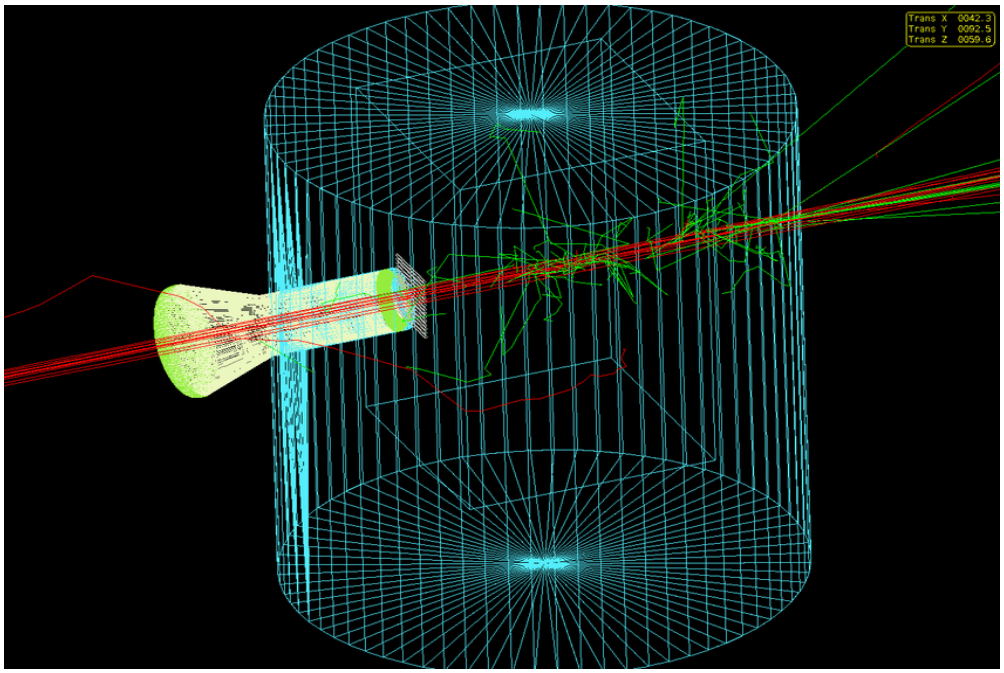
Université Ibn-Tofail
Faculté des Sciences - Kénitra
Département de Physique

Cours de la simulation Monte Carlo avec GEANT4



Une introduction pratique à La simulation Monte Carlo avec Geant4

Prof. Mohamed Gouighri

Master : Physique des Sciences et Techniques Nucléaires

Ce document est basé sur la documentation de l'école de Geant4 de 2005 présidé par le professeur **Sébastien Incerti**
de
IN2P3 / CNRS
Université Bordeaux 1
Centre d'Etudes Nucléaires de Bordeaux-Gradignan, France

Sommaire

Résumé

Ce manuscrit donne aux utilisateurs un bref aperçu de la boîte à outils Geant4 et leur propose de développer rapidement une application de simulation Geant4 de base, en supposant qu'ils aient déjà une connaissance raisonnable du C++, afin qu'ils puissent commencer à travailler sur leur propre application immédiatement après le Do. Tutoriel Geant4 de l'école des fils. Les diapositives présentées dans ce tutoriel présentent en détail un large panel de fonctionnalités Geant4 disponibles pour les utilisateurs dans tous les domaines d'application Geant4.

1. Introduction

La simulation joue un rôle fondamental dans divers domaines et phases d'un projet de physique expérimentale : conception du dispositif expérimental, évaluation et définition des résultats physiques potentiels du projet, évaluation des risques potentiels pour le projet, évaluation des performances du projet. Expérimentation, développement, test et optimisation de logiciels de reconstruction et d'analyse physique, contribution au calcul et à la validation de résultats de physique… La boîte à outils orientée objet Geant4 est un ensemble complet de bibliothèques écrites en C++ permettant à l'utilisateur de simuler son propre système de détection. En spécifiant la géométrie du détecteur, le système logiciel transporte automatiquement les particules projetées dans le détecteur en simulant les interactions des particules dans la matière sur la base de la méthode de Monte Carlo. Une telle méthode recherche des solutions à des problèmes mathématiques en utilisant un échantillonnage statistique avec des nombres aléatoires.

Geant4 a été initialement développé pour la simulation des détecteurs HEP de nouvelle génération (ATLAS, Alice, CMS, LHCb…), il est aujourd'hui largement utilisé pour la simulation des détecteurs de la génération actuelle ainsi que dans les communautés de physique spatiale et médicale. En principe, tout système expérimental basé sur des interactions de particules pourrait être simulé dans Geant4, à condition que les processus d'interaction correspondants aient été implémentés dans la boîte à outils. La boîte à outils est développée par une collaboration internationale de physiciens et d'ingénieurs logiciels (environ 100), collaborant tous ensemble dans un environnement de production et de gestion de logiciels distribués. Cela a débuté en 1993 au CERN sous la forme d'une phase de R&D (RD44) et la première version du logiciel a eu lieu en décembre 1998. Depuis, deux versions par an sont produites. Geant4 est entièrement ouvert, entièrement gratuit, régulièrement mis à jour et peut être installé sur les plateformes informatiques courantes (Linux, Windows©, Mac©,…). Il contient des exemples pédagogiques et un forum d'utilisateurs est disponible sur Internet à l'adresse : http://cern.ch/geant4, qui centralise toutes les informations concernant Geant4. Des tutoriels Géant4 courts, longs et spécialisés sont régulièrement organisés dans le monde entier. Geant4 est le successeur de Geant3, écrit en Fortran. Sa conception, son utilisation, sa maintenance et sa portabilité suivent des règles rigoureuses de programmation orientée objet.

Après une description de la structure de Geant4, nous décrirons les différentes étapes nécessaires au développement d'une application de base de Geant4. Des exemples de code sont fournis.

2. Présentation de Géant4

2.1. Structure générale de Geant4

Les particules sont générées dans Geant4 à partir d'un seul point ; leur trajectoire dans un matériau donné est calculée à partir d'une modélisation des processus physiques qui leur sont applicables. Chaque processus physique (par exemple l'ionisation des protons dans l'eau) est représenté à travers une classe C++, permettant le calcul de la probabilité d'interaction (libre parcours moyen) via ce processus ainsi que la génération de l'état final de la particule par ce processus. Chaque processus peut être décrit par plusieurs modèles complémentaires (comme l'ionisation paramétrée à partir des rapports de l'ICRU ou à partir du paramétrage de Ziegler) et une seule particule peut avoir différents processus (comme l'excitation des protons dans l'eau, l'ionisation des protons dans l'eau,…). Toutes les particules secondaires sont calculées de la même manière. Le suivi se produit jusqu'à ce que les particules

s'arrêtent ou quittent le volume de simulation. La plupart des grandeurs physiques (énergie, position, dépôt d'énergie…) sont accessibles à tout moment pendant la simulation et peuvent être extraites selon les besoins de l'utilisateur. De nombreux types de particules sont disponibles ainsi que des processus physiques, principalement classés en processus électromagnétiques et processus hadroniques. Les particules incluent un « geantino » qui n'est associé à aucun processus d'interaction et peut être utilisé pour vérifier une géométrie définie par l'utilisateur.

Geant4 se compose de 17 catégories de classes, illustrées à la figure 1 ; chacun est développé et maintenu indépendamment par un groupe de travail. Le noyau Geant4 est constitué de catégories en rouge. Il fournit les fonctionnalités centrales de la boîte à outils : gère les courses, les événements, les pistes, les étapes, les hits, les trajectoires, implémente Geant4 en tant que machine à états et fournit un cadre pour : les processus physiques, les pilotes de visualisation, les interfaces graphiques, la persistance, l'histogramme/analyse et le code utilisateur.

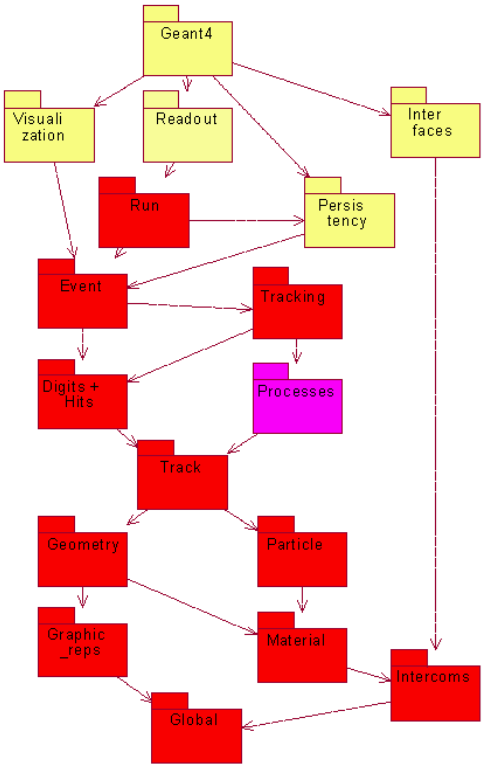


Fig. 1 : categories de classe

Les catégories au bas du diagramme sont utilisées par pratiquement toutes les catégories supérieures et constituent la base de la boîte à outils. La catégorie globale (**global**) couvre le système d'unités, de constantes, de valeurs numériques et de gestion des nombres aléatoires. Les deux catégories, matériaux **(material)** et particules **(particles)**, mettent en œuvre les installations nécessaires à la description des propriétés physiques des particules et des matériaux pour la simulation des interactions particules-matière. Le module de géométrie (**geometry**) offre la possibilité de décrire une structure géométrique et d'y propager efficacement des particules. Au-dessus de ces catégories se trouvent les catégories nécessaires pour décrire le suivi des particules et les processus physiques qu'elles subissent. La catégorie des pistes (**Track**) contient des classes pour les pistes et les étapes, utilisées par la catégorie des processus (**processes**), qui contient des implémentations de modèles d'interactions physiques : interactions électromagnétiques de leptons, photons, hadrons et ions, et interactions hadroniques. Tous les processus sont invoqués par la catégorie **tracking**, qui gère leur contribution à l'évolution de l'état d'une piste et fournit des informations en volumes sensibles pour les hits et la numérisation. Au-dessus de ceux-ci, la catégorie d'événements (**event**) gère les événements en fonction de leurs pistes et la catégorie d'exécution gère des collections d'événements qui partagent une implémentation commune de faisceau et de détecteur. Une catégorie de lecture (**readout**) permet la gestion des carambolages.

2.2. Unités de simulation

Plusieurs unités de simulation sont utilisées dans Geant4 et doivent être présentées à l'utilisateur avant de commencer à construire une application.

- **A run**

La plus grande unité de simulation dans Geant4 est une exécution. Elle est représentée par la classe G4Run. Une course est un ensemble d'événements qui se produisent dans des conditions identiques. Au cours d'une analyse, l'utilisateur ne peut pas modifier la géométrie du détecteur ou de l'appareil, ni les paramètres du processus physique. Par analogie avec la physique des hautes énergies, une exécution de Geant4 commence par la commande « **beamOn** ». Le détecteur est inaccessible une fois le faisceau allumé. Au début d'une analyse, la géométrie est optimisée pour la navigation, les sections transversales sont calculées en fonction des matériaux présents dans la configuration, les valeurs de coupure basse énergie sont définies.

- **An event**

Au début du traitement, un événement contient des particules primaires (provenant d'un générateur, d'un canon à particules, ...), qui sont poussées sur une pile. Pendant le traitement, chaque particule est extraite de la pile et suivie. Lorsque la pile est vide, le traitement de l'événement est terminé. La classe **G4Event** décrit un événement. En fin de traitement, il dispose des objets suivants : liste des sommets et particules primaires (l'entrée), collections de hits, collections de trajectoires (facultatif), collections de numérisation (facultatif).

- **A track**

Une trace est un instantané d'une particule dans son environnement au fur et à mesure que la particule se déplace. Les quantités dans l'instantané changent à chaque instant particulier, une piste a une position et des quantités physiques, ce n'est pas un ensemble d'étapes. Un objet trace (classe **G4Track**) a une durée de vie, il est créé par un générateur ou un processus physique (par exemple désintégration) et il est supprimé lorsqu'il quitte le volume mère Monde, disparaît (les particules se désintègrent ou sont absorbées), passe à une énergie nulle. Et aucun processus « au repos » n'est défini ou l'utilisateur le tue. Aucun objet de suivi ne survit à la fin d'un événement (non persistant). L'utilisateur doit prendre des mesures pour stocker les antécédents dans la trajectoire.

- **A step**

Le pas (classe **G4Step**) est l'unité de base de la simulation ; il comporte deux points (pré-étape, post-étape) « pre step, post step » – voir Fig. 2 – et il contient les informations incrémentielles sur les particules (perte d'énergie, temps écoulé, etc.). Chaque point contient des informations sur le volume et le matériau. Si l'étape est limitée par une frontière, le point final se situe exactement sur la frontière, mais fait logiquement partie du volume suivant. Ainsi, des processus limites tels que la réfraction et le rayonnement de transition peuvent être simulés.

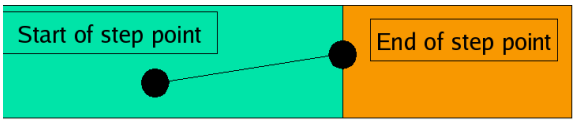


Fig. 2 : définition de l'étape “step”

Dans Geant4, chaque particule possède sa propre liste de processus applicables. Au début de chaque étape, tous ces processus sont interrogés pour connaître la durée d'interaction physique proposée. Le processus dont la durée proposée est la plus courte (en espace-temps) est celui qui se produit. Le processus choisi limite également la taille des étapes.

2.3. Unités internes de Geant4

Le système d'unités internes utilisé dans Geant4 est complètement caché du code utilisateur et de l'implémentation du code source de Geant4. Chaque numéro codé en dur doit être multiplié par son unité appropriée (par exemple. **radius=10.0*cm; kineticE=1.0*GeV;**). Pour récupérer un nombre, il faut le diviser par l'unité souhaitée : **G4cout << eDep / MeV**. Les unités les plus couramment utilisées sont fournies, mais l'utilisateur peut en ajouter de nouvelles. Avec ce système, l'importation/exportation de quantités physiques est simple et le code source est plus lisible.

2.4. Le préfixe de G4

Pour la portabilité "**G4**" est ajouté aux noms de types C++ bruts (**raw**), comme **G4int, G4double,**.... De cette façon, Geant4 implémente le type correct pour une architecture donnée. **G4cout** et **G4cerr** sont des objets ostream définis par Geant4 et **G4endl** est également fourni. Certaines interfaces utilisateur graphiques sont des flux de sortie tampons afin qu'elles affichent les impressions sur une autre fenêtre ou fournissent une fonctionnalité de stockage/édition. L'utilisateur ne doit pas utiliser **std::cout**, etc. Les utilisateurs ne doivent pas utiliser **std::cin** pour la saisie. Utilisez plutôt les commandes définies par l'utilisateur fournies par la catégorie des intercoms, par exemple, **G4UIcmdWithADouble**.

3. Installation de Geant4 et développement du code utilisateur

3.1. Installation de Geant4

La procédure d'installation de Geant4 est entièrement décrite sur le site web de Geant4.

En alternative, pour les utilisateurs qui ne souhaitent pas s'embêter avec l'installation, une configuration de virtualisation utile basée sur VMware© peut être téléchargée gratuitement sur **http://geant4.in2p3.fr**, dans la section Geant4 pour VMware©. Il contient une installation entièrement à jour et prête à l'emploi de Geant4 sous Scientific Linux 4.5 fonctionnant sous Windows© ou Mac OS© avec de nombreux outils utiles – voir Fig.3.

Fig. 3 : siteweb Geant4 pour WMware

3.2. Comment écrire votre propre code utilisateur

Les étapes menant au développement d'une application complète sont présentées ci-après.

L'utilisateur doit écrire un **main()** qui n'est pas fourni et doit utiliser des classes pour créer une application au-dessus de la boîte à outils Geant4 ; les cours habituels sont :

- Classes d'initialisation
  - **G4VUserDetectorConstruction**
  - **G4VUserPhysicsList**
- Classes d'action
  - **G4VUserPrimaryGeneratorAction**
  - **G4UserRunAction**
  - **G4UserEventAction**
  - **G4UserStackingAction**
  - **G4UserTrackingAction**
  - **G4UserSteppingAction**

Les noms de classes en **rouge gras** sont obligatoires.

3.2.1. main() - Obligatoire

Geant4 ne fournit pas de main(). Cependant, de nombreux exemples sont fournis dans le Guide des développeurs d'applications

Dans main(), vous devez construire un **G4RunManager** (ou une classe qui en dérive) et fournir à **G4RunManager** des pointeurs vers les classes d'utilisateurs obligatoires : **G4VUserDetectorConstruction, G4VUserPhysicsList** et **G4VUserPrimaryGeneratorAction**. Les classes de gestionnaire négocient les transactions entre les objets au sein d'une catégorie et communiquent avec d'autres gestionnaires. Ce sont des célibataires. L'utilisateur aura le plus de contacts avec **G4RunManager**. Il doit y enregistrer la géométrie du détecteur, la liste physique et le générateur de particules. Il existe d'autres classes de gestionnaires :

- **G4EventManager** – gère le traitement des événements, les actions des utilisateurs
- **G4TrackingManager** – gère les traces (tracks), le stockage de la trajectoire, les actions de l'utilisateur
- **G4SteppingManager** – gère les étapes (steps), les processus physiques, le score d'impact (hit scoring), les actions des utilisateurs
- **G4VisManager** – gère les pilotes de visualisation

D'autres classes facultatives peuvent être définies dans **main()** comme la session (G)UI pour définir une interface utilisateur (graphique).

3.2.2. Detector Construction - Obligatoire

L'utilisateur doit dériver sa propre classe concrète de **G4VUserDetectorConstruction**. Dans la méthode virtuelle **Construct()**, il faut assembler tous les matériaux nécessaires et construire les volumes de la géométrie du détecteur. En option, on peut construire des classes de détecteurs sensibles et les attribuer aux volumes du détecteur, définir des régions pour n'importe quelle partie du détecteur (pour les gammes de production), définir les attributs de visualisation des éléments du détecteur et des champs magnétiques (ou autres).

3.2.3. Physics : particles and processes - Obligatoire

Geant4 n'a pas de particules ou de processus par défaut, même le transport des particules doit être explicitement défini par l'utilisateur. L'utilisateur doit dériver sa propre classe concrète de la classe de base abstraite **G4VUserPhysicsList**, où il définit toutes les particules nécessaires, définit tous les processus nécessaires et les assigne aux particules appropriées et définit les plages de seuils de production (seuil) et les assigne au monde mère. volume et chaque région.

Geant4 fournit de nombreuses classes/méthodes utilitaires pour vous aider dans les tâches ci-dessus. Des exemples de listes de physique existent pour la physique électromagnétique (EM) et hadronique.

Les coupes sont souvent utilisées dans les applications Geant4. Une « coupure » dans Géant4 est un seuil de production ; cela ne s'applique qu'aux processus physiques qui ont une divergence infrarouge, ce n'est pas une coupe de suivi. Un seuil énergétique doit être déterminé à partir duquel une perte d'énergie discrète est remplacée par une perte d'énergie continue. Spécifiez la plage (qui est convertie en énergie pour chaque matériau) à laquelle commence la perte d'énergie continue, suivez les

Geant4 n'a pas de particules ou de processus par défaut, même le transport des particules doit être explicitement défini par l'utilisateur. L'utilisateur doit dériver sa propre classe concrète à partir du **G4VUserPhysicsList** classe de base abstraite, où il définit toutes les particules nécessaires, définit tous les processus nécessaires et les attribue aux particules appropriées et définit les plages de seuils de production (seuil) et les attribue au volume mère « World » et à chaque région.

Geant4 fournit de nombreuses classes/méthodes utilitaires pour vous aider dans les tâches ci-dessus. Des exemples de listes de physique existent pour la physique électromagnétique (EM) et hadronique.

Les coupes sont souvent utilisées dans les applications Geant4. Une « coupe » dans Géant4 est un seuil de production ; cela ne s'applique qu'aux processus physiques qui ont une divergence infrarouge, ce n'est pas une coupe de suivi. Un seuil énergétique doit être déterminé à partir duquel une perte d'énergie discrète est remplacée par une perte d'énergie continue. Spécifiez la plage (qui est convertie en énergie pour chaque matériau) à laquelle commence la perte d'énergie continue, suivez la plage primaire jusqu'à la plage zéro. Au-dessus de la plage spécifiée, cela crée des secondaires, en dessous de la plage, cela ajoute à une perte d'énergie continue du primaire.

3.2.4. Primary particles - Obligatoire

Pour chaque événement, l'utilisateur doit définir tous les détails de la particule initiale. Il doit tirer une classe concrète du **G4VUserPrimaryGeneratorAction** classe de base abstraite. Geant4 propose plusieurs manières de procéder : dérivez votre propre générateur à partir de **G4VPrimaryGenerator** ou utilisez les générateurs fournis:

- **G4ParticleGun** : l'utilisateur fournit le nombre, l'énergie, la direction et le type de particule
- **G4HEPEvtInterface**, **G4HepMCInterface** : interfaces avec les programmes de générateurs de haute énergie
- **G4GeneralParticleSource** : principalement pour la radioactivité

3.2.5. User actions - Facultatif

Plusieurs classes d'actions utilisateur facultatives avec des méthodes spécifiques peuvent être utilisées à des fins très diverses. En particulier, le **G4USerRunAction**, **G4UserEventAction** et **G4UserSteppingAction** les classes permettent à l'utilisateur d'accéder aux étapes utiles de la simulation.

G4UserRunAction

- **BeginOfRunAction** (définir des histogrammes)
- **EndOfRunAction** (remplir les histogrammes)

➢ **G4UserEventAction**

- **BeginOfEventAction (**sélection d'événements**)**
- **EndOfEventAction** (analyser l'événement)

➢ **G4UserTrackingAction**

- **PreUserTrackingAction** (créer une trajectoire définie par l'utilisateur)
- **PostUserTrackingAction**

➢ **G4UserSteppingAction**

- **UserSteppingAction** (tuer, suspendre, reporter la piste)
- **BeginOfSteppingAction**
- **EndOfSteppingAction**

➢ **G4UserStackingAction**

- **PrepareNewEvent** (réinitialiser le contrôle de priorité)
- **ClassifyNewTrack**
    - invoqué lorsqu'une nouvelle piste est poussée
    - peut définir une piste comme urgente, en attente, reportée ou tuée
- **NewStage**
    - invoqué lorsque la pile urgente est vide
    - filtrage des événements
    -

3.2.6. User interface - Facultatif

Geant4 fournit plusieurs classes concrètes **G4UISession** pour les fonctionnalités de l'interface utilisateur (par exemple, une interface de type terminal pour interagir avec le noyau Geant4). Vous pouvez sélectionner celui qui convient à votre environnement informatique. Dans **main()**, construisez-en un et invoquez sa méthode **sessionStart()**.

Les sessions d'interface utilisateur fournies sont :

User Interface sessions provided are :

- **G4UIterminal** : Terminal de caractères de type C et TC-shell, largement utilisé
- **G4GAG** : Tcl/Tk de l'interface graphique basée sur Java PVM
- **G4JAG** : interface vers JAS (Java Analysis Studio)
- **G4UIBatch** : batch job avec fichier macro

3.2.7. Visualization - Facultatif

Pour la visualisation de la configuration et des interactions simulées, l'utilisateur peut dériver sa propre classe concrète de **G4VVisManager** en fonction de son environnement informatique. Geant4 fournit des interfaces à plusieurs pilotes graphiques :

- DAWN – Fukui renderer
- WIRED – event display
- RayTracer – ray tracing par Geant4 tracking
- OpenGL – **le plus simple et le plus utilisé**
- OpenInventor
- VRML

4. Exemple de code utilisateur en huit étapes

Dans cette section, nous montrons une implémentation de fichiers sources pouvant être utilisés pour la construction d'une application de simulation standard présentée lors du tutoriel, en travaillant sur l'installation prête à l'emploi téléchargeable gratuitement de Geant4 (http://geant4.in2p3.fr). Cette application montre comment calculer les dépôts de dose par les protons incidents dans une cellule sphérique à eau liquide – voir Fig. 4. Les lignes de code sont commentées. Les fichiers d'en-tête correspondants sont également indiqués.

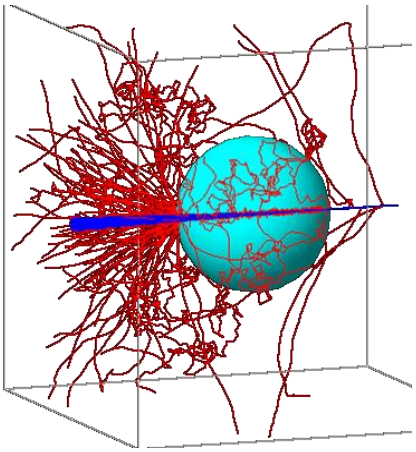


Fig. 4 : sortie graphique de l'application utilisateur proposée

4.1.1. ÉTAPE 1 : Édition et compilation de code

Habituellement, le code de simulation est stocké dans un répertoire de projet dédié (appelé « **simulation** » dans ce tutoriel) ; dans ce répertoire, deux sous-répertoires, « **include** » et « **src** », contiennent respectivement les fichiers d'en-tête du projet et les fichiers sources du projet – voir Fig. 5. Le main est placé dans le répertoire du projet dans le fichier **Simulation.cc**, ainsi que un fichier **GNUmakefile** qui sera utilisé lors de la compilation et du lien. Sous Linux, le projet est simplement compilé et lié à l'aide de la commande **gmake**

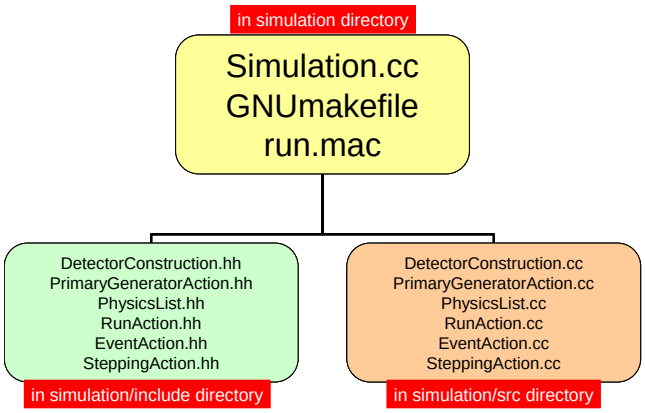


Fig. 5 : emplacement des fichiers dans l'application proposée

Exemple de fichier GNUmakefile pour la compilation et lien du projet Simulation ; à placer dans le répertoire « simulation »

```
# Add your project executable name
name := Simulation

G4TARGET := $(name)
G4EXLIB := true

ifndef G4INSTALL
  G4INSTALL = ../../..
endif

.PHONY: all
all: lib bin

include $(G4INSTALL)/config/binmake.gmk
```

Les fichiers du projet peuvent être modifiés avec n'importe quel éditeur de texte ou outil d'environnement de développement intégré. Dans ce tutoriel, nous utilisons l'outil gratuit snavigator© (disponible sur http://sourcenav.sourceforge.net/).

4.1.2. ÉTAPE 2 : Le main()

Exemple de fichier Simulation.cc à placer dans le répertoire « simulation » :

```
// Geant4 and user header files to include
#include "G4RunManager.hh"
#include "G4UImanager.hh"
#include "G4UIterminal.hh"
#include "G4UItcsh.hh"

#include "DetectorConstruction.hh"
#include "PhysicsList.hh"
#include "PrimaryGeneratorAction.hh"
#include "RunAction.hh"
#include "EventAction.hh"
#include "SteppingAction.hh"

// If one wishes to use vizualisation
#ifdef G4VIS_USE
  #include "G4VisExecutive.hh"
#endif

// Main called with optional arguments
int main(int argc,char** argv) {
```

```
// Construct the default run manager
G4RunManager * runManager = new G4RunManager;

// Set mandatory user initialization classes
DetectorConstruction* detector = new DetectorConstruction;
runManager->SetUserInitialization(detector);
runManager->SetUserInitialization(new PhysicsList);

// Set mandatory user action classes
runManager->SetUserAction(new PrimaryGeneratorAction(detector));
PrimaryGeneratorAction* primary = new PrimaryGeneratorAction(detector);

// Set optional user action classes
RunAction* RunAct = new RunAction(detector);
runManager->SetUserAction(RunAct);

runManager->SetUserAction(new EventAction(RunAct));

runManager->SetUserAction(new SteppingAction(RunAct,detector,primary));

// Visualization manager
#ifdef G4VIS_USE
  G4VisManager* visManager = new G4VisExecutive;
  visManager->Initialize();
#endif

// Initialize Geant4 kernel
runManager->Initialize();

   // Remove user output files
   system ("rm -rf dose.txt");

// Get the pointer to the User Interface manager
G4UImanager* UI = G4UImanager::GetUIpointer();

   if (argc==1)   // Define UI session for interactive mode.
   {
    G4UIsession * session = new G4UIterminal(new G4UItcsh);
    // Use of a command macro file
    UI->ApplyCommand("/control/execute /home/local1/simulation/run.mac");
    session->SessionStart();
    delete session;
   }

  else          // Batch mode
   {
    G4String command = "/control/execute ";
    G4String fileName = argv[1];
    UI->ApplyCommand(command+fileName);
   }

#ifdef G4VIS_USE
  delete visManager;
#endif

  delete runManager;

  return 0;
}
```

Il n'existe pas de fichier d'en-tête correspondant.

4.1.3. ÉTAPE 3 : définir une géométrie de détecteur

Dans l'exemple proposé, nous définissons le volume obligatoire du monde mère contenant de l'air et une sphère d'eau liquide. Ces matériaux sont définis ainsi que la géométrie. Chaque volume comporte trois descriptions obligatoires : solide, logique et physique. Les attributs de visualisation ainsi que la limitation de la taille des pas sont introduits.

Exemple de fichier DetectorConstruction.cc à placer dans le répertoire « simulation/src » :

```
#include "DetectorConstruction.hh"


//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

DetectorConstruction::DetectorConstruction()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

DetectorConstruction::~DetectorConstruction()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// Mandatory implementation of Construct method
G4VPhysicalVolume* DetectorConstruction::Construct()

{
  DefineMaterials();
  return ConstructDetector();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void DetectorConstruction::DefineMaterials()
{
  G4String name, symbol;
  G4double density;

  G4int ncomponents, natoms;
  G4double z, a;
  G4double fractionmass;

  // Define Elements
  // Example: G4Element* Notation  = new G4Element ("Element", "Notation", z, a);
  G4Element*   H  = new G4Element ("Hydrogen", "H", 1. ,  1.01*g/mole);
  G4Element*   N  = new G4Element ("Nitrogen", "N", 7., 14.01*g/mole);
  G4Element*   O  = new G4Element ("Oxygen"  , "O", 8. , 16.00*g/mole);

  // Define Material

  // Case 1: chemical molecule
  // Water
  density = 1.000*g/cm3;
  G4Material* H2O = new G4Material(name="H2O"  , density, ncomponents=2);
  H2O->AddElement(H, natoms=2);
  H2O->AddElement(O, natoms=1);

  // Case 2: mixture by fractional mass
  // Air
  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);

  // Vacuum standard definition...
  density = universe_mean_density;
  G4Material* vacuum = new G4Material(name="Vacuum", z=1., a=1.01*g/mole,
              density);

    // Display list of defined materials
    G4cout << G4endl << *(G4Material::GetMaterialTable()) << G4endl;

  // Default materials in setup
  defaultMaterial = Air;
  waterMaterial = H2O;

}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
G4VPhysicalVolume* DetectorConstruction::ConstructDetector()
{

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....
```

```
// MANDATORY MOTHER "WORLD" VOLUME

WorldSizeX  = 50*micrometer;
WorldSizeY  = 50*micrometer;
WorldSizeZ  = 50*micrometer;

  //---Solid
  solidWorld = new G4Box("World",                                    //its name
                         WorldSizeX/2,WorldSizeY/2,WorldSizeZ/2);  //its size


  //---Logic
  logicWorld = new G4LogicalVolume(solidWorld,          //its solid
                                   defaultMaterial,  //its material
                                   "World");              //its name


//---Physical
physiWorld = new G4PVPlacement(0,                          //no rotation
                               G4ThreeVector(), //at (0,0,0)
                   "World",              //its name
                   logicWorld,                  //its logical volume
                   NULL,                 //its mother  volume
                   false,                //no boolean operation
                   0);                          //copy number


// TARGET VOLUME

solidTarget = new G4Sphere("Target",              //its name
                           0,10*micrometer,                  //its minRadius and maxRadius
                           0,2*M_PI,          //its phiMin and deltaPhi
                           0,M_PI);          //its thetaMin and deltaTheta


logicTarget = new G4LogicalVolume(solidTarget,  //its solid
                                  waterMaterial,     //its material
                                  "Target");              //its name

physiTarget = new G4PVPlacement(0,                         //rotation
                                G4ThreeVector(0,0,0),// transl
                                "Target",                  //its name
                   logicTarget,          //its logical volume
                   physiWorld,                  //its mother  volume
                   false,                //no boolean operation
                   0);                          //copy number

  // Visualization attributes

 G4VisAttributes* worldVisAtt= new G4VisAttributes(G4Colour(1.0,1.0,1.0)); //White
 worldVisAtt->SetVisibility(true);
 logicWorld->SetVisAttributes(worldVisAtt);

 G4VisAttributes* targetVisAtt= new G4VisAttributes(G4Colour(0,1.0,1.0)); //Blue
 targetVisAtt->SetForceSolid(true);
 targetVisAtt->SetVisibility(true);

 logicTarget->SetVisAttributes(targetVisAtt);

  // User Limits on step size

 logicWorld->SetUserLimits(new G4UserLimits(1*micrometer));
 logicTarget->SetUserLimits(new G4UserLimits(1*micrometer));

 return physiWorld;
}
```

Exemple de fichier DetectorConstruction.hh à placer dans le répertoire « simulation/include » :

```
#ifndef DetectorConstruction_h
#define DetectorConstruction_h 1

#include "G4VUserDetectorConstruction.hh"
#include "G4VPhysicalVolume.hh"
#include "G4LogicalVolume.hh"
#include "G4Box.hh"
#include "G4Sphere.hh"
#include "G4Material.hh"

#include "G4PVPlacement.hh"
#include "G4UserLimits.hh"
#include "G4VisAttributes.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// Class deriving from the virtual G4VUserDetectorConstruction class
class DetectorConstruction : public G4VUserDetectorConstruction
{

public:

 DetectorConstruction();

 ~DetectorConstruction();

 G4VPhysicalVolume* Construct();

private:

 G4double          WorldSizeX;
 G4double          WorldSizeY;
 G4double          WorldSizeZ;

 G4VPhysicalVolume* physiWorld;
 G4LogicalVolume*   logicWorld;
 G4Box*             solidWorld;

 G4VPhysicalVolume* physiTarget;
 G4LogicalVolume*   logicTarget;
 G4Sphere*          solidTarget;

 G4Material*        defaultMaterial;
 G4Material*        waterMaterial;

 void DefineMaterials();

 G4VPhysicalVolume* ConstructDetector();
};


#endif
```

4.1.4. ÉTAPE 5 : définir la physique et les particules « Physics and particles »

Dans cette partie on explique comment définir des particules et leur associer des processus physiques. Des procédés électromagnétiques à faible énergie pour les gammas, les électrons, les positons et les ions légers sont introduits ainsi que des réductions de production.

Exemple de fichier PhysicsList.cc à placer dans le répertoire « simulation/src » :

```
#include "PhysicsList.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

PhysicsList::PhysicsList():  G4VUserPhysicsList()
{
    // Specify production cut for EM processes
    defaultCutValue = 1*nanometer;
  cutForGamma     = defaultCutValue;
```

```
  cutForElectron  = defaultCutValue;
  cutForPositron  = defaultCutValue;
  cutForProton    = defaultCutValue;

    // Specify verbosity level
    SetVerboseLevel(1);
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

PhysicsList::~PhysicsList()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// ***** Construction of particles

void PhysicsList::ConstructParticle()
{
  ConstructBosons();
  ConstructLeptons();
  ConstructBaryons();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructBosons()
{

  // gamma
  G4Gamma::GammaDefinition();

  // optical photon
  G4OpticalPhoton::OpticalPhotonDefinition();
}
 //....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructLeptons()
{
  // leptons
  G4Electron::ElectronDefinition();
  G4Positron::PositronDefinition();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructBaryons()
{
  //  baryons
  G4Proton::ProtonDefinition();
  G4AntiProton::AntiProtonDefinition();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// ***** Processses and particles

void PhysicsList::ConstructProcess()
{
  AddTransportation();
  ConstructEM();
  ConstructGeneral();
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

#include "G4MultipleScattering.hh"
#include "G4eIonisation.hh"
#include "G4eBremsstrahlung.hh"
#include "G4eplusAnnihilation.hh"

#include "G4LowEnergyPhotoElectric.hh"
#include "G4LowEnergyCompton.hh"
#include "G4LowEnergyGammaConversion.hh"
#include "G4LowEnergyRayleigh.hh"
```

```
#include "G4LowEnergyIonisation.hh"
#include "G4LowEnergyBremsstrahlung.hh"

#include "G4hLowEnergyIonisation.hh"
#include "G4StepLimiter.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructEM()
{
  theParticleIterator->reset();

  while( (*theParticleIterator)() ){

    G4ParticleDefinition* particle = theParticleIterator->value();
    G4ProcessManager* pmanager = particle->GetProcessManager();
    G4String particleName = particle->GetParticleName();


    if (particleName == "gamma") {

      pmanager->AddDiscreteProcess(new G4LowEnergyCompton);

      G4LowEnergyPhotoElectric * LePeprocess = new G4LowEnergyPhotoElectric();
      pmanager->AddDiscreteProcess(LePeprocess);

      pmanager->AddDiscreteProcess(new G4LowEnergyGammaConversion());

      pmanager->AddDiscreteProcess(new G4LowEnergyRayleigh());

      // Allow use of step size limitation specified in DetectorConstruction
      pmanager->AddProcess(new G4StepLimiter(),-1,-1,3);


    } else if (particleName == "e-") {

      pmanager->AddProcess(new G4MultipleScattering,-1, 1,1);

      G4LowEnergyIonisation * LeIoprocess = new G4LowEnergyIonisation("IONI");
      pmanager->AddProcess(LeIoprocess, -1,  2, 2);

      G4LowEnergyBremsstrahlung * LeBrprocess = new G4LowEnergyBremsstrahlung();
      pmanager->AddProcess(LeBrprocess, -1, -1, 3);

      // Allow use of step size limitation specified in DetectorConstruction
      pmanager->AddProcess(new G4StepLimiter(),-1,-1,3);

    } else if (particleName == "e+") {

      pmanager->AddProcess(new G4MultipleScattering,-1, 1,1);
      pmanager->AddProcess(new G4eIonisation,       -1, 2,2);
      pmanager->AddProcess(new G4eBremsstrahlung,   -1,-1,3);
      pmanager->AddProcess(new G4eplusAnnihilation,  0,-1,4);

      // Allow use of step size limitation specified in DetectorConstruction
      pmanager->AddProcess(new G4StepLimiter(),-1,-1,3);

    } else if ((!particle->IsShortLived()) &&
                 (particle->GetPDGCharge() != 0.0) &&
                 (particle->GetParticleName() != "chargedgeantino")) {

      pmanager->AddProcess(new G4MultipleScattering(),-1,1,1);

      G4hLowEnergyIonisation* hLowEnergyIonisation = new G4hLowEnergyIonisation();
      pmanager->AddProcess(hLowEnergyIonisation,-1,2,2);

      hLowEnergyIonisation->SetElectronicStoppingPowerModel(particle,"ICRU_R49p");

      // Allow use of step size limitation specified in DetectorConstruction
      pmanager->AddProcess(new G4StepLimiter(),-1,-1,3);

    }
  }
}
```

```
//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PhysicsList::ConstructGeneral()
{ }

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// ***** Set production cuts

void PhysicsList::SetCuts()
{
  if (verboseLevel >0)
  {
    G4cout << "PhysicsList::SetCuts:";
    G4cout << "CutLength : " << G4BestUnit(defaultCutValue,"Length") << G4endl;
  }

  // set cut values for gamma at first and for e- second and next for e+,
  // because some processes for e+/e- need cut values for gamma
  SetCutValue(cutForGamma, "gamma");
  SetCutValue(cutForElectron, "e-");
  SetCutValue(cutForPositron, "e+");

  // set cut values for proton and anti_proton before all other hadrons
  // because some processes for hadrons need cut values for proton/anti_proton
  SetCutValue(cutForProton, "proton");
  SetCutValue(cutForProton, "anti_proton");

  if (verboseLevel>0) DumpCutValuesTable();
}
```

Exemple de fichier PhysicsList.hh à placer dans le répertoire « simulation/include » :

```
#ifndef PhysicsList_h
#define PhysicsList_h 1

#include "G4VUserPhysicsList.hh"
#include "G4ProcessManager.hh"
#include "G4ParticleTypes.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

class PhysicsList: public G4VUserPhysicsList
{
public:
  PhysicsList();
  ~PhysicsList();

  void SetGammaCut(G4double);
  void SetElectronCut(G4double);
  void SetPositronCut(G4double);
  void SetProtonCut(G4double);

protected:
  // these methods construct particles
  void ConstructBosons();
  void ConstructLeptons();
  void ConstructBarions();

  // these methods construct physics processes and register them
  void ConstructGeneral();
  void ConstructEM();

  // Construct particle and physics
  void ConstructParticle();
  void ConstructProcess();

  // set cuts
  void SetCuts();

private:
  G4double cutForGamma;
```

```
  G4double cutForElectron;
  G4double cutForPositron;
  G4double cutForProton;

};
#endif
```

4.1.5. ÉTAPE 4 : génération de particules primaires « primary particles »

Cet exemple montre comment définir un simple canon « gun » à particules tirant des protons de 3 MeV juste avant la cible

Exemple de fichier PrimaryGeneratorAction.cc à placer dans le répertoire « simulation/src » :

```
#include "PrimaryGeneratorAction.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// Specify constructed detector in argument
PrimaryGeneratorAction::PrimaryGeneratorAction(DetectorConstruction* DC)
:Detector(DC)
{
    // Define particle gun object
  G4int n_particle = 1;
    particleGun  = new G4ParticleGun(n_particle);

}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

PrimaryGeneratorAction::~PrimaryGeneratorAction()
{
  delete particleGun;
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
    // Get simulation current event number
    G4int numEvent;
  numEvent=anEvent->GetEventID()+1;

  G4double x0,y0,z0,theta,phi,xMom0,yMom0,zMom0,e0;

    // Specify kinetic energy
  e0 = 3*MeV ;
    particleGun->SetParticleEnergy(e0);

    // Specify emission direction
  theta = 0;
  phi = 0;
  xMom0 = std::sin(theta);
  yMom0 = std::sin(phi);
  zMom0 = std::sqrt(1.-xMom0*xMom0-yMom0*yMom0);
    particleGun->SetParticleMomentumDirection(G4ThreeVector(xMom0,yMom0,zMom0));

    // Specify emission point
  x0 = 0;
  y0 = 0;
  z0 = -20*micrometer;
    particleGun->SetParticlePosition(G4ThreeVector(x0,y0,z0));

    // Select proton
    G4ParticleDefinition* particle=
   G4ParticleTable::GetParticleTable()->FindParticle("proton");

  particleGun->SetParticleDefinition(particle);

    // Example of output display
    G4cout
```

```
    << "-> Event= " << numEvent
    << " : Theta (mrad)= " << theta/mrad
    << " - Phi (mrad)= " << phi/mrad
    << " - x0 (um)= " << x0/um
    << " - y0 (um)= " << y0/um
    << " - z0 (um)= " << z0/um
    << " - e0 (MeV)= " << e0/MeV
    << G4endl;

    // Shoot
    particleGun->GeneratePrimaryVertex(anEvent);

}
```

Exemple de fichier PrimaryGeneratorAction.hh à placer dans le répertoire « simulation/include » :

```
#ifndef PrimaryGeneratorAction_h
#define PrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"
#include "G4ParticleGun.hh"
#include "DetectorConstruction.hh"
#include "G4Event.hh"
#include "G4ParticleTable.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
public:
  PrimaryGeneratorAction(DetectorConstruction*);
  ~PrimaryGeneratorAction();

  void GeneratePrimaries(G4Event*);

private:
  G4ParticleGun*          particleGun;
  DetectorConstruction*    Detector;
};

#endif
```

4.1.6. ÉTAPE 6 : collection de données

On peut utiliser des classes d'actions pour extraire des informations utiles de la simulation. Dans notre exemple, une analyse représenterait un ensemble de 10^3 protons envoyés vers la cible, un événement représenterait un proton parmi ces 10^3 protons, une étape représenterait toute étape d'interaction d'un proton ou de toutes les particules secondaires générées dans n'importe quelle partie de l'environnement simulé. installation.

Alors,

- nous calculons la masse cible à **Run Action**, c'est-à-dire au moment où la course commence
- la dose totale déposée par chaque proton incident ainsi que la position de sortie du faisceau de protons de la cible sont stockées dans un fichier texte pour une analyse ultérieure à la fin de l'action de l'événement « **Event Action** », c'est-à-dire après que chaque proton à tir unique a complètement interagi avec la configuration
- o la valeur de dose est calculée pour chaque proton et électron secondaire dans le milieu d'interaction à **Stepping Action**, ainsi que le transfert d'énergie linéaire et la propagation du faisceau à la sortie de la cellule

Commençons d'abord par la plus grande unité de la simulation : un run.

Exemple de fichier RunAction.cc à placer dans le répertoire « simulation/src » :

```
#include "RunAction.hh"
#include "G4Run.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// Specify constructed detector in argument
RunAction::RunAction(DetectorConstruction* det)
:Detector(det)
{
    // Initialize total deposited dose
    doseTarget=0;

    // Compute target mass
    G4double radius = 10*1e-6;
  G4double density = 1000;
  massTarget=(4/3)*M_PI*pow(radius,3)*density;
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

RunAction::~RunAction()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void RunAction::BeginOfRunAction(const G4Run* aRun)
{
   // Display run number
   G4cout << "---> Run " << (aRun->GetRunID())+1 << " start." << G4endl;
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void RunAction::EndOfRunAction(const G4Run*)
{
}
```

Le fichier d'en-tête contient des méthodes d'accès ; ces méthodes sont utiles lorsqu'il faut calculer des quantités qui doivent être mises à jour à différentes étapes de la simulation.

Exemple de fichier RunAction.hh à placer dans le répertoire « simulation/include » :

```
#ifndef RunAction_h
#define RunAction_h 1

#include "DetectorConstruction.hh"

#include "G4UserRunAction.hh"
#include "globals.hh"
#include <iostream>

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

class G4Run;

class RunAction : public G4UserRunAction
{
public:

  RunAction(DetectorConstruction*);
  ~RunAction();

  void BeginOfRunAction(const G4Run*);
  void EndOfRunAction(const G4Run*);

    // Accessor methods for dose computation
    void AddDoseTarget(G4double dose){ doseTarget += dose;}
  void SetDoseTarget(G4double dose){ doseTarget = dose;}
  G4double GetDoseTarget(){return doseTarget;}
```

```
    // Accessor methods for mass computation
  void SetMassTarget(G4double mT){ massTarget = mT;}
  G4double GetMassTarget(){return massTarget;}

private:

  DetectorConstruction* Detector;
  G4double doseTarget;
  G4double massTarget;

};

#endif
```

Passons maintenant à l'unité de simulation d'événements (un événement correspond à un tir de proton).

Exemple de fichier EventAction.cc à placer dans le répertoire « simulation/src » :

```
#include "EventAction.hh"
#include "RunAction.hh"

#include "G4Event.hh"
#include "G4EventManager.hh"
#include "G4TrajectoryContainer.hh"
#include "G4Trajectory.hh"
#include "G4VVisManager.hh"
#include "Randomize.hh"

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

// Specify run action object in argument
// Initialize a flag for trajectory drawing on visualization window
EventAction::EventAction(RunAction* run)
  :Run(run),drawFlag("all")
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

EventAction::~EventAction()
{}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void EventAction::BeginOfEventAction(const G4Event* /*evt*/)
{
// Dose is accessed through accessor and is set to zero at beginning of new event
  Run->SetDoseTarget(0);
}

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

void EventAction::EndOfEventAction(const G4Event* evt)
{

// Dose is written in text file at the end of event

  if (Run->GetDoseTarget()>0)
  {
              FILE* myFile;
              myFile=fopen("dose.txt","a");
              fprintf(myFile,"%e\n",float(Run->GetDoseTarget()) );
              fclose (myFile);

      G4cout << "   ===> The incident particle has reached the targeted cell :" << G4endl;
              G4cout << "   -----> total absorbed dose is (Gy) = " << Run->GetDoseTarget() << G4endl;
              G4cout << G4endl;
  }
  else
  {
              G4cout << "   ===> Sorry, the incident alpha particle has missed the targeted cell !"
            << G4endl;
              G4cout << G4endl;
```

```
  }

// Mandatory for trajectory drawing in vizualisation window

    if (G4VVisManager::GetConcreteInstance())
    {
     G4TrajectoryContainer * trajectoryContainer = evt->GetTrajectoryContainer();

     G4int n_trajectories = 0;
     if (trajectoryContainer) n_trajectories = trajectoryContainer->entries();

     for (G4int i=0; i<n_trajectories; i++)
       {
              G4Trajectory* trj = (G4Trajectory*)((*(evt->GetTrajectoryContainer()))[i]);
              if (drawFlag == "all")
                {
                  trj->DrawTrajectory(50);
                }
              else if ((drawFlag == "charged")&&(trj->GetCharge() != 0.))
                {
                  trj->DrawTrajectory(50);
                }
            }
    }
}
```

Exemple de fichier EventAction.hh à placer dans le répertoire « simulation/include » :

```
#ifndef EventAction_h
#define EventAction_h 1

#include "G4UserEventAction.hh"
#include "globals.hh"

class RunAction;

//....oooOO0OOooo........oooOO0OOooo........oooOO0OOooo........oooOO0OOooo....

class EventAction : public G4UserEventAction
{
 public:

   EventAction(RunAction*);
  ~EventAction();

   void BeginOfEventAction(const G4Event*);
   void   EndOfEventAction(const G4Event*);

 private:

   RunAction*             Run;
   G4String             drawFlag;
   G4int               printModulo;
};

#endif
```

Enfin, nous extrayons les quantités physiques au niveau de l'étape (étape d'interaction de toute particule générée dans la simulation).

Exemple de fichier SteppingAction.cc à placer dans le répertoire « simulation/src » :

```
#include "SteppingAction.hh"
#include "RunAction.hh"
#include "DetectorConstruction.hh"
#include "PrimaryGeneratorAction.hh"

#include "G4SteppingManager.hh"
#include "G4VTouchable.hh"
#include "G4VPhysicalVolume.hh"
```

```
//....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....

// RunAction, DetectorConstruction, PrimaryGeneratorAction objets in argument
SteppingAction::SteppingActio
(RunAction* run,DetectorConstruction* det,PrimaryGeneratorAction* pri)
:Run(run),Detector(det),Primary(pri)
{ }

//....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....

SteppingAction::~SteppingAction()
{ }

//....oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo.......oooOO0OOooo....

void SteppingAction::UserSteppingAction(const G4Step* s)

{

// Dose incrementation

 if (s->GetPreStepPoint()->GetPhysicalVolume()->GetName()  == "Target")
               {
                G4double dose = (e_SI*(s->GetTotalEnergyDeposit()/eV))/(Run->GetMassTarget());
                Run->AddDoseTarget(dose);
               }

// Beam spread after target

 if ( (s->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() == "proton")
   && (s->GetPreStepPoint()->GetPhysicalVolume()->GetName()  == "Target")
   && (s->GetPostStepPoint()->GetPhysicalVolume()->GetName() == "World")
    )
               {
               FILE* myFile;
               myFile=fopen("dose.txt","a");

               fprintf
                ( myFile,"%e %e %e ",
                 (s->GetTrack()->GetPosition().x())/micrometer,
                 (s->GetTrack()->GetPosition().y())/micrometer,
                 (s->GetTrack()->GetPosition().z())/micrometer
                );

               fclose (myFile);
               }

// Linear energy transfer computation

if ( (s->GetPreStepPoint()->GetPhysicalVolume()->GetName() == "Target")
 &&  (s->GetPostStepPoint()->GetPhysicalVolume()->GetName() == "World")
 &&  (s->GetTrack()->GetDynamicParticle()->GetDefinition() ->GetParticleName() == "proton")
   )
               {
                FILE *myFile;
                myFile=fopen("dose.txt","a");
                fprintf(myFile,"%e ",(s->GetTotalEnergyDeposit()/keV)/
        (s->GetStepLength()/micrometer));
                fclose (myFile);
               }

}
```

4.1.7. ÉTAPE 7 : example exécuter l'exemple

Une fois le code compilé avec **gmake**, il peut être exécuté avec la commande :

$G4WORKDIR/bin/Linux-g++/Simulation

Comme spécifié dans le code **Simulation.cc**, l'interface utilisateur lira les commandes intégrées données dans le fichier **run.mac** situé dans le répertoire « **simulation** ». Ce dossier précise notamment que 10^3 protons seront tirés vers la cible.

Exemple de fichier macro run.mac à placer dans le répertoire « simulation » :

```
# Visualization enabled
/vis/scene/create
/vis/scene/add/volume
/vis/sceneHandler/create OGLIX

/vis/viewer/create
/vis/viewer/set/viewpointThetaPhi 70 10

# Storing of trajectories
/tracking/storeTrajectory 1
/vis/scene/endOfEventAction accumulate

# Verbose control
/tracking/verbose 0
/run/verbose 0

# Shoot 1000 protons
/run/beamOn 1000
```

Une description de toutes les commandes est accessible en tapant help à l'invite de la session utilisateur Geant4. L'utilisateur peut définir ses propres commandes pour une meilleure interactivité avec son application.

4.1.8. ÉTAPE 8 : analyse de données

Geant4 n'approuve ni ne prend en charge des packages d'analyse particuliers. Cependant, une interface d'analyse abstraite est fournie : AIDA (Abstract Interfaces for Data Analysis). Il sera construit et lié à votre application si la variable d'environnement **$G4ANALYSIS_USE** est définie sur 1. Les en-têtes AIDA doivent être installés dans le code où l'analyse est configurée, avec **#include AIDA/AIDA.h**. L'utilisateur doit alors utiliser des outils d'analyse conformes à AIDA. Plus d'informations sur AIDA sont disponibles sur http://aida.freehep.org. Le package OpenScientist Lab est un package d'analyse conforme à la norme AIDA (http://www.lal.in2p3.fr/OpenScientist). Un exemple utilisant Open Scientist peut être trouvé dans **$G4INSTALL/examples/extended/analysis/AnaEx01**. Il permet notamment la production de fichiers résultats contenant des histogrammes, des tuples, …au format PAW (hbook) ou ROOT (root), largement utilisé dans la communauté de la physique des hautes énergies. Ce package est déjà installé sur l'installation téléchargeable VMware© de Geant4. Plusieurs exemples Geant4 (voir paragraphe suivant) illustrent comment utiliser ces bibliothèques, permettant la création de tels histogrammes directement dans Geant4..

Dans ce cours qui est considérer comme un tutoriel, nous créons simplement des fichiers texte de résultats de sortie directement dans l'application Geant4 et les analysons avec le package ROOT, en dehors de la simulation. À cette fin, nous utilisons le simple fichier macro ROOT **plot.C** pour tracer la distribution de dose, la distribution de transfert d'énergie linéaire à la sortie de la cible et la propagation du faisceau. Des ajustements de Gauss sont également appliqués. Cette macro lit le fichier texte **dose.txt** créé par l'application et placé dans le répertoire « **simulation** ». Le fichier **plot.C** se trouve dans le répertoire « **simulation** » et peut être édité avec n'importe quel éditeur de texte. Ouvrez une session ROOT dans la simulation « **simulation** » en tapant **root** et à l'invite et exécutez le fichier macro avec la commande **.X plot.C**. Des résultats typiques sont présentés sur la figure 6 pour 10^6 protons incidents de 3 MeV.

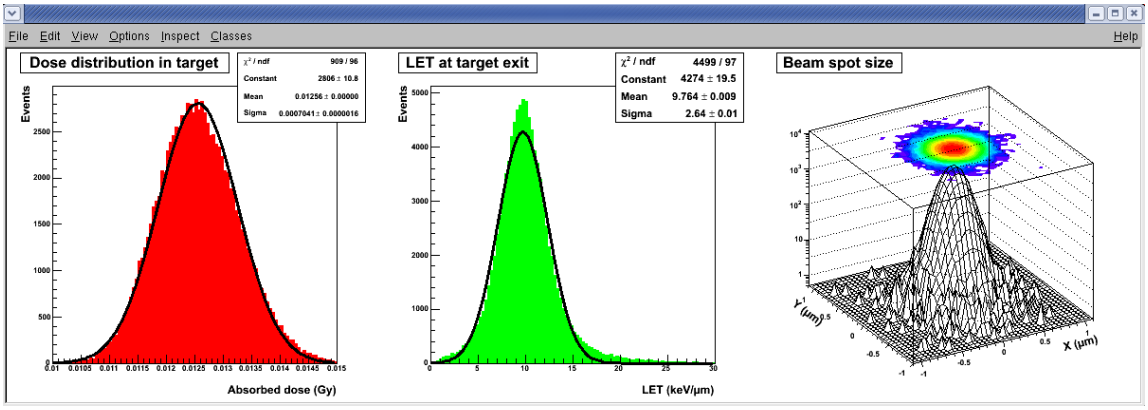


Fig. 6 : résultats d'application analysés avec ROOT

5. Documentation

Une documentation complète de Geant4 est disponible sur le site Web de Geant4 : http://cern.ch/geant4.

En particulier, une documentation complète est accessible depuis la section Support utilisateur (Fig. 7) de ce site Web, comprenant des instructions d'installation détaillées, un guide du développeur d'applications, un guide du développeur de la boîte à outils, un manuel de référence physique ainsi qu'un manuel de référence logiciel. Un forum interactif d'hyper actualités est disponible pour le partage d'informations et la pose de questions ainsi qu'un navigateur de code.

Sept exemples simples pour débutants, allant de très simple à complexe, sont disponibles dans le code source de Geant4 dans le répertoire **$G4INSTALL/examples**. Ils peuvent être utilisés comme modèles pour votre propre application. Des exemples étendus de tests et de validation, démontrant les outils Geant4 et étendant Geant4 sont également fournis. Les exemples avancés montrent des applications pratiques et des exemples extérieurs

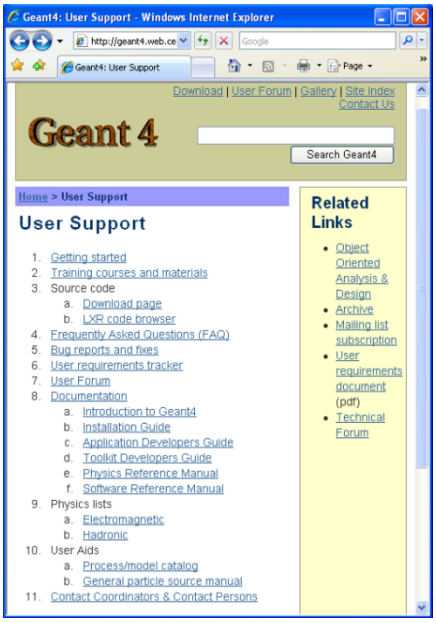


Fig. 7 : section d'assistance aux utilisateurs du site Web Geant4

Remerciements :

Ce document est basé sur l'école de Geant4 présidé le professeur **Sébastien Incerti** de

IN2P3 / CNRS
Université Bordeaux 1
Centre d'Etudes Nucléaires de Bordeaux-Gradignan, France

Publications de la collaboration Géant4

[1] S. Agostinelli *et al*., Geant4 - a simulation toolkit, Nucl. Instrum. Meth. A 506 (3) (2003) 250-303

[2] J. Allison *et al.*, Geant4 Developments and Applications, IEEE Trans. Nucl. Sci. 53 (1) (2006) 270-27