Środowisko programistyczne GEANT4

Leszek Adamczyk

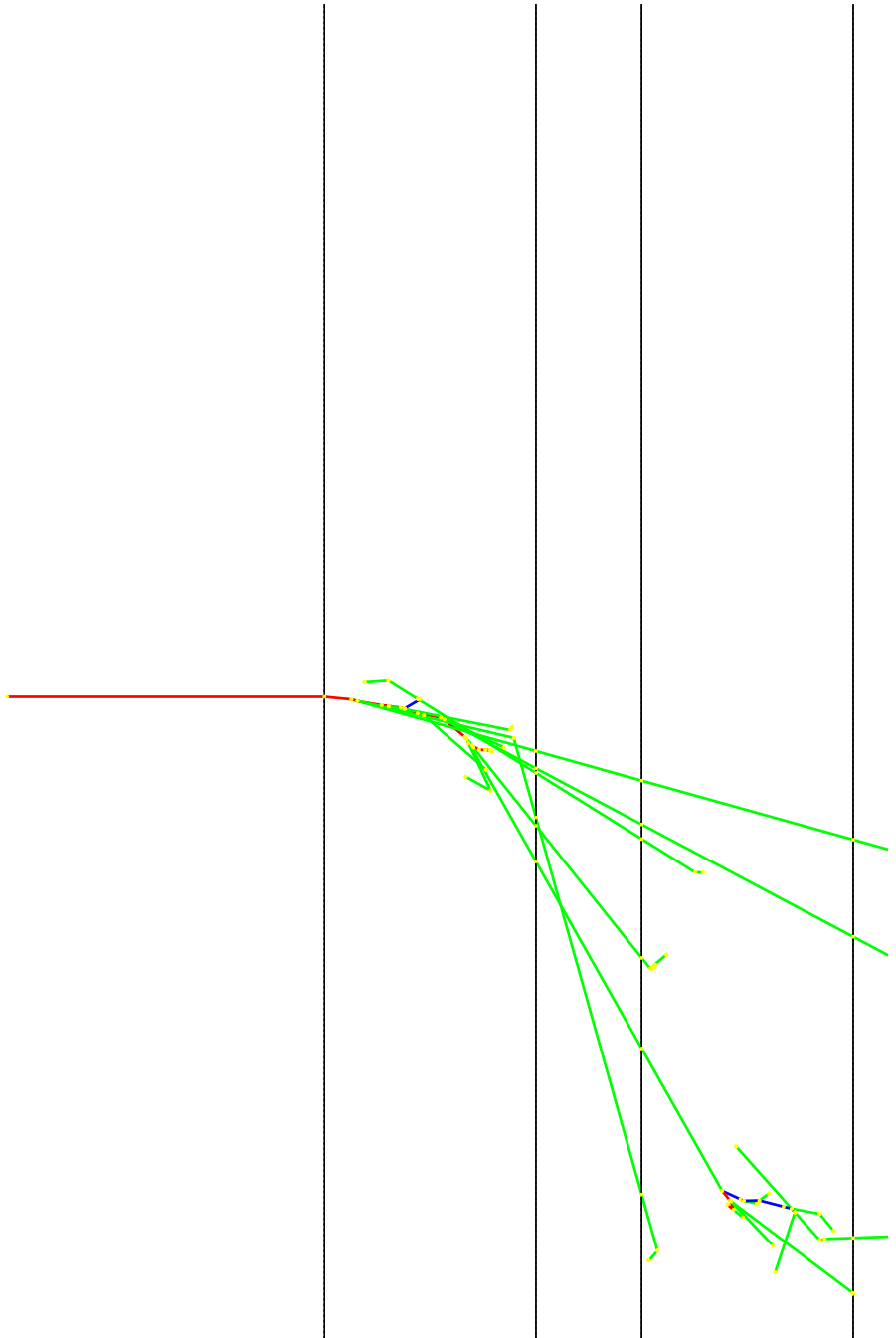
Wydział Fizyki i Informatyki Stosowanej
Akademia Górniczo-Hutnicza

Wykłady w semestrze zimowym 2015/2016

GEANT4: Symulacje przejścia cząstek przez materię

- proces przejścia cząstek przez materię jest zbyt złożony aby dało się jego wynik przewidzieć analitycznie;
- GEANT4 symuluje taki proces metodą Monte Carlo;
- z zadanych rozkładów prawdopodobieństwa losowane są wszystkie charakterystyki zderzeń elementarnych:
  - odległości pomiędzy zderzeniami;
  - rodzaju oddziaływania;
  - liczby cząstek wtórnych;
  - ich rodzaju, kąta emisji, energii, ...
- domyślnie symulowany jest los każdej z cząstek wtórnych;
- po wyczerpaniu cząstek otrzymujemy pełny opis ”sztucznie generowanego” procesu;

GEANT4: przejście elektronu przez materię



- przejście 50 *MeV* elektronu przez trzy warstwy materii;
- wtórnie powstają pozytony (kreacja pary elektron-pozyton) oraz fotony (promieniowania hamowania, anihilacja pozytonu).

GEANT4: Jądro

Jądro GEANT4 zapewnia:

- transport cząstek przez materię uwzględniając ich oddziaływanie z materiałem ośrodka oraz wpływ statycznych pól elektromagnetycznych aż do ich:
  - zatrzymania;
  - dezintegracji (rozpad lub absorpcje);
  - opuszczenia symulowanego obszaru;
- dostęp do informacji na temat procesu transportu oraz pełnych wyników symulacji:
  - na początku i końcu transportu cząstki;
  - na końcu każdego elementarnego kroku;
  - w momencie gdy cząstka wchodzi w interesujący nas obszar przestrzeni (np. obszar aktywny detektora, konkretny organ w ciele człowieka, tranzystor w układzie sterowania satelitą)

GEANT4: Twórca aplikacji

- użytkownik musi dostarczyć informacji o:
  - kształcie i składzie symulowanego ośrodka materialnego (Detector);
  - symulowanych procesach oddziaływania;
  - kinematyce cząstek pierwotnych.
- użytkownik może dostarczyć dodatkowych informacji:
  - o zewnętrznych polach elektromagnetycznych;
  - o działaniach jakie chcemy podjąć na końcu każdego elementarnego kroku podczas transportu lub w momencie gdy cząstka wchodzi w interesujący nas obszar detektora (metody UserAction).

GEANT4 : Event (przypadek)

- Event jest podstawową jednostką symulacji;
- na początku pierwotne cząstki umieszczane są na stosie;
- następnie cząstki jedna po drugiej pobierane są ze stosu i transportowane przez detektor;
- ewentualne cząstki wtórne odkładane są na stos;
- procesowanie przypadku kończy się w momencie opróżnienia stosu;
- przypadek reprezentowany jest przez klase G4Event składającą się się z następujących pól:
  - dane wejściowe: pierwotne cząstki;
  - dane wyjściowe: depozyty energii w obszarach aktywnych detektora oraz trajektorii cząstek;
- Procesowaniem przypadku kieruje klasa G4EventManager.

GEANT4 : Run

- Koncepcyjnie Run to zbiór Event'ów procesowanych przez ten sam detektor w identycznych warunkach fizycznych;
- w czasie Run'u użytkownik nie może zmienić:
  - detektora;
  - symulowanych procesów fizycznych;

  może zmienić kinematykę i rodzaj cząstek pierwotnych;
- Run reprezentowany jest przez klasę G4Run;
- procesowaniem Run'u steruje klasa G4RunManager.

GEANT4 : Track (Ślad)

- Track reprezentuje aktualny stan aktualnie transportowanej cząstki;
- Track nie ”pamięta” stanu cząstki z poprzedniego kroku transportu;
- Track nie jest ciągiem kroków, raczej Track jest aktualizowany po każdym kroku;
- do ”zapamiętania” kolejnych śladów cząstki używamy obiektu zwanego trajektorią;
- ślad reprezentowany jest przez klasę G4Track;
- procesowaniem śladu steruje klasa G4TrackingManager.

GEANT4 : Step (Krok)

- Krok składa się z punktu początkowego (PreStepPoint) oraz końcowego (PostStepPoint);
- oba punkty posiadają informacje o obszarze detektora (i materiale) w którym się znajdują, czy jest to pierwszy(ostatni) krok w obecnym obszarze detektora;
- w przypadku gdy krok ograniczony jest przez granicę ośrodków to punkt końcowy znajduje się fizycznie na granicy obszarów ale logicznie należy do następnego obszaru;
- krok posiada informacje o zmianie stanu cząstki podczas wykonywania kroku (strata energii, czas trwania kroku, ...)
- krok reprezentowany jest przez klase G4Step;
- klasa G4SteppingManager steruje procesowaniem kroku

GEANT4 : Trajectory (Trajektoria)

- W trakcie procesowania przypadku dostępna jest informacja tylko o aktualnie procesowanym śladzie oraz aktualnym kroku;
- okrojona informacja o historii śladów i kroków kopiowana jest dla każdej cząstki do obiektu zwanego trajektorią;
- klasa G4Trajectory zawiera okrojoną informacji o śladach;
- klasa G4TrajectoryPoint zawiera okrojoną informacji o krokach;
- w przypadku gdy klasy G4Trajectory lub G4TrajectoryPoint zawierają zbyt skąpe informacje, użytkownik może wyprowadzić własne klasy z klas bazowych G4VTrajectory oraz G4VTrajectoryPoint.

GEANT4 : Particle (Cząstki)

Cząstki w GEANT4 reprezentowane są przez trzy klasy:

- G4ParticleDefinition
  - określa ”statyczne” wielkości (ładunek, masa, czas życia, kanały rozpadu, ...);
- G4DynamicParticle
  - określa ”dynamiczne” wielkości fizyczne (pęd, energie, polaryzacje,...);
  - wszystkie obiekty klasy G4DynamicParticle tego samego typu stowarzyszone są z jednym obiektem klasy G4ParticleDefinition;
- G4Track
  - informacje geometryczne (pozycja, obszar detektora, materiał , straty energii, ...)
  - każdy G4Track stowarzyszony jest ze swoim własnym unikalnym obiektem klasy G4DynamicParticle

GEANT4 : Procesy fizyczne

- każdy typ cząstki ma własną listę procesów oddziaływania z materią;
- podczas każdego kroku wszystkie procesy z listy mają wpływ na:
  - wybór długości kroku;
  - zmianę wielkości fizycznych cząstki;
  - produkcję cząstek wtórnych;
  - zmianę stanu śladu.
- każdy proces charakteryzuje się jedną lub kombinacją kilku natur:
  - AtRest
    np. rozpad cząstki spoczywającej;
  - AlongStep (proces ciągły)
    np. promieniowanie Czerenkowa;
  - PostStep (proces dyskretny)
    np. rozpad cząstki w locie.

GEANT4: Algorytm wykonywania kroków

- dla każdego procesu dyskretnego losuje się odległość do następnego oddziaływania;
- najmniejszą z tych odległości wybiera się jako krok fizyczny;
- następnie oblicza się krok geometryczny jako odległość do granicy ośrodków;
- mniejszy z kroków fizycznego i geometrycznego decyduje o aktualnej długości kroku;
- zmiana energii kinetycznej w trakcie wykonywania kroku jest sumą wkładów od wszystkich procesów ciągłych;
- jeśli cząstka nie została zatrzymana przez procesy ciągłe to symuluje się proces dyskretny, który zdecydował o długości kroku.

GEANT4: Wyniki symulacji

- GEANT4 nie wie jakie informacje są nam potrzebne;
- użytkownik może mieć dostęp do interesujących go informacji na dwa sposoby:
  - korzystając z klas typu UserAction G4UserTrackingAction, G4UserSteppingAction,...
    - mamy dostęp do pełnej informacji po zakończeniu procesowania kazdego śladu, kroku, ...;
    - musimy sami przechować tę informację;
  - korzystając z funcjonalności GEANT4 zwanej obszarem aktywnym detektora (SensitiveDetector):
    - interesującemu nas obszarowi detektora przypisujemy obiekt klasy G4VSensitiveDetector;
    - GEANT4 automatycznie przechowuje wszystkie kroki wykonane w obszarze aktywnym detektora (Hits)
    - użytkownik ma do nich dostęp po zakończeniu procesowania przypadku korzystając z klasy G4UserEventAction

GEANT4: Układ jednostek

- GEANT4 nie ma domyślnych jednostek;
- aby nadać zmiennej wartość numeryczną, należy ją mnożyć przez jednostkę w której jest wyrażona np:

```
G4double length = 10.0*cm;
G4double energy = 10.0*GeV;
```

- prawie wszystkie powszechnie używane jednostki są zdefiniowane;
- można definiować własne jednostki;
- lista zdefiniowanych jednostek znajduje się w pliku nagłówkowym CLHEP:SystemOfUnits.h;
- aby uzyskać wartość:

```
G4cout << energy/MeV << "[MeV]" <<  G4endl;
G4cout << G4BestUnit(energy,"Energy") << G4endl;
```

GEANT4: Budowa aplikacji

Do komunikacji użytkownka z jądrem GEANT4 służą klasy bazowe (G4) :

- użytkownik tworzy klasy pochodne wyprowadzając je z klas bazowych;
- jądro GEANT4 operuje obiektami z klas pochodnych poprzez ich interfejs publiczny dziedziczony z klas bazowych;
- abstrakcyjne klasy bazowe (G4V) wymagają od użytkownika wyprowadzenia klas pochodnych oraz implementację metod czysto wirtualnych;
- nieabstrakcyjne klasy bazowe nie wymagają wyprowadzania klas pochodnych, ale ich wyprowadzenie umożliwia zmianę metod wirtualnych.

GEANT4: Klasy użytkownika

- main()
  GEANT4 jest środowiskiem, wymaga napisania programu głównego main();
- klasy inicjalizujące:
  - G4VUserDetectorConstruction
  - G4VUserPhysicsList
- klasy operacyjne
  - G4VUserPrimaryGeneratorAction
  - G4UserRunAction
  - G4UserEventAction
  - G4UserStackingAction
  - G4UserTrackingAction
  - G4UserSteppingAction

Klasy niebieskie sa opcjonalne, czerwone konieczne.

GEANT4: Katalog główny

- Przykład warsztaty - 1
- zawartość katalogu głównego:

```
GNUmakefile  include  My.cc  src

./include:
MyDetectorConstruction.hh  MyPhysicsList.hh  MyPrimaryGeneratorAction.hh

./src:
MyDetectorConstruction.cc  MyPhysicsList.cc  MyPrimaryGeneratorAction.cc
```

- nazwy plików są jednocześnie nazwami klas które deklarują lub definiują;
- nazwy klas pochodnych zawierają nazwy klas bazowych poprzedzone wspólnym przedrostkiem charakterystycznym dla danego projektu.

GEANT4: Program główny

Program główny musi zawierać:

- utworzenie obiektu klasy G4RunManager

```
G4RunManager * runManager = new G4RunManager;
```

- utworzenie i rejestracje do RunManager'a obiektów obligatoryjnych klas:

  G4VUserDetectorConstruction
  G4VUserPhysicsList
  G4VUserPrimaryGeneratorAction

```
runManager->SetUserInitialization(new MyDetectorConstruction);
runManager->SetUserInitialization(new MyPhysicsList);
runManager->SetUserAction(new MyPrimaryGeneratorAction());
```

- inicjalizacja jądra GEANT4

```
runManager->Initialize();
```

- wykonanie petli po przypadkach

```
runManager->BeamOn(numberOfEvents);
```

Uwaga: GEANT4 w wersji 10 jest w pełni wielowątkowy
G4MTRunManager

GEANT4: Opis detektora

- należy wyprowadzić z abstrakcyjnej klasy G4VUserDetectorConstruction własną klase pochodną;
- zaimplementować wirtualną funkcje Construct()
  - definicje potrzebnych materiałów
  - definicje brył geometrycznych
  - opis umiejscowienia detektora
  - definicje obszarów aktywnych detektora
  - definicje pól elektro-magnetycznych
  - definicje sposobu wizualizacji detektora

GEANT4: Procesy fizyczne

- należy wyprowadzić z abstrakcyjnej klasy G4VUserPhysicsList własną klase pochodną ;
- zaimplementować metody wirtualne:
  - ConstructParticles()
    definicje wszystkich potrzebnych cząstek
  - ConstructProcesses()
    definicje wszystkich wymaganych procesów oddziaływania cząstek
  - SetCuts()
    określenie warunków na produkcję cząstek wtórnych (w postaci zasięgu)

GEANT4: Cząstki pierwotne

- należy wyprowadzić z abstrakcyjnej klasy G4VUserPrimaryGeneratorAction własną klase pochodną
- zdefiniować cząstki pierwotne (rodzaj, pozycje, kierunek, energie).
  Można wyprowadzić klase pochodną z abstrakcyjnej klasy bazowej G4VPrimaryParticle lub skorzystać z gotowych klas:
  - G4PartcleGun
    każdy przypadek to identyczna cząstka
  - G4HEPEvtInterface
    interfejs do standardowego formatu HEPEVT
  - G4GeneralParticleSource
    modeluje źródła promieniowania o dowolnym widmie energetycznym, rozkładzie kątowym i przestrzennym
- zaimplementować wirtualną funkcje GeneratePrimeries()

GEANT4: GNUmakefile

- Plik reguł dla programu make, służącego do automatyzacji kompilacji programów:

```
name := My
G4TARGET := $(name)
G4EXLIB := true
G4WORKDIR :=.

.PHONY: all
all:  lib bin

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

GEANT4: Opcjonalne klasy użytkownika I

Użytkownik może przedefiniować wirtualne funkcje klas bazowych aby zwiększyć kontrole lub dostęp do informacji na różnych etapach symulacji.

- G4UserRunAction
  - BeginOfRunAction
    - np. utworzenie histogramów
  - EndOfRunAction
    - np. zapis histogramów do pliku
- G4UserEventAction
  - BeginOfEventAction
    - np. selekcja przypadków
  - EndOfEventAction
    - np. analiza przypadku, zapis informacji

GEANT4: Opcjonalne klasy użytkownika II

- G4UserStackingAction
  - ClassifyNewTrack
    - wywoływana dla każdego nowego śladu
    - dokonuje klasyfikacji śladu do różnych stosów:
    - Urgent, Waiting, PostponeToNextEvent
  - NewStage
    - wywoływana po opróżnieniu stosu Urgent
    - np. zakończenie przypadku
  - PrepareNewEvent
    - inicjalizacja nowego przypadku
    - zmiana kryteriów klasyfikacji
- G4UserTrackingAction
  - PreUserTrackingAction
    - definiowanie własnych trajektorii
  - PostUserTrackingAction
    - usuwanie niepotrzebnych trajektorii
- G4UserSteppingAction
  - UserSteppingAction
    - niszczenie śladu w trakcie jego procesowania
    - zapisać pełną informacje o śladach

GEANT4: Pomoc i dokumentacja

- Przewodnik dla twórców aplikacji
  `http://geant4.web.cern.ch/geant4/UserDocumentation/UsersGuides/ForApplicationDeveloper/html/`
  wprowadzenie dla nowych użytkowników
  opis najbardziej przydatnych narzędzi
  opis procesu tworzenia aplikacji
  ogólny przegląd środowiska.
- Przykładowe aplikacji zawarte w dystrybucji
  `$G4INSTALL/examples`

Sterowanie symulacją

GEANT4 umożliwia sterowanie wykonaniem aplikacji na trzy sposoby:

- z poziomu kodu c++ (w programie main)
- za pomocą interwejsu użytkownika (UI)
  - z poziomu makr (poleceń UI zapisane w plikach tekstowych)

Pobranie wskaźnika do menegera UI

```
G4UImanager* UI = G4UImanager::GetUIpointer();
```

Przekazanie sterowania do UI

```
G4String macro = argv[1];
```

```
UI->ApplyCommand("/control/execute "+macro);
```

Uruchomienie programu

```
My run.mac
```

- interakcyjnie (poleceń UI przechwytywane przez interfejs)

Interfejs użytkownika (UI)

- GEANT4 umożliwia prace z różnymi interfejsami:
  - znakowy(terminal tcsh);
  - z elementami graficznymi opartymi o widżety: Motif, Athena, Qt, Windows;
  - w pełni graficzny (GAG).
- Istnieje narzędzie G4UIExecutive które automatycznie uruchamia wybrany interfejs na podstawie zmiennych środowiskowych.
  Instancja nakładki na UI

```
G4UIExecutive * ui = new G4UIExecutive(argc,argv);
```

Przekazanie sterowania do UI

```
ui->SessionStart();
```

Uruchamiany jest pierwszy interfejs który ma zdefiniowana zmienną środowiskową:
`G4UI_USE_XX (XX= QT, XM, WIN32, GAG, TCSH)`

UI : polecenia

- Polecenie UI składa się z:
  /pełnej/ścieżki/dostępu/polecenia parametrów

```
/run/beamOn 3
```

- Parametry mogą być typu logicznego, liczbą całkowitą, rzeczywistą lub ”ciągiem znaków”;
- Parametry oddzielamy spacją;
- Parametry można omijać (przyjmowana jest wtedy wartość domyślna)
- Używamy ! gdy pierwszy parametr ma wartość domyślną a drugi zadaną

```
/dir/command ! 2
```

UI : polecenia systemowe

- Interfejs znakowy posiada wbudowane polecenia charakterystyczne dla Unix'a:
  - cd, pwd, ls
  - history – pokazuje poprzednio wykonane polecenia
  - !id – ponowne wykonanie polecenia o numerze id;
  - działaja klawisze strzałek i backspace (tylko tcsh)
  - help <polecenie> - opis i składnia polecenia
  - exit – zakończenie pracy
- Powyższe polecenia dostępne tylko interaktywnie, nie mogą być wykonane z poziomu programu w c++ ani wewnątrz makra.

UI : katalogi poleceń

```
Idle> ls
Command directory path : /
 Sub-directories :
   /control/    UI control commands.
   /units/      Available units.
   /process/   Process Table control commands.
   /persistency/   Control commands for Persistency package
   /geometry/   Geometry control commands.
   /tracking/   TrackingManager and SteppingManager control commands.
   /event/      EventManager control commands.
   /cuts/       Commands for G4VUserPhysicsList.
   /run/        Run control commands.
   /random/     Random number status control commands.
   /particle/   Particle control commands.
   /gun/        Particle Gun control commands.
   /material/   Commands for materials
   /gui/   UI interactors commands.
 Commands :
```

- po instancji menegera wizualizacji pojawi się katalog `/vis/`
- mozna zdefiniować własne polecenia np. w katalogu `/My/`
- link do opisu wbudowanych poleceń UI na stronie warsztatów.

UI : makra

- Makra są plikami zawierającymi polecenia UI
- Polecenia w makrze muszą posiadać pełną ścieżkę dostępu (brak poleceń cd, ...);
- # oznacza komentarz
- Makra można wykonać na dwa sposoby:
  - z poziomu interfejsu użytkownika lub z innego makra

```
/control/execute <nazwa_makra>
```

  - z poziomu kodu c++:

```
G4UImanager * UI = G4UImanager::GetUIpointer();
UI->ApplyCommand("/control/execute <nazwa_makra>");
```

UI : aliasy

- Aliasy w GEANT4 to nazwy zastępcze oraz zmienne makr;
- Aliasy definiujemy za pomocą polecenia //

```
/control/alias [nazwa]   [wartość]
```

np.

```
Idle> /control/alias  bo  "/run/beamOn"
```

- Wartość aliasu jest zawsze traktowana jak ciąg znaków;
- Alias jest automatycznie definiowany podczas wywołania poleceń

```
/control/loop
/control/foreach
```

- Alias może być używany jako składnik polecenia wywoływanego z poziomu UI lub wewnątrz makra. Do tego celu używamy nazwy aliasa w nawiasach pisanych np.

```
Idle> {bo}  3
```

UI : pętle

- Polecenia `/control/loop` i `/control/foreach` wykonują makro więcej niż raz:
  - `/control/loop [makro] [alias] [pocz] [końc] [kro`
    `[alias]` – alias którego wartość zmienia się przy każdym wywołaniu makra od `[pocz]` o `[krok]`

```
Idle> /control/loop run.mac Ekin 2. 10. 2.
```

  - `/control/foreach [makro] [alias] [lista]`
    `[alias]` – alias do kolejnych wartości z listy `[lista]`
    `[lista]` – musi być ograniczona ()

```
Idle> /control/foreach run.mac  Ekin "1. 7. 10."
```

- wewnątrz wywoływanego makra np.:
  `/gun/energy {Ekin} GeV`

Wizualizacje

GEANT4 umożliwia wizualizację:

- geometrii detektora;
- trajektorii czastek;
- depozytów energii;
- własnych obiektów (linie, symbole, opisy, ...)

Wizualizacje umożliwiają:

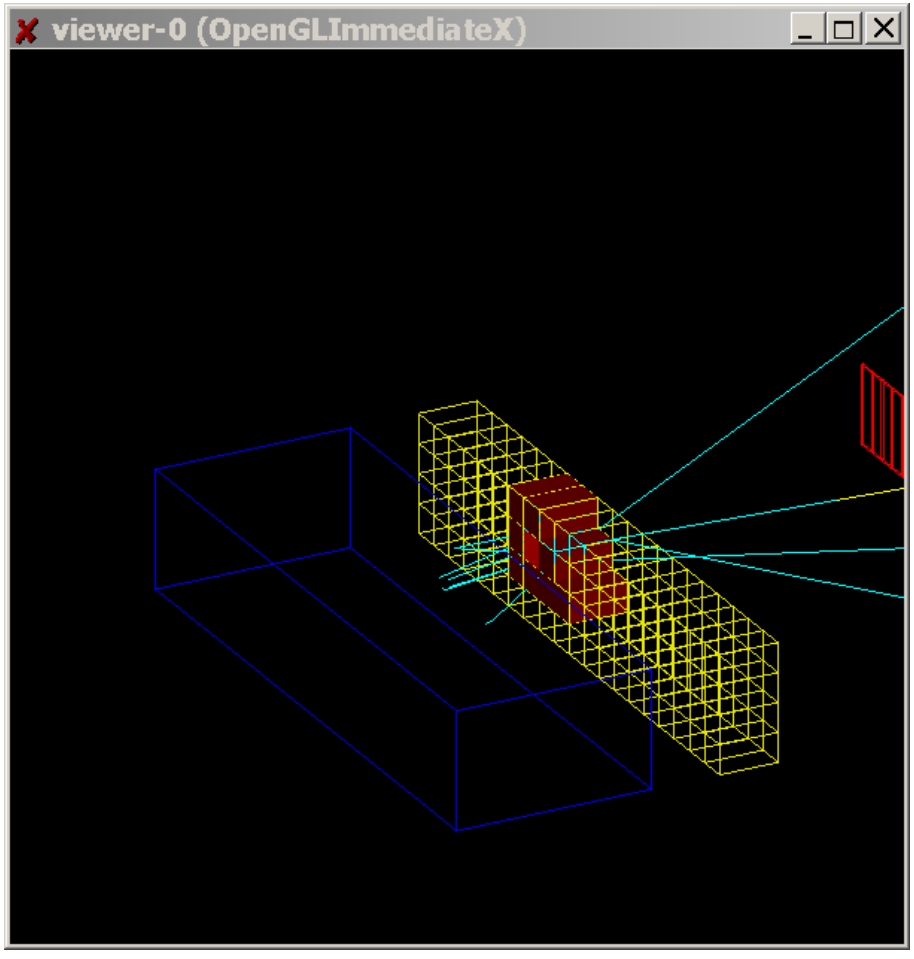
- tworzenie wysokiej jakości rysunków np. do publikacji;
- wykrywanie błędów w geometrii detektora (np. nakładanie się obszarów);
- interaktywne uzyskiwania informacji o wizualizowanych obiektach(pędy czastek, depozyty energii);

Wizualizacje: interfejsy

- GEANT4 posiada interfejsy obsługujące osiem zewnętrznych programów do wizualizacji:
  - OpenGl
  - HepRep
  - DAWN
  - OpenInventor
  - VRML
  - RayTracer
  - gMocren
  - ASCIITree
- Różne programy odpowiadają różnym potrzebom;
- Lista poleceń wydawanych z poziomu kodu c++ jest taka sama dla wszystkich programów.

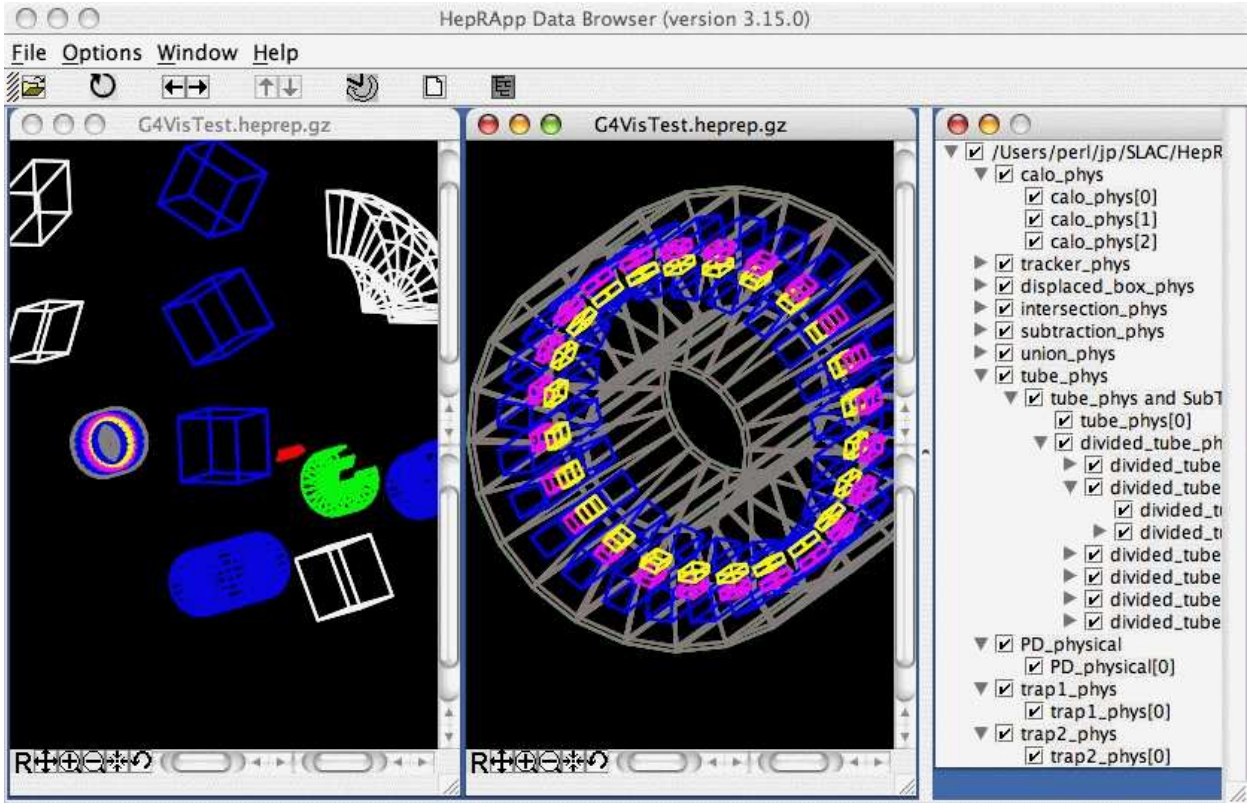
Wizualizacje: OpenGl

- Daje możliwość uzyskiwania szybkich wizualizacji: trajektorii, hitów, geometrii;
- Sterowanie tylko za pomocą poleceń interfejsu GEANT4;



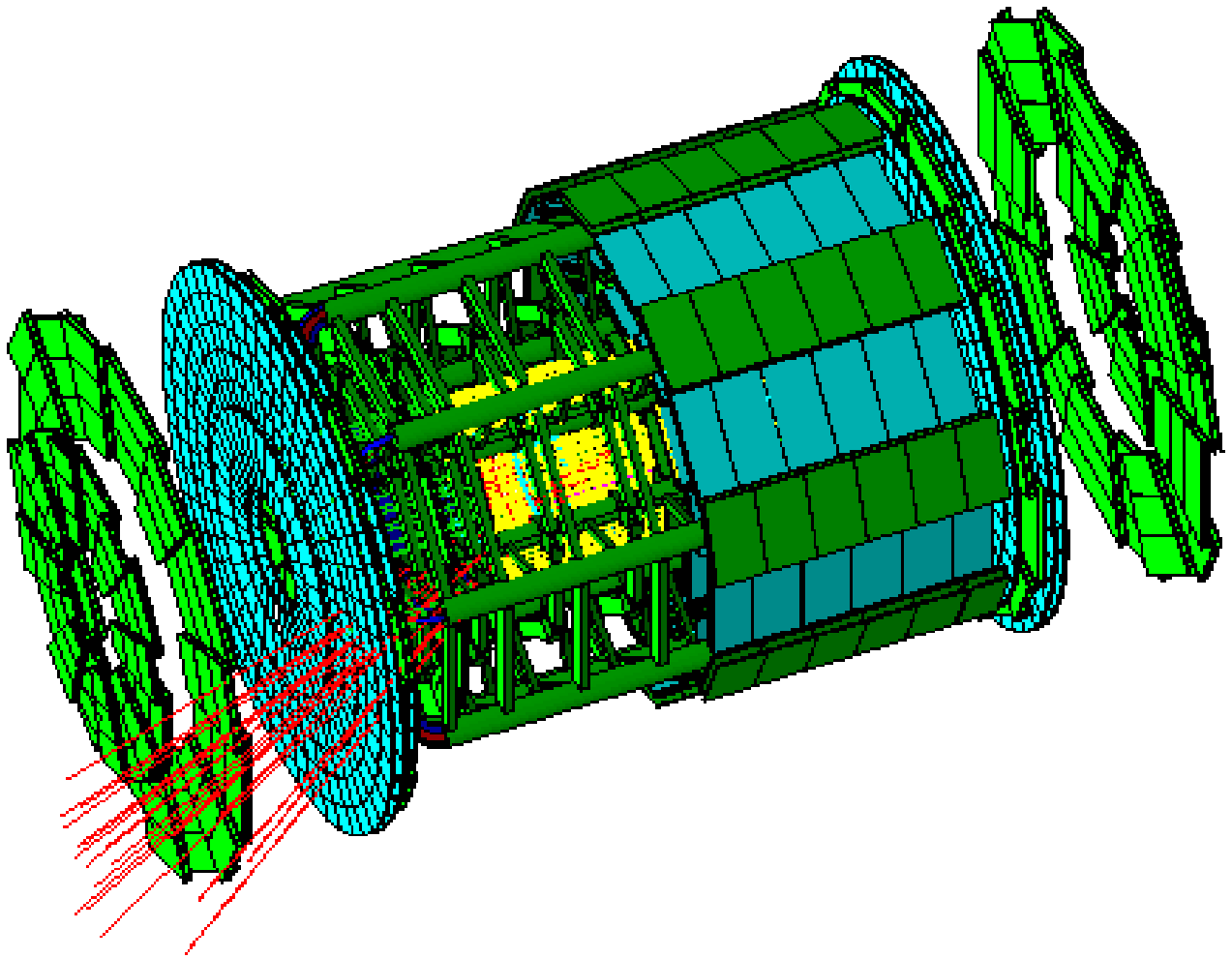
Wizualizacje: HepRep

- Daje możliwość interaktywnej (interfejs graficzny) wizualizacji: trajektorii, hitów, geometrii;
- Jest narzędziem do wykrywania błędów w geometrii detektora;
- Sterowanie za pomocą zewnętrznego interfejsu graficznego (wymaga tworzenia plików tymczasowych).



Wizualizacje: Dawn

- Daje możliwość wizualizacji: trajektorii, hitów, geometrii;
- Produkuje wysokiej jakości rysunki np. do publikacji
- Sterowanie za pomocą zewnętrznego programu (wymaga tworzenia plików tymczasowych).



Wizualizacje: interfejsy

- Sześć interfejsów niewymagających zewnętrznych bibliotek jest zawsze dostępnych w każdej instalacji GEANT4:
  - HepRep
  - DAWN
  - VRML
  - RayTracer
  - gMocren
  - ASCIITree
- Pozostałe wymagają instalacji bibliotek i ustawienia zmiennych środowiskowych podczas kompilacji jądra GEANT4;
  w szczególności OpenGL

Wizualizacje: interfejsy

- Niezależnie od interfejsu, kod źródłowy pozostaje w zasadzie taki sam;
- Wizualizacje realizujemy za pomocą: interfejsu użytkownika, z poziomu kodu c++ lub za pmocą zewnetrznych programów;
- Bezpośrednio z poziomu GEANT4 działają: OpenGL, OpenInventor, RayTracer i ASCIITree;
- Pozostałe, produkuje pliki umożliwiające wizualizacje za pomocą zewnętrznych aplikacji: HepRep, DAWN, VRML, gMocren;
- Można korzystać z kilku interfejsów w tym samym czasie (np. OpenGL/DAWN).

Wizualizacje: Manager

- Aby wizualizować należy utworzyć i zainicjalizować obiekt klasy wyprowadzonej z klasy G4VVisManager;
- Można skorzystać z gotowej implementacji G4VisExecutive;
- Użycie (w programie głównym):
  - Instancja

```
G4VisManager * visManager = new G4VisExecutive;
```

  - Inicjalizacja

```
visManager->Initialize();
```

  - Unicestwienie

```
delete visManager;
```

Wizualizacje : katalogi poleceń /vis

```
Idle> ls /vis
   /vis/ASCIITree/   Commands for ASCIITree control.
   /vis/heprep/      HepRep commands.
   /vis/rayTracer/   RayTracer commands.
   /vis/gMocren/     gMocren commands.
   /vis/ogl/         G4OpenGLViewer commands.
   /vis/modeling/    Modeling commands.
   /vis/filtering/   Filtering commands.
   /vis/geometry/    Operations on vis attributes of Geant4
   /vis/scene/       Operations on Geant4 scenes.
   /vis/sceneHandler/   Operations on Geant4 scene handlers
   /vis/viewer/      Operations on Geant4 viewers.
 Commands :
   verbose * Simple graded message scheme - digit or string
   initialize * Initialise visualisation manager.
   abortReviewKeptEvents * Abort review of kept events.
   enable * Enables/disables visualization system.
   disable * Disables visualization system.
   list * Lists visualization parameters.
```

Wizualizacje : żargon

- scene
  zbiór 3d. obiektów które chcemy wizualizować (format GEANT'a) ;
- scene handler
  obsługa sceny przygotowująca scenę do wizualizacji (format graficzny);
- viewer
  generuje obraz na podstawie danych produkowanych przez obsługę sceny;
- Interfejs wizualizacji = obsługa sceny + generator obrazu

Wizualizacje : ogólny schemat

1. Kreacja obsługi sceny i generatora obrazu
2. Kreacja pustej sceny
3. Dodanie obiektów 3D do pustej sceny
4. Przypisanie obsługi sceny do sceny
5. Ustalenie stylu grafiki (kąt patrzenia, przeźroczystość)
6. Generator obrazu dokonuje wizualizacji
7. Deklaracja zakończenia wizualizacji sceny

Wiele poleceń wykonuje kilka kroków na raz:

- `/vis/open` krok 1 i 2
- `/vis/drawVolume` krok 3,4,5 i 6

Wizualizacje: /vis/viewer/

- poniższe polecenia określają sposób wizualizacji i są wymagane tyko do pracy z OpenGL, w przypadku HepRep i DAWN odpowiednie polecenia wykonuje się później w programach zewnętrznych:
- część poleceń dostępna za pomocą skrótów klawiszowych, myszy i ich kombinacji.

1 Ustawienie kąta widzenia

```
/vis/viewer/set/viewpointThetaPhi <theta> <phi>
```

2 Powiększenie/pomniejszenie

```
/vis/viewer/zoom <scale_factor>
```

3 Powrót do początkowego sposobu wizualizacji

```
/vis/viewer/reset
```

4 Ustawienie stylu wizualizacji obiektów

```
/vis/viewer/set/style <style>
```

`<style>=wireframe, surface`
Polecenie nie działa jeśli styl wizualizacji został wymuszony w kodzie c++ za pomocą metod: `setForceWireframe` lub `setForceSolid`

Wizualizacje: /vis/viewer/ /vis/scene/add

5 Wizualizacja linii pomocniczych

```
/vis/viewer/set/auxiliaryEdge <bool>
```

6 Brak wizualizacji zasłoniętych linii

```
/vis/viewer/set/hiddenEdge <bool>
```

7 Dodanie prostoktnego układu osi

```
/vis/scene/add/axes <x0> <y0> <z0> <length> <unit>
```

8 Dodanie trajektorii czastek

```
/vis/scene/add/trajectories <smooth> <rich>
```

`smooth`
Dodanie pomocniczych punktów w celu polepszenia wizualizacji torów krzywoliniowych

Wizualizacje: Akumulacja przypadków

9 Aby wizualizować trajektorie cząstek należy je symulować

```
/run/beamOn <nevents>
```

Domyślnie zobaczymy wizualizacje każdego przypadku z osobna.

10 Akumulacja trajektorii dla wszystkich przypadków

```
/vis/scene/endOfEventAction accumulate
```

- Powrót do wizualizacji po każdym przypadku

```
/vis/scene/endOfEventAction refresh
```

- Przeglądanie zgromadzonych przypadków

```
/vis/reviewKeptEvents
```

polecenia `continue` lub `/vis/abortReviewKeptEvents`

- W kodzie c++ można użyć polecenia do gromadzenia ciekawych przypadków

```
G4EventManager->KeepTheCurrentEvent()
```

Takie rozwiązanie pozwala na symulację dużej liczby przypadków ale gromadzenie do wizualizacji tylko tych które są ciekawe.

Wizualizacje: Odświeżanie grafiki

- W OpenGl wyniki poleceń widoczne sa natychmiast w okienku graficznym. Jesli geometria, trajektorie sa bardzo skomplikowane to tak częste odświeżanie obrazu spowalnia wykonanie symulacji.

11 Aby tymczasowo wyłączyć odświeżanie:

```
/vis/viewer/set/autoRefresh false
```

11 Aby wymusić odświeżenie:

```
/vis/viewer/set/autoRefresh true
```

12 Ilością informacji o wizualizacji jaka pojawia się w okienku tekstowym sterujemy za pomocą polecenia:

```
/vis/verbose <level>
```

```
level> = quit,...,errors,..., all
```

13 W OpenGl tworzymy plik postscriptowy z grafiką za pomocą polecenia:

```
/vis/ogl/printEPS
```

W trakcie sesji powstaja pliki G4OpenGl_n.eps, gdzie n=0,1,...

Wizualizacje: pliki tymczasowe HepRep i DAWN

- Aby tworzyć pliki tymczasowe z grafiką korzystjąc z HepRep i DAWN należy utworzyć nową/kolejną obsługę sceny

```
/vis/open HepRepFile
/vis/open DAWNFILE
```

- W OpenGl wyniki poleceń widoczne sa natychmiast w okienku graficznym
- W HepRep i DAWN tymczasowy plik z wizualizacją tworzony jest w momencie wykonania polecenia

```
/run/beamOn <nevents>
```

14 Jesli chcemy utworzyć plik z wizualizacja w innym momencie

```
/vis/viewer/flush
```

W trakcie sesji powstaja pliki G4Datan.heprep oraz g4_nn.prim, gdzie n,nn=0,1,...

- trajektorie zawierają dodatkowe (niewizualizowane przez OpenGl) informacje:
  - początkowe wartości energii i pędu rodzaj cząstki, ...
  - te informacje sa dostepne poprzez kliknięcie na trajektorie w niektórych interfejsach np. HepRep
- Ta informacja może być rozszerzona poprzez polecenie:

```
/vis/scene/add/trajectories rich
```