Środowisko programistyczne GEANT4

Leszek Adamczyk

Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo-Hutnicza

Wykłady w semestrze zimowym 2015/2016

Analiza danych

GEANT 4 nie ma rozbudowanego systemu analizy danych:

- Zbyt różnorodne środowisko użytkowników
- Każda grupa użytkowników ma swoje własne ulubione narzędzie do analizy (ROOT, EXCEL, ...)

Typowy wynik symulacji to: histogram lub tablica n-wymiarowa (ntuple) gdzie wierszem jest np. przypadek, cząstka lub hit a kolumnami wielkości takie jak: depozyt energii, skladowe pędu cząstki, pozycja hitu,...

GEANT umożliwia zapisanie histogramów i ntupli w wielu formatach (ROOT, XML, CSV, ...)

- Cały kod jest w GEANT4 zatem nie ma potrzeby instalacji dodatkowego oprogramowania do tworzenia plików wynikowych
- Programy zewnętrzne są potrzebne do analizy i wizualizacji wyników symulacji.

Analiza danych

System analizy danych zawiadamia G4AnalysisMenager.
Meneger zapenia:

- tworzenie plików wynikowych
- tworzenie i obsługę obiektów klasy histogram oraz ntuple
- jeden wspólny interfejs
  szczegóły specyficzne dla każdego formatu są dla użytkownika niewidoczne
- obsługe obiektów typu pliki, histogramy oraz ntuple
  dostęp w dowolnej partii kodu do plików, histogramów lub kolumn ntupli
- integracje ze środowiskiem GEANT 4
  dostęp do obiektów z poziomu interfejsu użytkownika
- pełną wielowątkowość
  sumowanie wyników po wszystkich wątkach

Histogramy

```
MyAnalysis.hh
.............
#include "g4root.hh"
//#include "g4xml.hh"
//#include "g4csv.hh"
```

```
MyRunAction.cc
--------------
#include "MyAnalysis.hh"
void MyRunAction::BeginOfRunAction(const G4Run* aRun)
// Create analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  analysisManager->SetVerboseLevel(1);
  analysisManager->SetFirstHistoId(1);

// Open an output file
  analysisManager->OpenFile("hist.root");

// Creating histograms
  G4int idPhotonFlux = analysisManager->CreateH1("PhotonFlux","photon flow",15,0.,90.);
-----------------------------------------------------
void MyRunAction::EndOfRunAction(const G4Run* aRun)
  // Get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
// Get histogram id not efficient in event loop
  G4int idPhotonFlux = analysisManager->GetH1Id("PhotonFlux");

  while( iter != theRun->frunHitsMap.GetMap()->end() ) {
// Fill histagrams as weighted events
  analysisManager->FillH1(idPhotonFlux, (iter->first+0.5)*90./15. , *iter->second);
}
// Save histograms and close file
  analysisManager->Write();
  analysisManager->CloseFile();
```

Histogramy

Identyfikatory

- Identyfikator histogramu tworzony automatycznie przy jego utworzeniu
  G4AnalysisManager::CreateH1();
- domyślny identyfikator pierwszego histogramu 0 mozna zmienić
  G4AnalysisManager::SetFirstHistoId(1);
- 1D, 2D, 3D histogramy numerowane niezaleznie:
  CreateH1(); CreateH2(), CreateH3();
- Nazwa histogramu nie ma nic wspólnego z jego identyfikatorem ale może mieć
  CreateH1("1","photon flow",15,0.,90.);

Oprócz jedno, dwu i trójwymiarowych histogramów można również tworzyć Profile dające możliwość wizulizacji wartości średniej i odchylenia standardowego wielkości z w funkcji wielkości x lub x, y
CreateP1(); CreateP2();

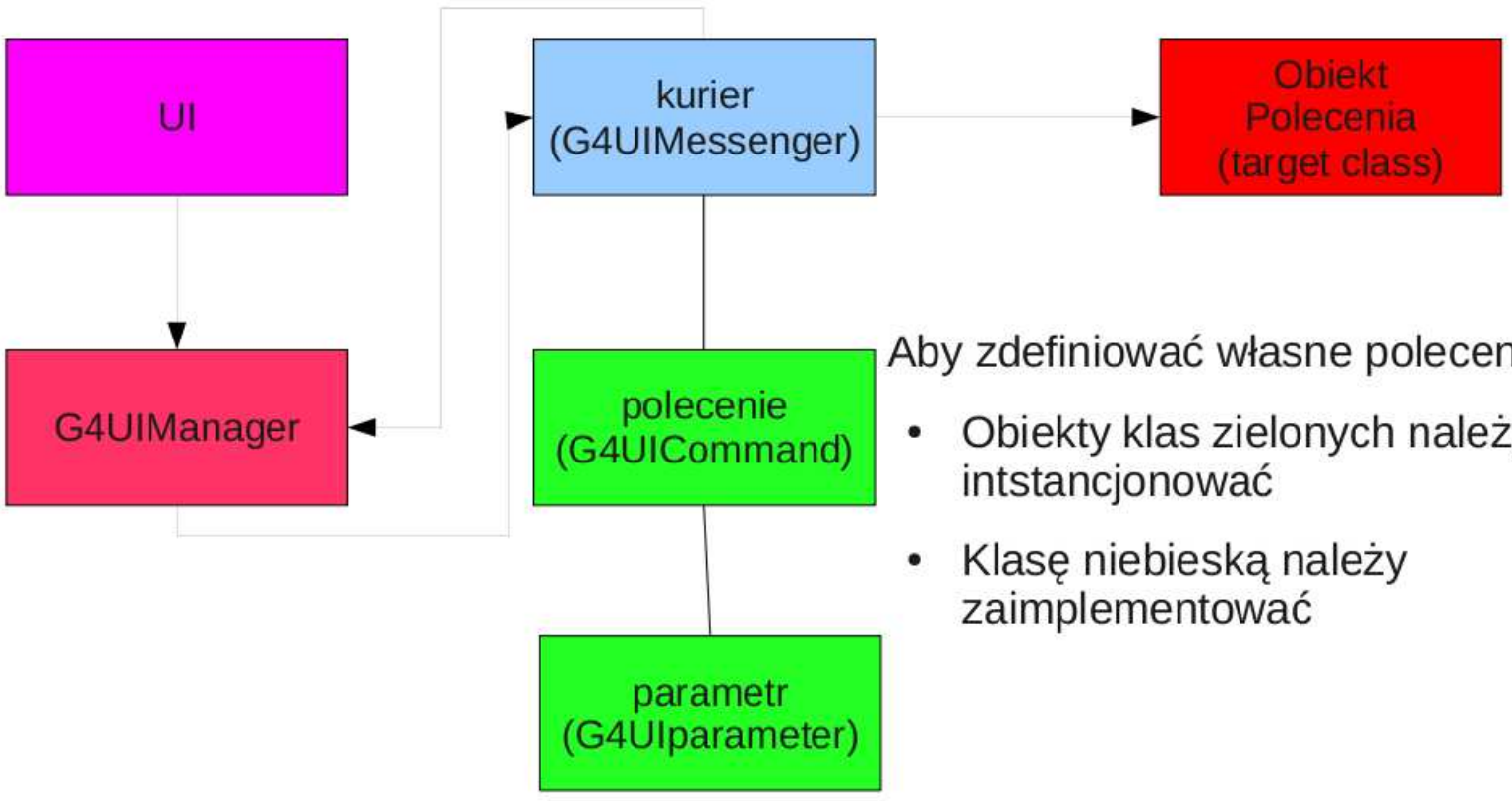
Ntuple: MyRunAction.cc

```
MyRunAction.cc
--------------
#include "MyAnalysis.hh"
void MyRunAction::BeginOfRunAction(const G4Run* aRun)
// Create analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();
  analysisManager->SetVerboseLevel(1);
  analysisManager->SetFirstNtupleId(1);

// Open an output file
  analysisManager->OpenFile("tree.root");

// Creating ntuple
  analysisManager->CreateNtuple("PhotonCollection","Kolekcja fotonów");
  analysisManager->CreateNtupleDColumn("E");
  analysisManager->CreateNtupleDColumn("X");
  analysisManager->CreateNtupleDColumn("Y");
  analysisManager->CreateNtupleDColumn("Z");
  analysisManager->FinishNtuple();
-----------------------------------------

void MyRunAction::EndOfRunAction(const G4Run* aRun)
// Get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

// Save ntuple and close file
  analysisManager->Write();
  analysisManager->CloseFile();
```

Ntuple: MyRun.cc

```
MyRun.cc
--------------
#include "MyAnalysis.hh"

void MyRun::RecordEvent(const G4Event* evt)
{
.....................................................................
  G4THitsCollection<MyPhotonHit>* photonCollection =
      dynamic_cast<G4THitsCollection<MyPhotonHit>*> (hce->GetHC(fCollectionI

// Get analysis manager
  G4AnalysisManager* analysisManager = G4AnalysisManager::Instance();

  for( G4int i = 0; i< photonCollection->entries(); i++) {
     float e = (*photonCollection)[i]->GetEnergy()/MeV;
     float x = (*photonCollection)[i]->GetPosition().x()/cm;
     float y = (*photonCollection)[i]->GetPosition().y()/cm;
     float z = (*photonCollection)[i]->GetPosition().z()/cm;

     analysisManager->FillNtupleDColumn(0,e);
     analysisManager->FillNtupleDColumn(1,x);
     analysisManager->FillNtupleDColumn(2,y);
     analysisManager->FillNtupleDColumn(3,z);
     analysisManager->AddNtupleRow();
  }
```

Polecenia interfejsu użytkownika



Aby zdefiniować własne polecenie:

- Obiekty klas zielonych należy intstancjonować
- Klasę niebieską należy zaimplementować

G4UImessenger

- Klasa bazowa G4UImessenger służy do wyprowadzania własnych klas definiujących polecenia UI.
- Obiekt takiej klasy powinien być utworzony (zniszczony) przez konstruktora (destruktora) obiektu na który działa dane polecenie (target class);
- Metody klasy G4UImessenger:
  - Constructor - definicja (+instancja) poleceń UI;
  - Destructor - unicestwienie poleceń UI;
  - SetNewValue(G4UICommand *, G4String NewValue) konwersja ”NewValue” na odpowiedni typ wartości oraz wywołanie odpowiedniej funkcji klasy target
  - GetCurrentValue(G4UICommand *) wywołuje odpowiednią funkcję klasy target i konwertuje wartość parametru na typ G4String

G4UImessenger- Przykład tworzenia polecenia

- Implementacja polecenia w konstruktorze klasy wyprowadzonej z G4UIMessenger.
- Przykład: Załóżmy, że chcemy z poziomu UI zmieniać grubość tarczy berylowej.
- Wyprowadzamy klasę MyDetectorMessenger z klasy bazowej G4UIMessenger

```
MyDetectorMessenger::MyDetectorMessenger
   (MyDetectorConstruction* detector) :fTarget(detector)

{   detDir = new G4UIdirectory("/My/");
    detDir->SetGuidance("My detector setup commands.");
    fBerCmd = new
      G4UIcmdWithADoubleAndUnit("/My/beryllium", this);
    fBelCmd->SetGuidance("Beryllium target.");
    fBerCmd->SetParameterName("depth",true);
    fBerCmd->SetRange("depth>0. && depth< 40.");
    fBerCmd->SetDefaultValue(15);
    fBerCmd->SetDefaultUnit("cm");  }
```

G4UICommand

- Klasa G4UICommand reprezentuje polecenia UI a G4UIParameter reprezentuje parametry poleceń.
- GEANT ma szereg gotowych klas wyprowadzonych z klasy G4UICommand w zależności od typu parametrów z nią stowarzyszonych:
  - G4UIcmdWithoutParameter
  - G4UIcmdWithAString
  - G4UIcmdWithABool
  - G4UIcmdWithAnInteger
  - G4UIcmdWithADouble, G4UIcmdWithADoubleAndUnit
  - G4UIcmdWith3Vector, G4UIcmdWith3VectorAndUnit
  - G4UIdirectory
- Polecenia z innymi typami parametrów należy samemu wyprowadzić z klasy bazowej G4UICommand

G4UICommand - Nazwa parametru

- Wszystkie klasy typu G4UIcmdWithAxxxx posiadające parametr mają zaimplementowaną metodę:

```
SetParameterName(
       const char* name,
     G4bool par2,
     G4bool par3=false)
```

- Nazwa parametru używana jest w helpie oraz w definicji zakresu parametrów;
- Jeśli `par2=true` to polecenie można wydać nie określając wartości parametru, użyta zostanie wartość domyślna;
- Jeśli `par3=true` to aktualna wartość parametru jest przyjęta jako domyślna, w przeciwnym przypadku wartość domyślną należy określić korzystając z SetDefaultValue

G4UICommand - zakres, jednostka parametru

- `void SetRange (const char*  range)`
  dostępna dla poleceń z parametrami numerycznymi.
  Warunki zapisujemy za pomocą składni c++
  `cmd->SetRange("x>0. && x>y && x+y<z")`
- `void SetDefaultUnit (const char* unit)`
  funkcja dostępna dla poleceń z określoną jednostką
  Po zdefiniowaniu jednostki polecenie nie zostanie zaakceptowane z jednostką o innym wymiarze.

G4UICommand - konwersje String-wartość numeryczna

Klasy wyprowadzone z klasy G4UICommand z numerycznymi lub logicznymi parametrami mają wbudowane funkcje do konwersji:

- String – wartość numeryczna:
  - `G4bool GetNewBoolValue (const char*)`
  - `G4int GetNewIntValue (const char*)`
  - `G4double GetNewDoubleValue(const char*)`
  - `G4ThreeVector GetNew3VectorValue(const char*)`
  - Należy ich używać w funkcji SetNewValue()
  - Jednostka uwzględniana jest automatycznie
- Wartość numeryczna - String
  - `G4String ConvertToString(.....)`
  - `G4String ConvertToString(....., const char* unit)`
  - Należy ich używać w funkcji GetCurrentValue()

G4UICommand - Przykład

```
void MyDetectorMessenger::SetNewValue
      (G4UIcommand* command, G4String newValue)
{
  if (command == fBerCmd)
    fTarget->SetBerylliumDepth
       (fBerCmd->GetNewDoubleValue(newValue));
}
```

```
G4String MyDetectorMessenger::GetCurrentValue
      (G4UIcommand* command)
{
  if (command == fBerCmd)
    return fBerCmd->ConvertToString
          (fTarget->GetBerylliumDepth(),"cm");
}
```

Błędy implementacji geometrii

- Obszar fizyczny nie jest całkowicie zawarty w obszarze w którym został umieszczony;
- Dwa obszary fizyczne umieszczone we wspólnym obszarze mają części wspólne;
- GEANT nie wykrywa tego typu błędów a jego działanie w ich obecności jest nieprzewidywalne;
- Istnieją narzędzia pomocne w wykrywaniu niepoprawnej implementacji geometrii:
  - opcjonalny test wykonywamne podczas konstrukcji obszaru fizycznego;
  - testy wykonywane za pomocą poleceń UI
  - narzędzie graficzne (DAVID)

Test podczas teorzenia obszaru fizycznego

- Konstruktory obiektów klas G4VPVPlacement oraz G4VPVParameterised posiadają opcjonalny argument sterujący wykonaniem testu:

```
G4PVPlacement( G4RotationMatrix*  Rot,
        const G4ThreeVector&      trans,
                        G4LogicalVolume*   CurrentLogical,
               const G4String&             Name,
                        G4LogicalVolume*   MotherLogical,
                        G4bool             Many,
                        G4int              CopyNo,
                        G4bool             SurfChk=false )
```

- Jeśli ostatni argument ma wartość logiczną true to podczas tworzenia obszaru fizycznego wykonywany jest test polegający na losowej generacji punktów na powierzchni obszaru i sprawdzeniu czy leżą one wewnątrz obszaru matki i nie należą do innych obszarów fizycznych;
- Test zajmuje bardzo dużo czasu.

Testy z poziomu UI

- Polecenia UI służące do weryfikacji geometrii znajdują się w katalogu `/geometry/test`
- Test polega na zdefiniowaniu krzywej (prosta lub okrąg) wzdłuż której GEANT transportuje wirtualną cząstkę i informuje o błędzie gdy:
  - cząstka wchodzi do jednego obszaru przed opuszczeniem poprzedniego a oba obszary są umieszczone we wspólnym obszarze;
  - cząstka poruszająca się w jakimś obszarze opuszcza jej obszar ”matkę”
- Możliwe jest zdefiniowanie całej sieci krzywych wzdłuż których przeprowadzany jest test.

Zewnętrzne narzędzie graficzne - DAVID

- DAVID jest narzędziem graficznym wykorzystującym opis geometrii detektora zawarty w pliku zapisanym przez interfejs do wizualizacji w formacie DAWNFILE;
- Należy mieć zainstalowany program DAWN. Oba programy: DAVID i DAWN można ściągnąć z `http:/geant4.kek.jp/~tanaka`
- DAVID wizualizuje na czerwono przecinające się obszary detektora (zapisuje również raport tekstowy)