD");
G4SDManager* SDman = G4SDManager::GetSDMpointer();
SDman->AddNewDetector(sd);
alogicBox->SetSensitiveDetector(sd);
aphysiBox = new G4PVPlacement( ... )
```

To be implemented by the user

```
#include "G4VSensitiveDetector.hh"
#include "MySingleHit.hh"
class G4Step;
class G4HCofThisEvent;
class G4TouchableHistory;

class MySD : public G4VSensitiveDetector
{
    MySD(G4String name = "/MySD");
    void Initialize(G4HCofThisEvent *HCE);
    G4bool ProcessHits(G4Step *aStep, G4TouchableHistory *hist);
    void EndOfEvent(G4HCofThisEvent *HCE);
private:
    MyHitsCollection *myCollection;
};
```

ProcessHits method called
#include "G4VSensitiveDetector.hh"
#include "MySingleHit.hh"

class G4Step;

class G4HCofThisEvent;

class G4TouchableHistory;

class MySD : public G4VSensitiveDetector
{
    MySD(G4String name = "MySD");
    void Initialize(G4HCofThisEvent *HCE);
    G4bool ProcessHits(G4Step *aStep, G4TouchableHistory *hist);
    void EndOfEvent(G4HCofThisEvent *HCE);

private:
    MyHitsCollection *myCollection;
};
Sensitive detector, implementation

```cpp
#include "G4VSensitiveDetector.hh"
#include "MySingleHit.hh"

class G4Step;
class G4HCofThisEvent;
class G4TouchableHistory;

class MySD : public G4VSensitiveDetector
{
   MySD(G4String name = "/MySD");
   void Initialize(G4HCofThisEvent *HCE);
   G4bool ProcessHits(G4Step *aStep, G4TouchableHistory *hist);
   void EndOfEvent(G4HCofThisEvent *HCE);

private:
   MyHitsCollection *myCollection;
};

typedef G4THitsCollection<MyHit> MyHitsCollection;
extern G4Allocator<MyHit> MyHitAllocator;
inline void *MyHit::operator new(size_t)
{
   void *aHit = (void *) MyHitAllocator.MallocSingle(); return aHit;
}
inline void ParisSingleHit::operator delete(void *aHit)
{
   MyHitAllocator.FreeSingle((MyHit*) aHit);
}
```

G4 utilities for efficient memory management
MySD::MySD(G4String name): G4VSensitiveDetector(name)
{
    G4String HCname = "myHitCollection"; collectionName.insert(HCname);
}
void MySD::Initialize(G4HCofThisEvent* HCE)
{
    static int HCID = -1; myCollection = new MyHitsCollection(SensitiveDetectorName,collectionName[0]);
    if ( HCID < 0 )
        HCID = GetCollectionID(0);
    HCE->AddHitsCollection(HCID,myCollection);
}
G4bool ParisTrackerSD::ProcessHits(G4Step* aStep, G4TouchableHistory */*touch*/)
{
    // nothing to be stored if no energy
    G4double edep = aStep->GetTotalEnergyDeposit();
    if ( edep == 0. ) {
        return false;
    }
    // a new hit is created
    MySingleHit *newHit = new MySingleHit();
    newHit->Edep = edep;
    newHit->detID = aStep->GetTrack()->GetVolume()->GetCopyNo();
    // add this hit to the collection
    myCollection->insert( newHit );
    return true;
}
void MySD::EndOfEvent(G4HCofThisEvent*){};
Sensitive detector, implementation

MySD::MySD(G4String name): G4VSensitiveDetector(name)
{
    G4String HCname = "myHitCollection"; collectionName.insert(HCname);
}
void MySD::Initialize(G4HCofThisEvent* HCE)
{
    static int HCID = -1; myCollection = new MyHitsCollection(SensitiveDetectorName, collectionName[0]);
    if ( HCID < 0 )
        HCID = GetCollectionID(0);
    HCE->AddHitsCollection(HCID, myCollection);
}
G4bool ParisTrackerSD::ProcessHits(G4Step* aStep, G4TouchableHistory * touch)
{
    // nothing to be stored if no energy
    G4double edep = aStep->GetTotalEnergyDeposit();
    if ( edep == 0. ) {
        return false;
    }
    // a new hit is created
    MySingleHit *newHit = new MySingleHit();
    newHit->eDep = edep;
    newHit->detID = aStep->GetTrack()->GetVolume()->GetCopyNo();

    // add this hit to the collection
    myCollection->insert( newHit );
    return true;
}
void MySD::EndOfEvent(G4HCofThisEvent*){
}
void MyEventAction::EndOfEventAction(const G4Event *evt)
{
    MyHitsCollection *THC = NULL;
    // to get back my collection
    G4HCofThisEvent * HCE = evt->GetHCofThisEvent();
    if (HCE) {
        THC = (MyHitsCollection *)(HCE->GetHC(0));
    }
    if ( THC ) {
        int n_hit = THC->entries();
        for (int i = 0 ; i < n_hit; i++) {
            MyHit *hit= (*THC)[i];
            G4cout << hit->eDep << " " << hit->detID << G4cout;
        }
    }
}
Sensitive detectors provided

A general concrete sensitive detector **G4MultiFunctionalDetector** exists. It is a collection of **G4VPrimitiveScorer**

- a **G4VPrimitiveScorer** class accumulates one physics quantity for each physical volume. Ex:

  - Track length: `G4PSTrackLength`, `G4PSPassageTrackLength`
  - Deposited energy: `G4PSEnergyDeposit`, `G4PSDoseDeposit`
  - Current/Flux: `G4PSFlatSurfaceCurrent`, `G4PSSphereSurfaceCurrent`, `G4PSPassageCurrent`
  - Others: `G4PSMinKinEAtGeneration`, `G4PSNofSecondary`, `G4PSNofStep`

Commands are available:

```
/score/dumpQuantityToFile - to dump the result in a CSV (column separated values) file
```

Geant4 also introduces **G4VSDFilter** to filter what kind of tracks is to be considered by sensitivity. Ex, Accepts:

- only charged/neural tracks: `G4SDChargedFilter`, `G4SDNeutralFilter`
- tracks within the defined range of kinetic energy: `G4SDKineticEnergyFilter`
- tracks of registered particle types: `G4SDParticleFilter`
- tracks of registered particle types within defined range of kinetic: `G4SDParticleWithEnergyFilter`

* it inherits from **G4VSensitiveDetector**
The user's application

TODO List

• The files containing the definition of aHit and aTracker are provided see /group/formateurs/stezowski/utilities

• Integrate them to your application, set the detectors sensitives
• Run, check how the hit collection is filled and how hits are retrieved
Conclusions of W4

We have seen:

• how to extract information
  ➤ using user’s hooks
  ➤ using the detector sensitivity

• There are more advanced levels:
  ➤ notion of analysis manager: **G4VAnalysisManager**
  ➤ one can implement new **G4VAnalysisManager**
  ➤ one can implement new **G4VPrimitiveScorer**
  ➤ one can play with **G4VUserXXXInformation***
  ➤ ...

* XXX being Run, Event, etc ...
Conclusions of W4

We have seen:

• how to extract information
  ➤ using user’s hooks
  ➤ using the detector sensitivity

• There are more advanced levels:
  ➤ notion of analysis manager: G4VAnalysisManager
  ➤ one can implement new G4VAnalysisManager
  ➤ one can implement new G4VPrimitiveScorer
  ➤ one can play with G4VUserXXXInformation*
  ➤ ...

* XXX being Run, Event, etc ...