Zastosowanie pakietu Geant4 w fizyce jądrowej Wykład 4 Budowanie geometrii

Aleksandra Fijałkowska

21 października 2021

Tworzenie geometrii

Klasą bazową dla klasy tworzącej geometrię jest `G4VUserDetectorConstruction`

Opis klasy `G4VUserDetectorConstruction` z pliku nagłówkowego:
This is the abstract base class of the user's mandatory initialization class for detector setup. It has only one pure virtual method Construct() which is invoked by G4RunManager when it's Initialize() method is invoked. The Construct() method must return the G4VPhysicalVolume pointer which represents the world volume.

Czyli: Klasa `G4VUserDetectorConstruction` jest KLASĄ ABSTRAKCYJNĄ (już wiemy, co to znaczy!). Ma jedną CZYSTO WIRTUALNĄ metodę `Construct()`. Metoda ta jest wołana przez `G4RunManager` w metodzie `Initialize()`.
Pamiętacie `G4RunManager`? Stworzyliśmy go w funkcji głównej naszego programu i zainicjowaliśmy instancją naszej klasy `DetectorConstruction`. `G4RunManager` woła metodę `Construct()`, gbybyśmy jej nie stworzyli (lub np. zrobili literówkę w nazwie, albo nazwali od małej litery), geometria by nie powstała. Bardzo trudno by było znaleźć przyczynę błędu. Na szczęście składnia języka c++ wymusza implementację metody `Construct()`.
Druga ważna informacja, płynąca z opisu klasy jest taka, że metoda `Construct()` musi zwracać obiekt typu `G4VPhysicalVolume`, który reprezentuje ŚWIAT, czyli całą geometrią. Szczegóły dotyczące typu `G4VPhysicalVolume` już za chwilę.

Tworzenie geometrii

Małe podsumowanie:

Klasa, w której stworzyliśmy geometrią nazywa się `DetectorConstruction` i jest tworzona w pliku głównym (main):

```
G4RunManager * runManager = new G4RunManager;
runManager->SetUserInitialization(new DetectorConstruction());
```

Klasa musi implementować interfejs G4VUserDetectorConstruction (czyli po nim dziedziczyć) oraz musi **obowiązkowo** implementować czysto wirtualną metodę G4VPhysicalVolume* Construct(). To właśnie w tej metodzie musi zostać stworzony świat (World) i wszystkie jego elementy.

Tworzenie geometrii układu detekcyjnego

Aby umieścić jakiś element w geometrii należy stworzyć trzy typy obiektów:

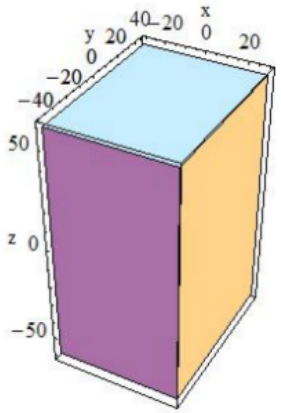
1. bryłę, czyli zmienną odpowiadającą za kształt i rozmiar elementu. Za to odpowiada typ `G4VSolid`.
2. objętość logiczną, czyli zmienną, która przyjmuje `G4VSolid` w konstruktorze, ponadto posiada informacje o materiale, kolorze, ew. polu magnetycznym. Typ to `G4LogicalVolume`.
3. objętość fizyczną, czyli zmienną, którą tworzy się umieszczając objętość logiczną w jakimś miejscu w przestrzeni. Tworząc objętość fizyczną musimy określić MATKĘ bryły, którą właśnie budujemy, czyli bryłę nadrzedną. Każdy element musi się w całości mieścić w innym elemencie, czyli musi mieć jedną matkę. Nadrzędną matką jest świat. Typ to `G4VPhysicalVolume`.

G4VSolid

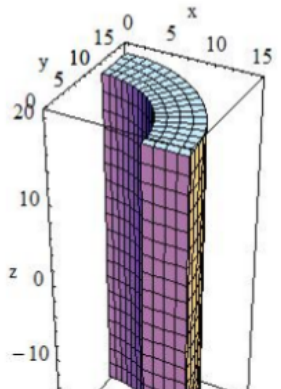
Krztałt i rozmiar elementu geometrii jest określony przez klasę `G4VSolid`. Biblioteki GEANT4 dostarczają 23 podstawowe kształty, od prostopadłościanu (G4Box) i walca (G4Tubs) po dużo bardziej wyszukane obiekty. Wszystkie ona są zademonstrowane, wraz z przykładem użycia w *Geant4 User's Guide for Application Developers*.

Na etapie tworzenia obiketu `G4VSolid` określamy rozmiary brył. Proszę zwracać uwagę na opis brył, zazwyczaj podaje się POŁOWĘ rozmiaru (połowę wysokości, szerokości itp.).

```
G4Box(const G4String& pName,
            G4double  pX,
            G4double  pY,
            G4double  pZ)
```




```
G4Tubs(const G4String& pName,
            G4double  pRMin,
            G4double  pRMax,
            G4double  pDz,
            G4double  pSPhi,
            G4double  pDPhi)
```

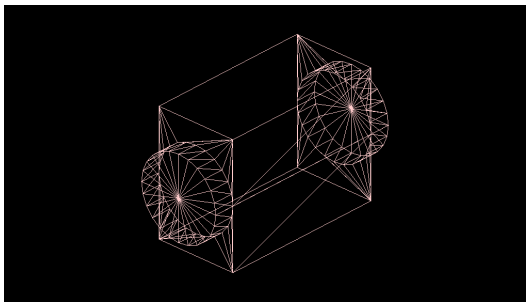
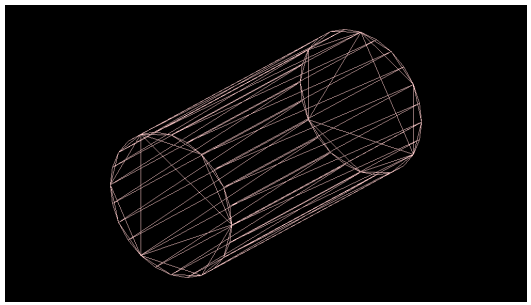
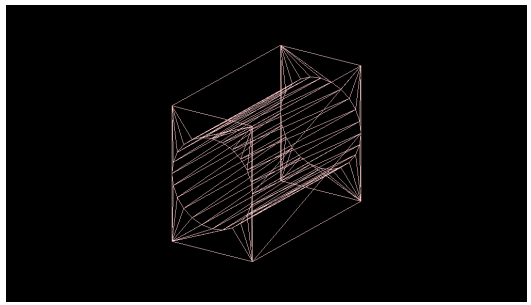


Geometria, G4VSolid cd.

Dodatkowo GEANT4 dostarcza narzędzi do logicznego składania dwóch różnych kształtów - tworzenia ich sumy, iloczynu oraz różnicy.
Przykład:

```
G4Box* box = new G4Box("Box",20*mm,30*mm,40*mm);
//G4Box (const G4String &pName, G4double pX, G4double pY, G4double pZ)
G4Tubs* cyl = new G4Tubs("Cylinder",0,20*mm,50*mm,0,360*deg);
//G4Tubs (const G4String &pName, G4double pRMin,
          G4double pRMax, G4double pDz, G4double pSPhi, G4double pDPhi)
//dodajemy dwie bryły
G4UnionSolid* un = new G4UnionSolid("Box+Cylinder", box, cyl);

//przecięcie
G4IntersectionSolid* intersec = new G4IntersectionSolid("Box*Cylinder",
                                                        box, cyl);
//odejmowanie (wycinanie)
G4SubtractionSolid* subtr = new G4SubtractionSolid("Box-Cylinder", box, cyl);
```

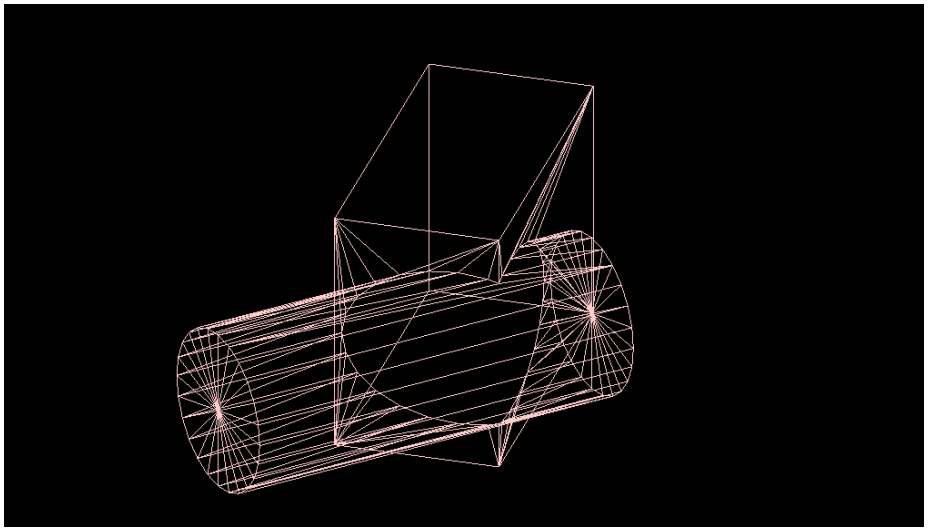




Geometria, G4VSolid cd.

Bryły, na których operujemy działaniami logicznymi mogą być dodatkowo przesuwane i obracane.

```
G4Box* box = new G4Box("Box",20*mm,30*mm,40*mm);
G4Tubs* cyl = new G4Tubs("Cylinder",0,20*mm,50*mm,0,360*deg);

G4RotationMatrix* yRot = new G4RotationMatrix;
yRot->rotateY(M_PI/4.*rad);
G4ThreeVector zTrans(0, 0, 50);
G4UnionSolid* unionMoved = new G4UnionSolid("Box+CylinderMoved",
                                            box, cyl, yRot, zTrans);
```



Geometria, G4LogicalVolume

Mając stworzony kształt i określony rozmiar nadajemy bryle materiał, ew pole, kolor. Do tego służy klasa G4LogicalVolume.
Konstruktor i opis klasy jest następujący:

```
G4LogicalVolume(G4VSolid* pSolid,
                G4Material* pMaterial,
          const G4String& name,
                G4FieldManager* pFieldMgr=0,
                G4VSensitiveDetector* pSDetector=0,
                G4UserLimits* pULimits=0,
                G4bool optimise=true);
// Constructor. The solid and material pointer must be non null.
// The parameters for field, detector and user limits are optional.
// The volume also enters itself into the logical volume Store.
// Optimisation of the geometry (voxelisation) for the volume
// hierarchy is applied by default. For parameterised volumes in
// the hierarchy, optimisation is -always- applied.
```

Geometria, G4LogicalVolume

Parametry wywołania konstruktora, które mają zadaną wartość domyślną (np. ostatni `G4bool optimise=true`) są OPCJONALNE, nie trzech ich podawać, chyba że chce się aby miały inną niż domyślna wartość.
Bryła logiczna posiada więc:

- Kształt i rozmiar (G4VSolid)
- Materiał - o tym za moment
- Nazwę
- Zdefiniowane pole np. magnetyczne (opcjonalnie)
- Informację o potencjalnej czułości (opcjonalnie, w praktyce najczęściej definiowane później)
- Określone limity na produkcję cząstek (opcjonalnie)

Proszę zwrócić uwagę na to, że klasa G4LogicalVolume przyjmuje w konstruktorze wskaźnik do obiektu typu G4VSolid. Po tym typie dziedziczą wszystkie klasy reprezentujące różne kształty brył (G4Box, G4Tube, G4Sphere itd.), dzięki temu możemy każdy z tych typów włożyć do konstruktora klasy G4LogicalVolume. Oto dobry przykład tego, jak wspaniałe jest dziedziczenie.

Budowanie materiału

Klasa G4LogicalVolume przyjmuje w konstruktorze wskaźnik do materiału, z jakiego ma być zbudowany projektowany element.
Materiały można budować na kilka sposóbów:

I. Sposób najłatwiejszy – skorzystanie z bazy gotowych materiałów.
Materiały są dostępne w klasie `G4NistManager`.
`G4NistManager` jes SINGLETONEM (przypominam omówienie testu z pierwszych zajęć), czyli nie posiada konstruktora, a jedynie statyczną metodę `Instance()`. Aby stworzyć obiekt typu `G4NistManager` piszemy:

```
G4NistManager* man = G4NistManager::Instance();
```

Następnie możemy poprosić o jakiś materiał:

```
G4Material* H2O = man->FindOrBuildMaterial("G4_WATER");
G4Material* Sci = man->FindOrBuildMaterial("G4_PLASTIC_SC_VINYLTOLUENE");
G4Material* SiO2 = man->FindOrBuildMaterial("G4_SILICON_DIOXIDE");
G4Material* Air = man->FindOrBuildMaterial("G4_AIR");
```

Pełna lista dostępnych materiałów:
http://geant4-userdoc.web.cern.ch/geant4-userdoc/UsersGuides/ForApplicationDeveloper/html/Appendix/materialNames.html
WAŻNE: Aby skorzystać z zasobów `G4NistManager` musimy dodać do nagłówków

```
#include "G4NistManager.hh"
```

Budowanie materiału

II. Jeśli materiału nie ma w bazie `G4NistManager` możemy go zbudować, podając jego skład chemiczny i podstawowe własności fizyczne. W przypadku ciał stałych podaje się gęstość, w przypadku gazu ciśnienie i temperaturę.
Skład chemiczny można podawać zgodnie ze wzorem chemicznym lub jako stosunek mas poszczególnych składników (przykład 1 i 2).
Przykład pochodzący z *Geant4 User's Guide for Application Developers* s. 139-142

```
G4double density;
G4int ncomponents;
G4NistManager* man = G4NistManager::Instance();

// define elements
G4Element* elH = man->FindOrBuildElement("H");
G4Element* elC = man->FindOrBuildElement("C");
G4Element* elO = man->FindOrBuildElement("O");
G4Element* elN = man->FindOrBuildElement("N");

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

//case 2: mixture by fractional mass
density = 1.290*mg/cm3;
G4Material* Air = new G4Material(name="Air " , density, ncomponents=2);
Air->AddElement(elN, fractionmass=0.7);
Air->AddElement(elO, fractionmass=0.3);
```

Budowanie materiału


```
//case 3: define a material from elements and/or others materials
density = 0.200*g/cm3;
G4Material* Aerog = new G4Material(name="Aerogel", density, ncomponents=3);
Aerog->AddMaterial(SiO2, fractionmass=62.5*perCent);
Aerog->AddMaterial(H2O , fractionmass=37.4*perCent);
Aerog->AddElement (elC , fractionmass= 0.1*perCent);
```

Proszę zwrócić uwagę, że w ostatnim przypadku budujemy materiał z innych materiałów, które muszą być wcześniej stworzone.

Pierwiastki

W poprzednich przykładach ponownie skorzystaliśmy z `G4NistManager`, tym razem do uzyskania pierwiastków chemicznych. Możemy jednak zrobić je sami, podając ich liczbę atomową i masę molową, określając nazwę i symbol:
Geant4 User's Guide for Application Developers s. 139-142

```
G4String name, symbol;
G4double a, z, n;
//a=mass of a mole
//z=mean number of protons;
G4int ncomponents;
G4double abundance;
G4UnitDefinition::BuildUnitsTable();

a = 1.01*g/mole;
G4Element* elH = new G4Element(name="Hydrogen",symbol="H" , z= 1., a);

a = 12.01*g/mole;
G4Element* elC = new G4Element(name="Carbon" ,symbol="C" , z= 6., a);
...
a = 207.20*g/mole;
G4Element* elPb = new G4Element(name="Lead" ,symbol="Pb", z=82., a);
```

Można nawet stworzyć pierwiastek jako mieszaniana izotopów:

```
// define an Element from isotopes, by relative abundance
G4Isotope* U5 = new G4Isotope(name="U235", z=92, n=235, a=235.01*g/mole);
G4Isotope* U8 = new G4Isotope(name="U238", z=92, n=238, a=238.03*g/mole);
G4Element* elU = new G4Element(name="enriched Uranium", symbol="U", ncomponents=2);
elU->AddIsotope(U5, abundance= 90.*perCent);
elU->AddIsotope(U8, abundance= 10.*perCent);
```

Geometria, G4Material, przykłady cd., gazy

Na koniec przykład z gazami, dla których chcemy precyzyjnie określić gęstość, ciśnienie i temperaturę *Geant4 User's Guide for Application Developers* s. 139-142

```
// examples of gas in non STP conditions
density = 27.*mg/cm3;
pressure = 50.*atmosphere;
temperature = 325.*kelvin;
G4Material* CO2 = new G4Material(name="Carbonic gas", density, ncomponents=2,
                                 kStateGas,temperature,pressure);
CO2->AddElement(elC, natoms=1);
CO2->AddElement(elO, natoms=2);

density = 0.3*mg/cm3;
pressure = 2.*atmosphere;
temperature = 500.*kelvin;
G4Material* steam = new G4Material(name="Water steam ", density, ncomponents=1,
                                   kStateGas,temperature,pressure);
steam->AddMaterial(H2O, fractionmass=1.);

// What about vacuum ? Vacuum is an ordinary gas with very low density
density = universe_mean_density; //from PhysicalConstants.h
pressure = 1.e-19*pascal;
temperature = 0.1*kelvin;
new G4Material(name="Galactic", z=1., a=1.01*g/mole, density,
               kStateGas,temperature,pressure);
```

Podsumowanie

GEANT4 dostarcza 3 typy związane z materiałami

- G4Isotope – reprezentuje izotop, a więc zbiór atomów o określonej liczbie atomowej i masowej
- G4Element – reprezentuje pierwiastek chemiczny, czyli zbiór atomów o określonej liczbie atomowej. G4Element ma pola klasy takie jak: nazwa, symbol, liczba atomowa, liczba izotopów oraz wektor trzymający wskaźniki do tych izotopów, wektor częstości występowania tych izotopów w przyrodzie (jako średnia liczba atomów danego izotopu na jednostkę objętości). Obiekty G4Element możemy utworzyć sami (posługując się uprzednio stworzonymi G4Isotope), lub skorzystać z bazy NIST (przykład poniżej).
- G4Material – reprezentuje ostateczny materiał, z którego zbudowane są elementy konstrukcyjne symulowanego układu. Własności materiału to: nazwa, gęstość, stan skupienia, temperatura, ciśnienie, pierwiastki z którego jest zbudowany oraz ich proporcje (wrażone w w formie liczby atomów w cząsteczce lub jako procent wagowy). Materiał może być stworzony przez użytkownika, albo wyciągnięty z bazy NIST.

G4LogicalVolume

Posiadając wskaźnik do potrzebnego materiału i kształtu możemy wreszcie skończyć budowanie elementu układu – stworzyć instancję klasy G4LogicalVolume

```
G4LogicalVolume(G4VSolid* pSolid,
                G4Material* pMaterial,
          const G4String& name,
                G4FieldManager* pFieldMgr=0,
                G4VSensitiveDetector* pSDetector=0,
                G4UserLimits* pULimits=0,
                G4bool optimise=true);
```

Przykład:

```
G4double radiusMin = 0;
G4double radiusMax = 60*cm;
G4double length = 170*cm;
G4Tubs* fantomSolid = new G4Tubs("fantomSolid", radiusMin, radiusMax,
                                 length/2., 0*deg, 360*deg);

G4NistManager* man = G4NistManager::Instance();
G4Material* water = man->FindOrBuildMaterial("G4_WATER");

G4LogicalVolume* fantomLogVol = new G4LogicalVolume(fantomSolid, water,
                                                    "fantomLogVol");
```

G4VisAttributes

Stworzenie obiektu typu G4LogicalVolume jest dobrym momentem aby określić sposób wyświetlania budowanego elementu, za co odpowiada klasa G4VisAttributes
G4VisAttributes reprezentuje cechy wyświetlania danego obiektu (widoczność, kolor, przezroczystość, widoczność wewnętrznych krawędzi itp).
Klasa ma kilka konstruktorów, ja zawsze używam drugiego z wymienionych:

```
G4VisAttributes (G4bool visibility) \\argumentem jest flaga widoczności
G4VisAttributes (const G4Colour &colour) \\argumentem jest kolor
G4VisAttributes (G4bool visibility, const G4Colour &colour)
\\argumentami są jest flaga widoczności i kolor
```

Inne przydatne metody:

```
void SetColour (const G4Colour &)
void SetLineWidth (G4double)
void SetForceSolid (G4bool) //wyświetlanie jako "pełną bryłę"
void SetForceAuxEdgeVisible (G4bool) //widoczność wewnętrznych krawędzi
```

Klasa G4Colour, którą przyjmuje w konstruktorze G4VisAttributes reprezentuje kolor. Konstruktor klasy przyjmuje 4 liczby reprezentujące kanały R, G, B i alfa (przeźroczystość).

```
G4Colour (G4double r=1., G4double g=1., G4double b=1., G4double a=1.)
```

Ponadto klasa zawiera szereg **statycznych publicznych** metod, zwracający jeden z podstawowych kolorów:

```
static G4Colour  White ()
static G4Colour  Grey ()
static G4Colour  Black ()
static G4Colour  Brown ()
static G4Colour  Red ()
```

```
static G4Colour  Green ()
static G4Colour  Blue ()
static G4Colour  Cyan ()
static G4Colour  Magenta ()
static G4Colour  Yellow ()
```

Kolor np. zielony możemy stworzyć na dwa sposoby:

```
G4Colour zielony(0,1,0, 1);//R=0, G=1, B=0, alfa=1
G4Colour zielony = G4Colour::Green();
```

A gdyby ktoś koniecznie chciał mieć wskaźnik, a nie wartość:

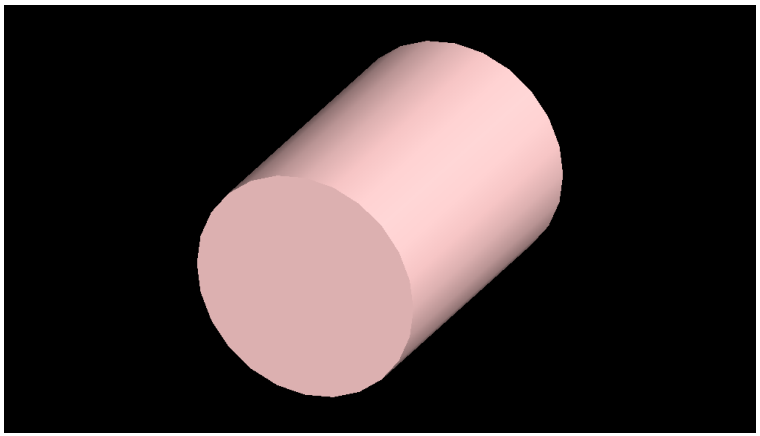
```
G4Colour* zielony = new G4Colour(0,1,0, 1);
```

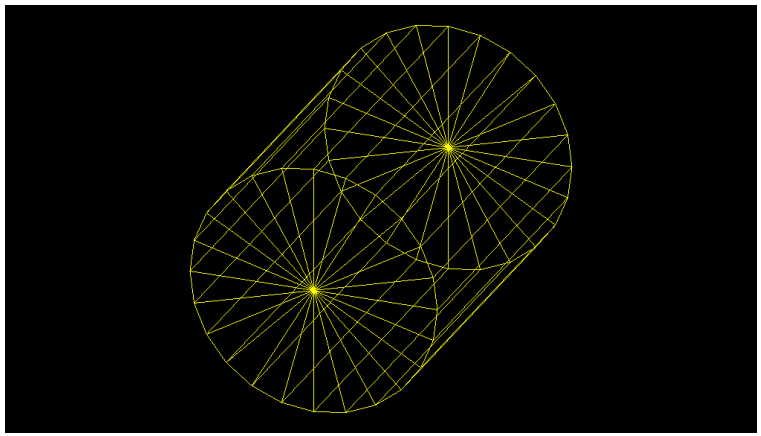
G4VisAttributes, przykłady


```
G4LogicalVolume* fantomLogVol = new G4LogicalVolume(fantomSolid, water,
                                                    "fantomLogVol");

//kolor różowy, ścianki "pełne"
G4VisAttributes* fantomVisAtt = new G4VisAttributes( G4Colour(1,0.8,0.8));
fantomVisAtt->SetForceAuxEdgeVisible(true);
fantomVisAtt->SetForceSolid(true);
fantomLogVol->SetVisAttributes(fantomVisAtt);

//kolor żółty, ścianki przeźroczyste, widoczne krawędzie
G4VisAttributes* fantomVisAtt = new G4VisAttributes( G4Colour::Yellow ());
fantomVisAtt->SetForceAuxEdgeVisible(true);
fantomLogVol->SetVisAttributes(fantomVisAtt);
```





G4PhysicalVolume, obszar fizyczny

Mamy już kształt (G4VSolid), materiał (G4Material), kolor (G4VisAttributes) i wszystko "zapakowane" do zmiennej typu G4LogicalVolume. Obszar logiczny należy gdzieś umieścić w przestrzeni. Każdy obiekt typu G4LogicalVolume, **z wyjątkiem świata**, musi się w CAŁOŚCI mieścić w jednej, nadrzędnej bryle - matce. Świat jest "główną" matką. Umieszczenie obszaru logicznego w przestrzeni jest jednoznaczne z utworzeniem instancji klasy G4PhysicalVolume.
Konstruktor klasy G4PhysicalVolume:

```
G4VPhysicalVolume (G4RotationMatrix *pRot, const G4ThreeVector &tlate,
                   const G4String &pName, G4LogicalVolume *pLogical,
                   G4VPhysicalVolume *pMother)
```

Ten sam skutek wywiera wywołanie konstruktora G4PVPlacement, który dziedziczy (a więc rozszerza) G4PhysicalVolume:

```
G4PVPlacement (G4RotationMatrix *pRot, const G4ThreeVector &tlate,
               G4LogicalVolume *pCurrentLogical, const G4String &pName,
               G4LogicalVolume *pMotherLogical, G4bool pMany,
               G4int pCopyNo, G4bool pSurfChk=false)
```

G4PhysicalVolume, obszar fizyczny

Opis parametrów konstruktora klasy G4PVPlacement:

```
G4PVPlacement (G4RotationMatrix *pRot, const G4ThreeVector &tlate,
               G4LogicalVolume *pCurrentLogical, const G4String &pName,
               G4LogicalVolume *pMotherLogical, G4bool pMany,
               G4int pCopyNo, G4bool pSurfChk=false)
```

- G4RotationMatrix *pRot – macierz rotacji, dotyczy rotacji osi bryły względem osi MATKI. Wskaźnik może być zerowy jeśli nie chcemy rotować bryły.
- const G4ThreeVector &tlate – wektor przesunięcia środka lokalizowanej bryły względem środka bryły matki.
- G4LogicalVolume *pCurrentLogical – wskaźnik do obszaru logicznego, który chcemy umieścić.
- const G4String &pName – nazwa obszaru fizycznego.
- G4VPhysicalVolume *pMotherLogical – wskaźnik do obszaru logicznego bryły MATKI. Świat ma w tym miejscu wskaźnik zerowy.
- G4bool pMany – cytując za dokumentacją GEANT4 Ćurrently NOT used. For future use to identify if the volume is meant to be considered an overlapping structure, or not." Ja zawsze wpisuję 0.
- G4int pCopyNo – numer kopii, jeśli ta sama bryła logiczna jest umieszczana w przestrzeni więcej niż jeden raz warto jest ją "ponumerować". Daje to możliwość dostania się do własności interesującej bryły. TO JEST WAŻNE
- G4bool pSurfChk – oznaczenie jako prawda aktywuje sprawdzanie, czy bryła nie pokrywa się z innymi, już ulokowanymi.

G4PhysicalVolume, obszar fizyczny

Jest jeszcze druga wersja konstruktora klasy G4PVPlacement, która zamiast macierzy rotacji i wektora przesunięcia ma obiekt tranformacji 3D G4Transform3D, który uwzględnia obie operacje.

```
G4PVPlacement::G4PVPlacement( const G4Transform3D &Transform3D,
                              const G4String& pName,
                              G4LogicalVolume *pLogical,
                              G4VPhysicalVolume *pMother,
                              G4bool pMany,
                              G4int pCopyNo,
                              G4bool pSurfChk );
```

```
//przykład użycia
G4ThreeVector pos(0.0, 10*cm,0.0);
G4RotationMatrix rot;
rot.rotateZ(-90*deg);
G4Transform3D transform(rot, centerPosition);
```

Skorzystanie z tej wersji konstruktora klasy G4PVPlacement jest użyteczne gdy dodajemy jakiś element wieloktornie w pętli. Nie musimy się wtedy martwić o tworzenie nowych wskaźników do G4RotationMatrix

G4PhysicalVolume, obszar fizyczny, przykład

Matka:

```
G4VPhysicalVolume* DetectorConstruction::ConstructWorld()
{
    G4double worldX = 5.*m;
    G4double worldY = 5.*m;
    G4double worldZ = 5.*m;
    G4Material* vaccum = new G4Material("GalacticVacuum", 1., 1.01*g/mole,
                            CLHEP::universe_mean_density,
                            kStateGas, 3.e-18*pascal, 2.73*kelvin);
    G4Box* worldSolid = new G4Box("worldSolid",worldX,worldY,worldZ);
    worldLogic = new G4LogicalVolume(worldSolid, vaccum,"worldLogic", 0,0,0);
    G4VPhysicalVolume* worldPhys = new G4PVPlacement(0, G4ThreeVector(),
                                          worldLogic, "world", 0, false, 0);
    return worldPhys;
}
```

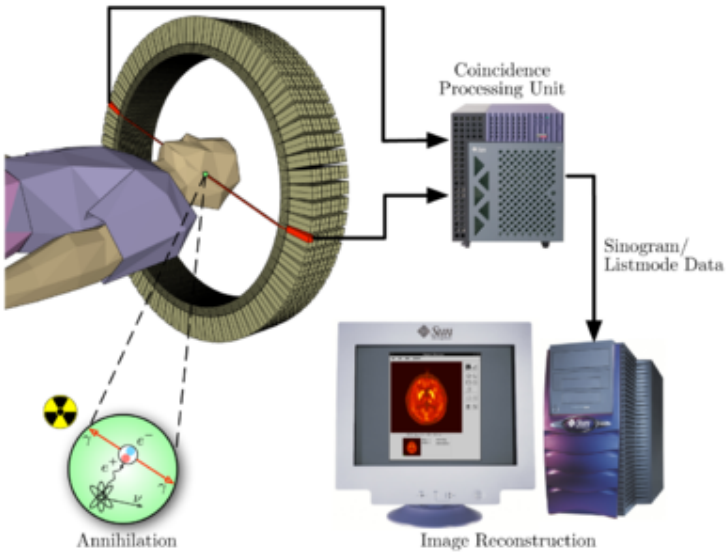
Fantom:

```
G4Tubs* fantomSolid = new G4Tubs("fantomSolid", radiusMin, radiusMax,
                                 length/2., 0*deg, 360*deg);
G4LogicalVolume* fantomLogVol = new G4LogicalVolume(fantomSolid, water,
                                                    "fantomLogVol");
G4RotationMatrix* rot = new G4RotationMatrix;
rot->rotateX(M_PI/4.*rad);
G4ThreeVector pos(0,0,10*cm);
new G4PVPlacement(rot, pos, fantomLogVol, "fantom", worldLogic, 0, 0);
```

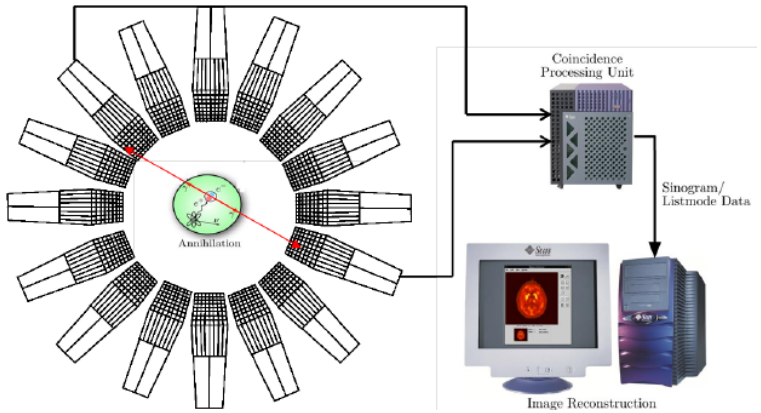
PET

Zaprogramuj uproszczoną geometrię skanera PET.

- Średnica wewnętrzna torusa otaczającego pacjenta wynosi 80 cm, przy czym 1 cm stanowi warstwa polipropylenu
- Układ detekcyjny składa się z 1 pierścienia (w rzeczywistości jest więcej!)
- Pierścienie zbudowane są z kryształów NaI o rozmiarach 8x8x10 cm (szerokość x wysokość x grubość)
- Każdy kryształ otoczony jest 2 mm warstwą teflonu
- Całość "zanurzona"jest w polipropylenie



https://pl.wikipedia.org



https://www.researchgate.net/figure/A-schematic-of-the-PET-principle_fig5_266887992

PET

Kroki:

- Dodaj kręgosłup do pacjenta (na rozgrzewkę)
- Stwórz polipropylenowy torus
- Zbuduj pojedynczy detektor (teflon + NaI)
- Rozmieść detektory w formie pierścienia w warstwie polipropylenu (pamiętaj o numerowaniu kopii!)
- Umieść pierścień, wraz z detektorami wewnątrz świata
- Rozdziel elementy konstrukcyjne na osobne klasy (np klasa fantom i klasa pierścień). Proponuję, aby każda klasa, która będzie trzymać elementy geometrii miała publiczną metodę:
  void Place(G4RotationMatrix *pRot,
  G4ThreeVector &tlate,
  const G4String &pName,
  G4LogicalVolume *pMotherLogical,
  G4int pCopyNo = 0);