



Geant4 @ NCBJ, 2022

3a. Geometria

czyli

zróbmy sobie świat*

* aby tylko nie Brave New World

Przemysław Adrich



NARODOWE
CENTRUM
BADAŃ
JĄDROWYCH
ŚWIERK



Źródła

Ten w wykład w dużej mierze bazuje na materiałach opracowanych przez

G. Cosmo, M. Asai, M. Novak, V. Ivanchenko

na potrzeby oficjalnych kursów Geant4 w CERN. W szczególności kursu:

[Geant4 beginners at CERN, Geneva \(Switzerland\), 25-31 May 2021](#)

Plan kursu

Wykłady

1. Wstęp do Geant4. Rys historyczny. Zastosowania. Przegląd możliwości. Instalacja. Dokumentacja.
2. Podstawowa struktura kodu. Hierarchie klas. Klasy użytkownika (obowiązkowe, opcjonalne).
~~Interfejsy. System jednostek. Liczby losowe. Śledzenie przebiegu symulacji („verbosity”).~~
3. **Geometria i materiały.**
4. Źródło. Fizyka. Wizualizacja.
5. Detektory typu „primitive scorer”, „probe”. (Phasespace).
6. Detektory użytkownika („Hits”).
7. Obiekty typu „UserAction” jako detektory. Histogramy i n-tuple. Niepewność statystyczna w obliczeniach Monte Carlo. Geant4 na klastrze CiŚ.

* „Monte Carlo First Run
(wykład bonusowy)

Przegląd zagadnień pozostawionych na przyszłość:

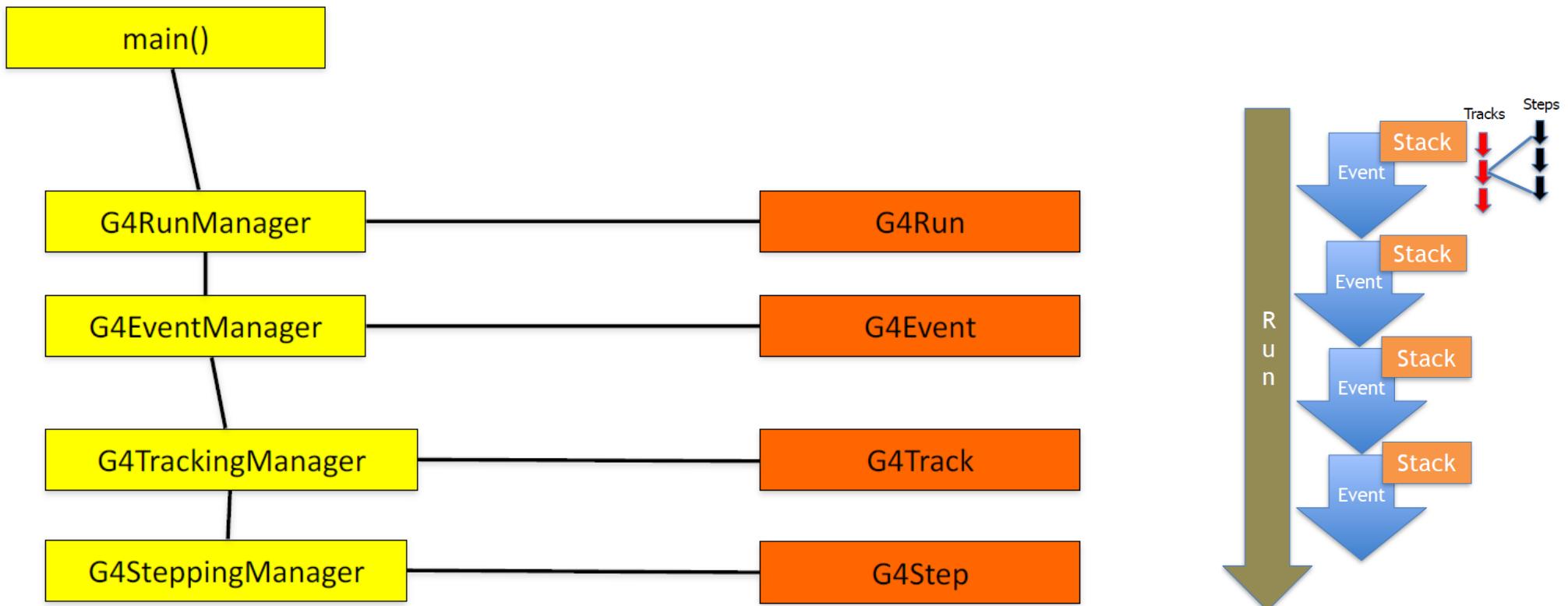
- wielowątkowość („multithreading”),
- własne interfejsy („messengers”),
- interfejs Roota (histogramy, n-tuple), interfejs python
- redukcja wariancji, „physics biasing”, „event biasing”, „geometrical biasing”,
- fotony optyczne, fizyka hadronowa, procesy i cząstki użytkownika,
- obcięcia energetyczne zależne od cząstki, regionu geometrii,
- zmiany geometrii i detektorów w trakcie wykonania programu,
- pole EM,
- światy równoległe,
- trackInformation, eventInformation, runInformation
- „stacking”,
- fast simulation,
- import geometrii z CAD,
- periodic boundary conditions,
- specjalistyczne kody bazujące na Geant4 (G4Beamline, GAMOS, GATE ...),
- ...

Geant4 – wprowadzenie. Co jest potrzebne by zbudować aplikację?

- Geant4 jest zestawem bibliotek i interfejsów. Użytkownik musi zbudować własną aplikację.
- W tym celu trzeba:
 - **Zdefiniować układ:**
 - **geometria, materiały**
 - Zdefiniować fizykę, która ma być stosowana w symulacji:
 - Cząstki, procesy/modele procesów fizycznych
 - Progi produkcji
 - Określić od czego mają zaczynać się zdarzenia:
 - Generator cząstek pierwotnych (źródło)
 - Zadbać o wydobycie użytecznych informacji
- Opcjonalnie można również:
 - Wizualizować geometrię, trajektorie, wyniki fizyczne
 - Dodać interfejs użytkownika (np. graficzny)
 - Zdefiniować własne komendy, itd., itd.

Żeby to wszystko osiągnąć musimy poznać podstawową strukturę kodu Geant4

Geant4 – podstawowa struktura



Zapożyczone od M. Asai za pośrednictwem J.Apostolakis

Geant4 – podstawowa struktura

G4RunManager

- „Główny rozgrywający” - zarządza jądrem Geant4
- Steruje przebiegiem symulacji:
 - dba o stworzenie obiektu klasy G4Run
 - powołuje do życia menadżerów niższych szczebli i wydaje im rozkazy, a oni powołują kolejnych podwładnych, itd.
- „Singleton”, który musi być „powołany do życia” przez użytkownika
- Użytkownik musi stworzyć i zarejestrować do RunManagera trzy obiekty odpowiedzialne za:
 - **opis geometrii, detektorów, pola (G4VUserDetectorConstruction)**
 - konfigurację cząstek i procesów fizycznych (G4VUserPhysicsList lub G4VModularPhysicsList)
 - generację cząstek pierw *Lekko myląca nazwa – może sugerować, że chodzi tylko o elementy aktywne (zbierające jakąś wielkość fizyczną). Podczas gdy interwencję (dostęp do dania naprawdę chodzi o znacznie więcej):*
 - Użytkownik może zarejestrować interwencję (dostęp do dania naprawdę chodzi o znacznie więcej):
 - Całą materiałną strukturę geometryczną układu (elementy pasywne i aktywne)
 - Przypisanie detektorów (w znaczeniu elementów zbierających dane) do konkretnych elementów geometrii (opcjonalnie)
 - Zdefiniowanie pól (opcjonalnie)

Geant4 – podstawowa struktura – funkcja main()

moja_mala_aplikacja_geant4.cc

```
#include "G4RunManager.hh"
#include "MyDetectorConstruction.hh"
#include "FTFP_BERT.hh"
#include "MyActionInitialization.hh"

int main(int argc,char** argv) {
    G4RunManager* runManager = new G4RunManager; → kernel = new G4RunManagerKernel();
                                                eventManager = kernel->GetEventManager();

    runManager->SetUserInitialization(new MyDetectorConstruction());
    G4VModularPhysicsList* MyPhysicsList = new FTFP_BERT;
    runManager->SetUserInitialization(MyPhysicsList);
    runManager->SetUserInitialization(new MyActionInitialization()); → kernel->DefineWorldVolume(userDetector->Construct(), ...);

    runManager->Initialize(); → if(!geometryInitialized) InitializeGeometry();
                                    if(!physicsInitialized) InitializePhysics();

    G4UIExecutive* UI = new G4UIExecutive(argc, argv);
    UI->SessionStart(); → /run/beamOn 100

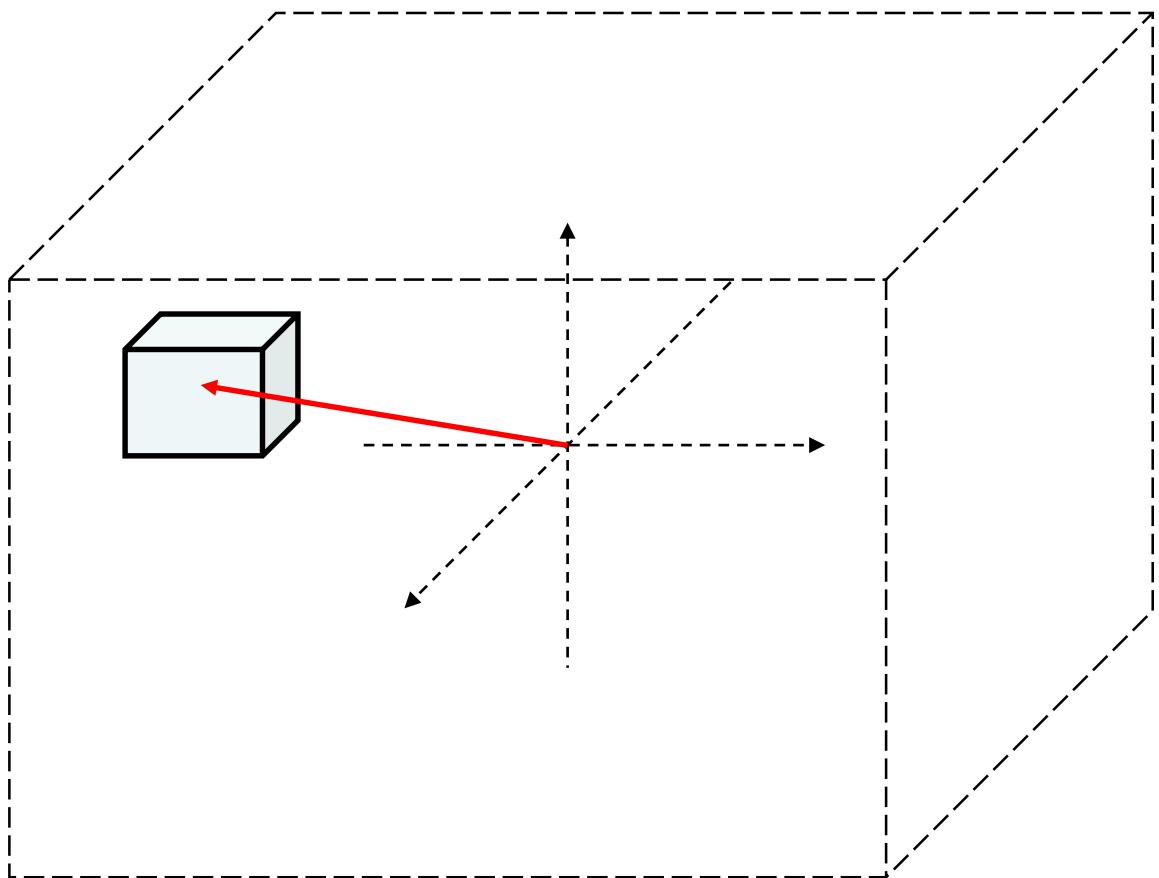
    delete UI;
    delete runManager;
    return 0;
}
```

Wnioski:

- Większość wykonywanego kodu nie jest bezpośrednio widoczna
- Bardzo dużo dzieje się „automagicznie” ;)

Geometria

Geometria – układ(y) współrzędnych

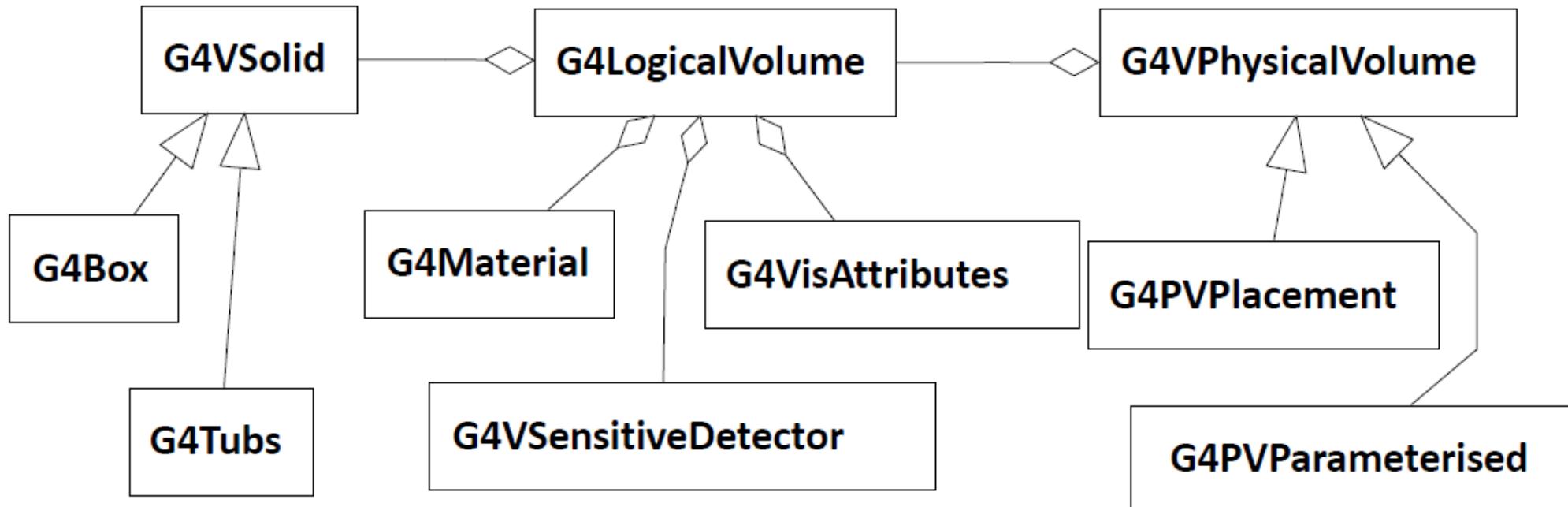


- *Globalny układ współrzędnych ma początek w geometrycznym środku bryły „świat”.*
- *Każda inna bryła ma swój lokalny układ współrzędnych, którego środek pokrywa się z geometrycznym środkiem tej bryły.*
- *Każda bryła, oprócz bryły „świat”, jest umiejscawiana wewnątrz innej bryły („matki”).*
- *Umiejscawiając bryłę wewnątrz innej bryły podajemy wektor translacji (w lokalnym układzie współrzędnych) i macierz obrotu (o niej w dalszej części).*
- *Bryła córka nie może wystawać poza bryłę matkę.*
- *Bryły córki wewnątrz wspólnej matki nie mogą na siebie nachodzić.*

Geometria – koncepcja budowy bryły

Bryła umiejscowiona w geometrii składa się z trzech warstw:

- **G4VSolid** – opisuje kształt i rozmiary
- **G4LogicalVolume** – opisuje materiał, zawiera wskaźniki do brył córek, aktywnych detektorów, limitów użytkownika, pól, itp.
- **G4VPhysicalVolume** – opisuje pozycję i rotację w przestrzeni.



Geometria – tworzenie brył. Przykładowy fragment kodu.

```
G4VSolid* pBoxSolid = new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
```

Kształt Nazwa (string)

1.*m, 2.*m, 3.*m)

Rozmiary. Uwaga! Podajemy połowy długości boków!

Zawsze pamiętać o jednostkach! (Standardową jednostką długości w Geant4 jest [mm]).

Geometria – tworzenie brył. Przykładowy fragment kodu.

```
G4VSolid* pBoxSolid = new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
```

Kształt Nazwa (string)

Rozmiary. Uwaga! Podajemy połowy długości boków!

Zawsze pamiętać o jednostkach! (Standardową jednostką długości w Geant4 jest [mm]).

```
G4LogicalVolume* pBoxLog = new G4LogicalVolume(
```

pBoxSolid,	Wskaźnik do obiektu typu G4VSolid
pBoxMaterial,	Wskaźnik do obiektu definiującego materiał
"aBoxLog",	Nazwa (string)
pFieldMgr=0,	Wskaźnik do menadżera pola (standardowo pusty)
pSDetector=0,	Wskaźnik do obiektu obsługującego aktywny detektor
pULimits=0,	Wskaźnik do limitów użytkownika
optimise=true);	Flaga optymalizacji.

Geometria – tworzenie brył. Przykładowy fragment kodu.

```
G4VSolid* pBoxSolid = new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
```

Kształt Nazwa (string)

Rozmiary. Uwaga! Podajemy połowy długości boków!

Zawsze pamiętać o jednostkach! (Standardową jednostką długości w Geant4 jest [mm]).

```
G4LogicalVolume* pBoxLog = new G4LogicalVolume( pBoxSolid,
```

pBoxMaterial,

"aBoxLog",

pFieldMgr=0,

pSDetector=0,

pULimits=0,

optimise=true); Flaga optymalizacji.

Wskaźnik do obiektu typu G4VSolid

Wskaźnik do obiektu definiującego materiał

Nazwa (string)

Wskaźnik do menadżera pola (standardowo pusty)

Wskaźnik do obiektu obsługującego aktywny detektor

Wskaźnik do limitów użytkownika

Dopiero teraz nasza bryła pojawi się w modelu geometrii, przez którą transportowane są cząstki

```
G4VPhysicalVolume* aBoxPhys = new G4PVPlacement( pRotation,
```

Macierz obrotu (wskaźnik)

G4ThreeVector(X, Y, Z), Wektor translacji (referencja)

pBoxLog, Wskaźnik do LogicalVolume

"aBoxPhys", Nazwa (string)

pMotherLog, Wskaźnik do LogicalVolume matki

0, Currently NOT used. For future use

copyNo, Numer kopii.

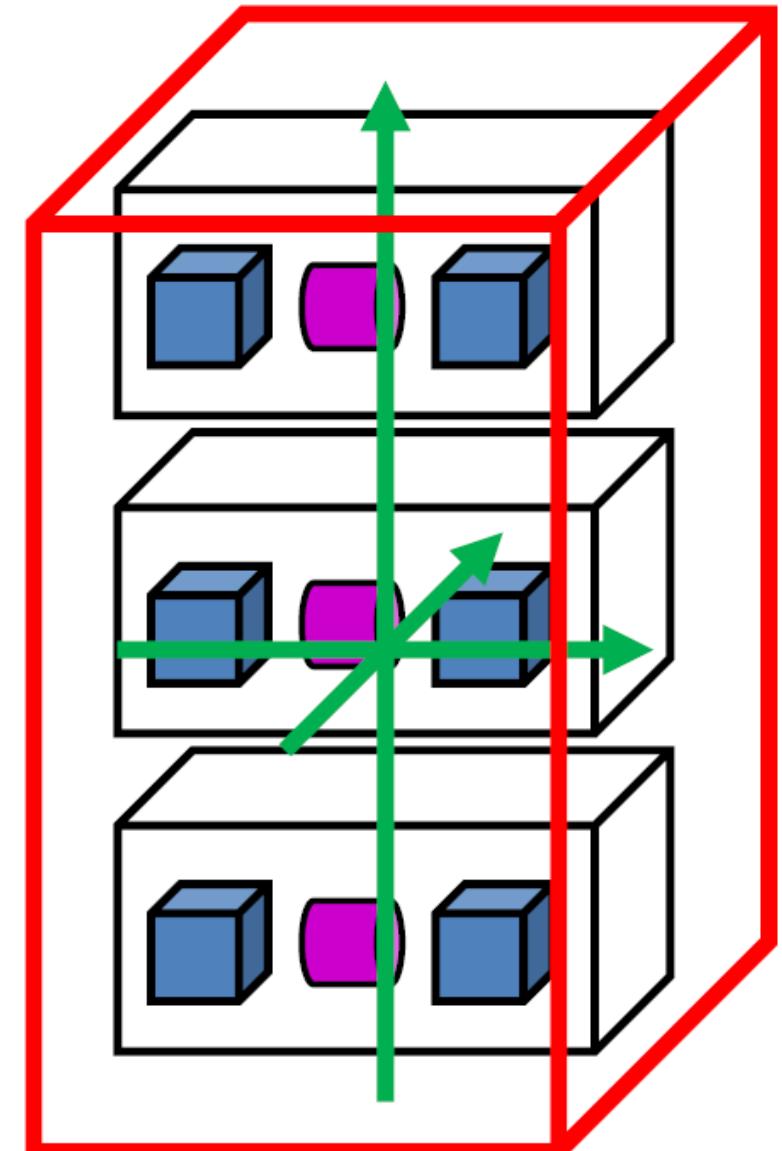
pSurfChk=false); Sprawdzanie kolizji (opcjonalnie; przez losowanie 10000 punktów na powierzchni bryły). Może być czasochłonne!

Jest pusty (NULL) jeśli tworzymy bryłę „świat”

Tę samą bryłę logiczną można umiejscowić wielokrotnie w różnych miejscach

Geometria – jeszcze o hierarchii brył

- Każda bryła logiczna może być wykorzystana (umiejscowiona) wielokrotnie.
- Kilka brył córek może być umiejscowionych we wspólnej bryle matce.
- Informacja o relacji matka-córka przechowywana jest w bryle logicznej (*LogicalVolume*) mimo iż definiujemy ją osadzając bryłę logiczną w geometrii, czyli tworząc bryłę fizyczną (*PhysicalVolume*). Z tego wynika, że:
 - jeśli jakąś bryłę logiczną osadzimy w geometrii wielokrotnie, to wszystkie jej córki również automatycznie pojawią się we wszystkich utworzonych egzemplarzach brył fizycznych.
- Bryła „świat” musi być unikalna (można stworzyć tylko jedną). Wszystkie inne bryły fizyczne muszą w całości mieścić się wewnątrz bryły świat.
- Bryła świat definiuje globalny układ współrzędnych.
- Pozycja cząstki (track) podawana jest względem globalnego układu współrzędnych.



Geometria – jak zrobić własne DetectorConstruction?

- Z klasy bazowej **G4VUserDetectorConstruction** wyprowadź własną klasę pochodną.
Może się nazywać dowolnie, byle publicznie dziedziczyła po G4VUserDetectorConstruction.

Geometria – klasa bazowa G4VUserDetectorConstruction

Deklaracja klasy. Typowo w pliku nagłówkowym typu „.hh”.

(Definicja, czyli ciała funkcji składowych, najczęściej znajdują się w pliku typu „.cc”.)

```
class G4VUserDetectorConstruction
{
public:
    G4VUserDetectorConstruction();
    virtual ~G4VUserDetectorConstruction();
    virtual G4VPhysicalVolume* Construct() = 0; // Konstruktor
    virtual void ConstructSDandField(); // Destruktor („wirtualny” – nie trzeba go definiować)
// tu jest jeszcze trochę kodu ale nie musimy teraz się nim zajmować
protected:
    void SetSensitiveDetector(G4LogicalVolume* logVol, G4VSensitiveDetector* aSD);
};
```

Uwaga na marginesie.
To „`= 0`” w deklaracji funkcji składowej czyni tę klasę czysto abstrakcyjną (tzn. powoduje, że nie da się utworzyć obiektu tej klasy).

- Metoda `Construct()` musi zwrócić wskaźnik do objętości fizycznej bryły „świat”. Za pomocą tej jednej bryły jądro Geant'a ma dostęp do całej geometrii i będzie potrafiło w niej nawigować (transportować cząstki przez geometrię).
- Objętości czynne (detektory) i pola powinny być stworzone w metodzie `ConstructSDandField()`. O tym na kolejnym wykładzie.

Uwagi na marginesie.

`public:` wszystko zadeklarowane w tej sekcji jest dostępne z zewnątrz obiektu danej klasy.

`private:` i `protected:` wszystko co jest zadeklarowane tutaj jest dostępne tylko dla obiektu danej klasy (wzg. klasy zaprzyjaźnionej).

`virtual` oznacza, że dana funkcja składowa (metoda) jest zadeklarowana ale nie musi zostać zdefiniowana. Może się skończyć na dobrych chęciach ;)
`Typ void` to tzw. typ pusty. Funkcja zadeklarowana jako zwracająca wartość typu `void` po prostu nic nie zwraca (i nie wolno jej czegokolwiek zwrócić!)

Geometria – jak zrobić własne DetectorConstruction?

- Z klasy bazowej **G4VUserDetectorConstruction** wyprowadź własną klasę pochodną. Może się nazywać dowolnie, byle publicznie dziedziczyła po G4VUserDetectorConstruction.
- Zaimportuj (zdefiniuj) własną metodę **Construct()** i opcjonalnie **ConstructSDandField()**.
- W obrębie swojej klasy pochodnej klasy:
 1. Stwórz wszystkie potrzebne materiały (zwykle robię to w konstruktorze; ewentualnie w odrębnej funkcji wywoływanej z konstruktora).
 2. Zdefiniuj objętość fizyczną „świat”. (bryła -> objętość logiczna -> objętość fizyczna).
 3. Zdefiniuj inne potrzebne bryły i objętości logiczne.
 4. Osadź objętości logiczne w modelu geometrii (zaczynając od tych, które są osadzone bezpośrednio w objętości świat).
 5. Jeśli potrzeba dołącz pola (np. magnetyczne) do odpowiednich objętości logicznych (opcjonalnie).
 6. Stwórz detektory (sensitive detectors) i połącz je z właściwymi objętościami logicznymi (opcjonalnie).
 7. Zdefiniuj atrybuty wizualizacji, np. kolory, przeźroczystość, widoczność (opcjonalnie).
 8. Zdefiniuj tzw. regiony (opcjonalnie). Fizyka i progi energetyczne mogą się zmieniać pomiędzy regionami.
 9. Pamiętaj, że **Construct()** musi zwrócić wskaźnik do objętości fizycznej reprezentującej „świat”.
- W funkcji main() stwórz jeden obiekt twojej klasy DetectorConstruction i przekaż RunManagerowi wskaźnik do tego obiektu.

Geometria – jak zrobić własne DetectorConstruction?

```
// DetectorConstruction.hh

#ifndef DetectorConstruction_h
#define DetectorConstruction_h 1

#include "G4VUserDetectorConstruction.hh"
#include "G4NistManager.hh"
class G4LogicalVolume;
class G4VPhysicalVolume;
class G4Box;
class G4Tubs;

class DetectorConstruction : public G4VUserDetectorConstruction
{
public: ← Interfejs publiczny
    DetectorConstruction();
    virtual ~DetectorConstruction();
    virtual G4VPhysicalVolume* Construct();
    void ConstructSDandField();

    void SetWorldMaterial(G4String newMaterial);
    void SetupPhantom();
    void SetPDDPhantomNLayers(G4int newNLayers);
    G4int GetPDDPhantomNLayers();
    void SetPDDPhantomLayerThickness(G4double newPDDPhantomLayerThickness);

private: ← Implementacja.
    ...
};

#endif
```

Geometria – jak zrobić własne DetectorConstruction?

```
// DetectorConstruction.hh

#ifndef DetectorConstruction_h
#define DetectorConstruction_h 1
#include "G4VUserDetectorConstruction.hh"
#include "G4NistManager.hh"

class DetectorConstruction : public G4VUserDetectorConstruction
{
public:
    DetectorConstruction();
    virtual G4VPhysicalVolume* Construct();
    ...

private:
    G4NistManager* nist;
    // World:
    G4double world_size_X, world_size_Y, world_size_Z;
    G4Box* solidWorld;
    G4LogicalVolume* logicWorld;
    G4VPhysicalVolume* physWorld;
    G4Material* World_Material;
    // Phantom:
    G4Material* PhantomMaterial;
    G4double TankOuterRadius;
    G4Tubs* solidTank;
    G4LogicalVolume* logicTank;
    G4VPhysicalVolume* physTank;
    G4int PDDPhantomNLayers;
    G4double PDDPhantomLayerThickness;
};

#endif
```



Geometria – jak zrobić własne DetectorConstruction?

```
// DetectorConstruction.cc

#include "DetectorConstruction.hh"
G4bool checkOverlaps = true;

DetectorConstruction::DetectorConstruction()
: G4VUserDetectorConstruction(),
 // World:
 solidWorld(0), logicWorld(0), physWorld(0),
 World_Material(0),
 world_size_X(100*cm), world_size_Y(100*cm), world_size_Z(200*cm),
 // Phantom:
 PDDPhantomNLayers(40), PDDPhantomLayerThickness(2.0*mm)
{
    nist = G4NistManager::Instance();
    World_Material = nist->FindOrBuildMaterial("G4_AIR");
    PhantomMaterial = nist->FindOrBuildMaterial("G4_WATER");
}

G4VPhysicalVolume* DetectorConstruction::Construct()
{
    G4cout << " DetectorConstruction::Construct() Przystepuje do budowy Swiata ..." << G4endl;

    solidWorld = new G4Box("World", 0.5*world_size_X, 0.5*world_size_Y, 0.5*world_size_Z);
    logicWorld = new G4LogicalVolume(solidWorld, World_Material, "World");
    physWorld = new G4PVPlacement(0, G4ThreeVector(), logicWorld, "World", 0, false, 0,
                                 checkOverlaps);

    SetupPhantom();

    return physWorld;
}
```

Inicjalizacje

Klasa abstrakcyjna. Wszystkie bryły w Geant4 dziedziczą po G4VSolid

- Definiuje, ale nie implementuje wszystkie funkcje potrzebne do:
 - obliczenia odległości pomiędzy bryłą a punktem w przestrzeni,
 - sprawdzeniem czy punkt znajduje się wewnątrz bryły,
 - obliczenia maksymalnych rozmiarów (rozciągłości) bryły,
 - znalezienia płaszczyzny normalnej do bryły w zadanym punkcie.
- Użytkownik może tworzyć własne klasy pochodne od G4VSolid.

Geometria - bryły

CSG (Constructed Solid Geometry) solids

- G4Box, G4Tubs, G4Cons, G4Trd,
G4Para, G4Trap, G4Torus, G4CutTubs,
G4Orb, G4Sphere

Specific solids (CSG like)

- G4Polycone, G4Polyhedra, G4Hype,
G4Ellipsoid, G4EllipticalTube, G4Tet,
G4EllipticalCone, G4Hype,
G4GenericPolycone, G4GenericTrap,
G4Paraboloid

Tessellated solids

- G4TessellatedSolid, G4ExtrudedSolid

Boolean & scaled solids

- G4UnionSolid, G4SubtractionSolid,
G4IntersectionSolid, G4MultiUnion,
G4ScaledSolid

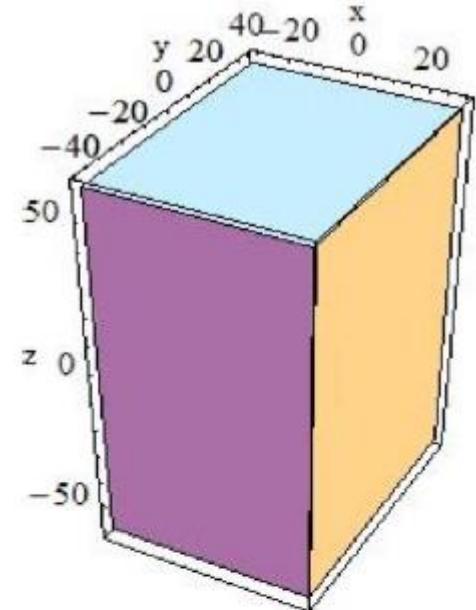
Twisted shapes

- G4TwistedBox, G4TwistedTrap,
G4TwistedTrd, G4TwistedTubs

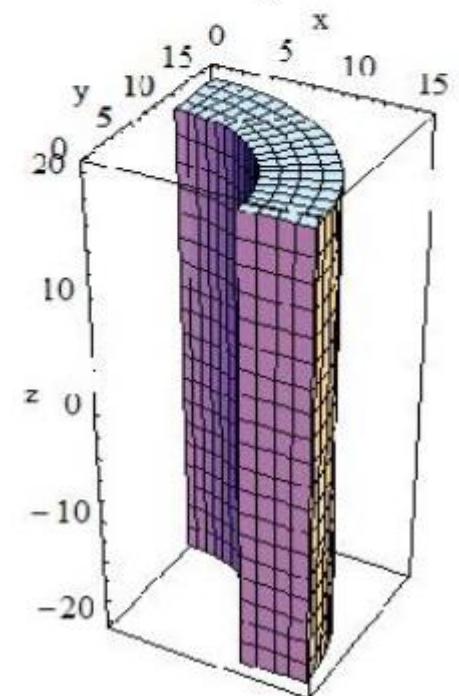


Geometria – prosty przykład brył podstawowych

```
G4Box(const G4String &pname,           // nazwa  
       G4double half_x,    // połowa długości boku X  
       G4double half_y,    // połowa długości boku Y  
       G4double half_z); // połowa wysokości boku Z
```



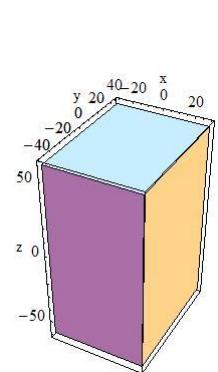
```
G4Tubs(const G4String &pname,           // nazwa  
        G4double pRmin,     // promień wewnętrzny  
        G4double pRmax,     // promień zewnętrzny  
        G4double pDz,       // połowa wysokości (Z)  
        G4double pSphi,     // kąt φ rozpoczęcia segmentu  
        G4double pDphi);   // szerokość kątowa segmentu
```



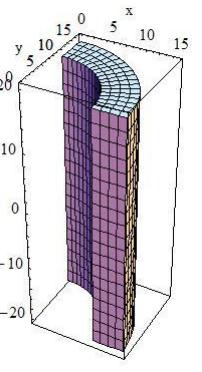
Geometria – pozostałe bryły podstawowe

Elementarne bryły (Constructive Solid Geometry)

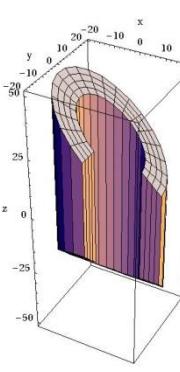
- Najbardziej wydajny kod
- Możliwa implementacja własnych brył



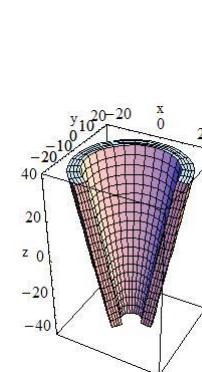
G4Box



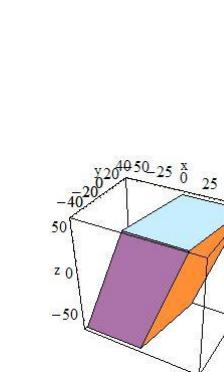
G4Tubs



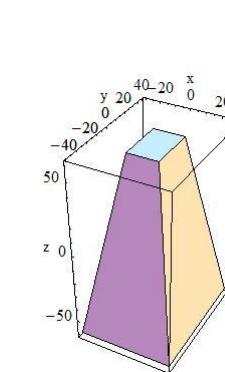
G4CutTubs



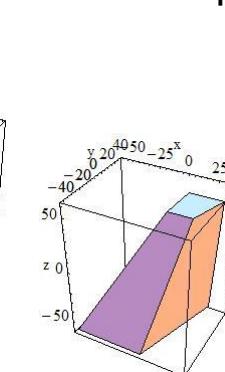
G4Cons



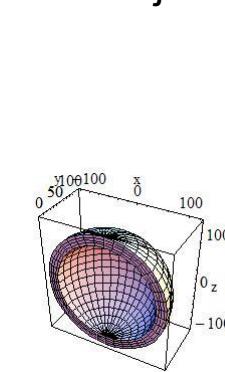
G4Para



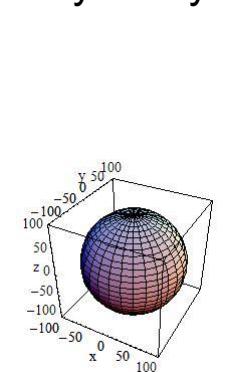
G4Trd



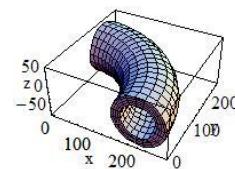
G4Trap



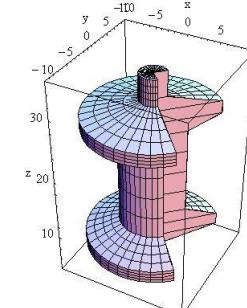
G4Sphere



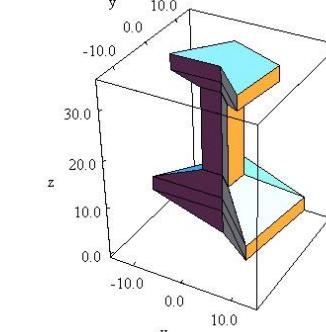
G4Orb



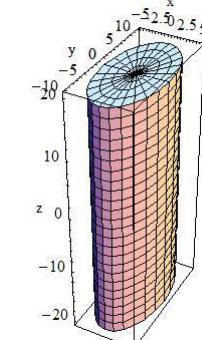
G4Torus



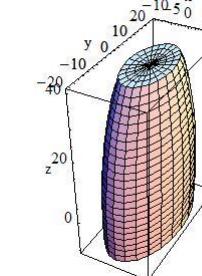
G4Polycone



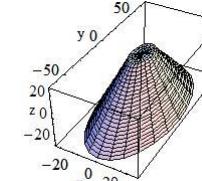
G4Polyhedra



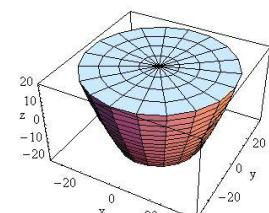
G4EllipticalTube



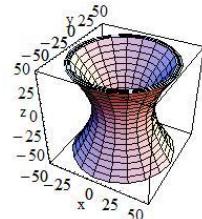
G4Ellipsoid



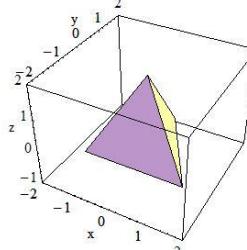
G4EllipticalCone



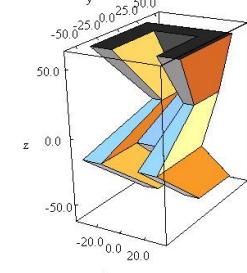
G4Paraboloid



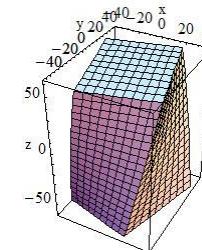
G4Hype



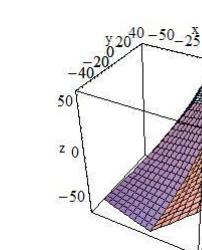
G4Tet



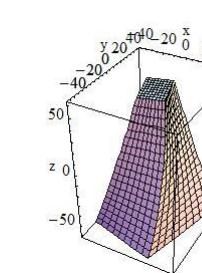
G4ExtrudedSolid



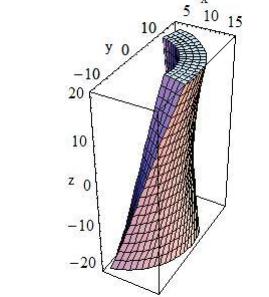
G4TwistedBox



G4TwistedTrap



G4TwistedTrd

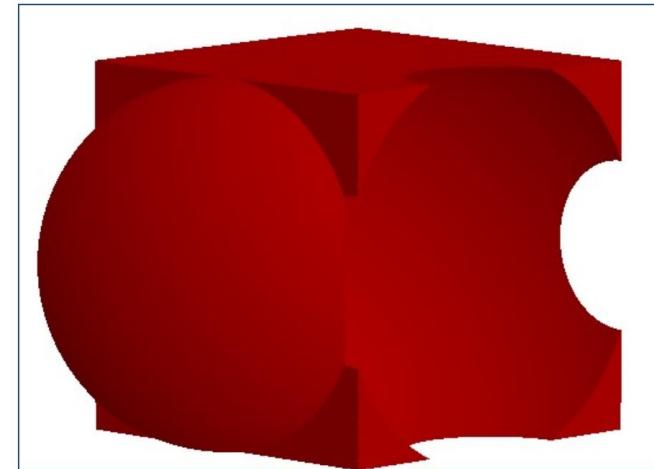


G4TwistedTubs

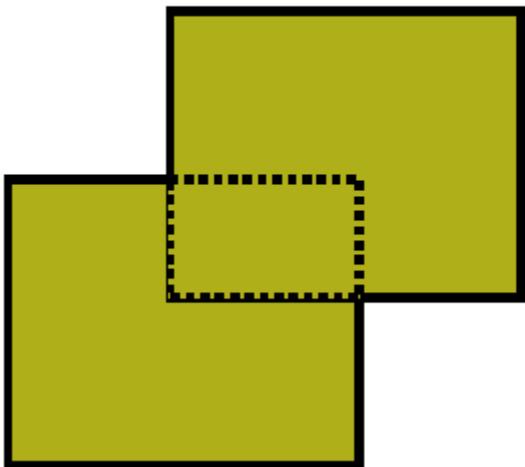
Geometria – operacje logiczne na bryłach

Na bryłach można wykonywać operacje logiczne (Boolean operations):

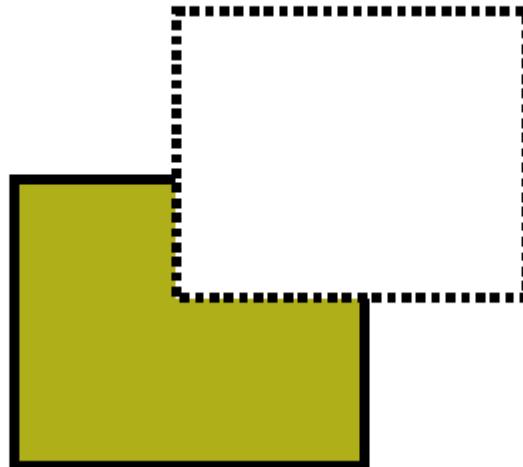
- **G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid**
- Potrzebne są: 2 bryły, 1 operacja logiczna, (opcjonalnie) transformacja drugiej bryły
- Druga bryła jest (opcjonalnie) transformowana względem lokalnego układu współrzędnych pierwszej bryły.
- Operacja logiczna zwraca trzecią bryłę
- Kolejna bryła może być składana operatorem logicznym z trzecią bryłą.



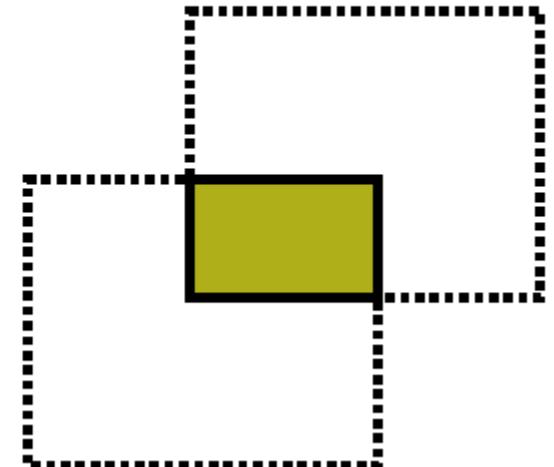
G4UnionSolid



G4SubtractionSolid



G4IntersectionSolid



[Geant4 Homepage](#)

[Book For Application Developers](#)



[Docs](#) » [Detector Definition and Response](#) » [Geometry](#) » [Solids](#)

Solids

The GEANT4 geometry modeller implements Constructive Solid Geometry (CSG) representations for geometrical primitives. CSG representations are easy to use and normally give superior performance.

All solids must be allocated using 'new' in the user's program; they get registered to a [G4SolidStore](#) at construction, which will also take care to deallocate them at the end of the job, if not done already in the user's code.

All constructed solids can stream out their contents via appropriate methods and streaming operators.

For all solids it is possible to estimate the geometrical volume and the surface area by invoking the methods:

```
G4double GetCubicVolume()  
G4double GetSurfaceArea()
```

which return an estimate of the solid volume and total area in internal units respectively. For elementary solids the functions compute the exact geometrical quantities, while for composite or complex solids an estimate is made using Monte Carlo techniques.

<https://geant4-userdoc.web.cern.ch/UsersGuides/ForApplicationDeveloper/html/Detector/Geometry/geomSolids.html>

Geometria - G4LogicalVolume

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

- Zawiera wszystkie informacje o obiekcie geometrycznym poza jego pozycją (i rotacją) w przestrzeni:
 - kształt i rozmiary (G4VSolid)
 - materiał, czułość (czy jest aktywnym detektorem), atrybuty wizualizacji, masę, objętość,
 - obiekty córki znajdujące się wewnątrz,
 - pola, progi produkcji, przynależność do regionu.
- Objętości fizyczne tego samego rodzaju mogą współdzielić jeden obiekt typu G4LogicalVolume
- Nie wolno utworzyć objętości logicznej z pustym (NULL) wskaźnikiem do bryły (solid).
- Wskaźnik do materiału nie może być pusty (nie dotyczy geometrii świata równoległego).
- Klasa G4LogicalVolume nie jest przeznaczona do dziedziczenia.

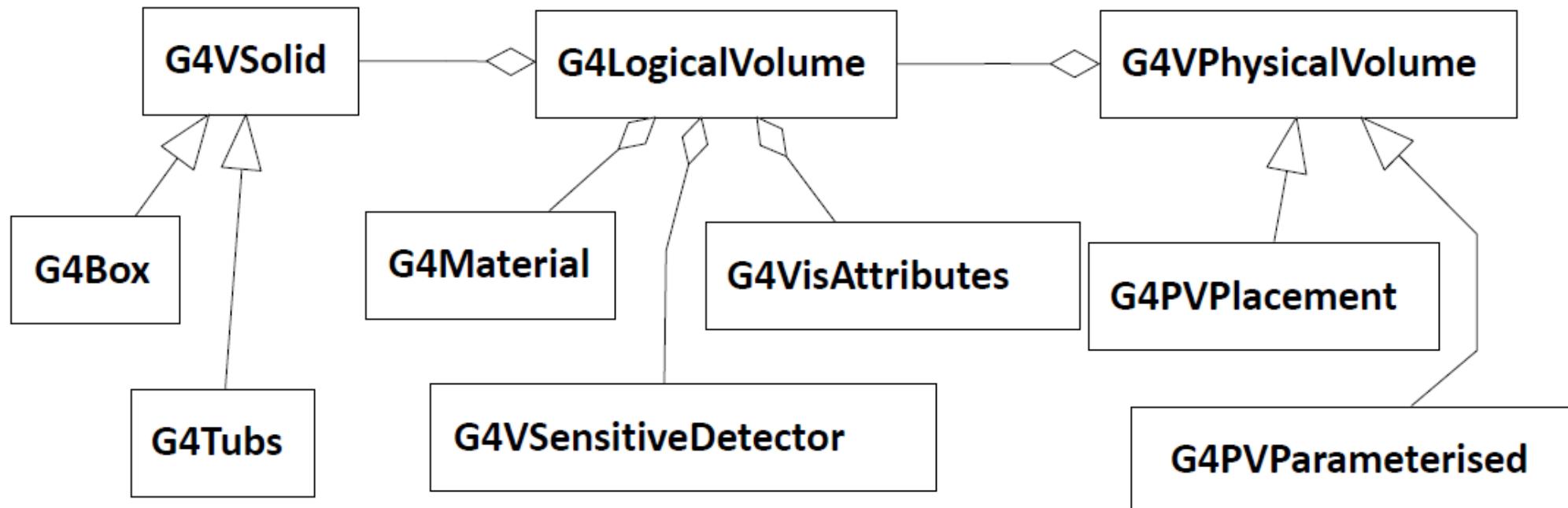
Geometria – objętość i masa układu

- Geometryczną objętość bryły (także w przypadku powstałej z operacji logicznej) można obliczyć za pomocą metody składowej **GetCubicVolume()** (zadeklarowanej w klasie bazowej G4VSolid)
 - Dla większości brył objętość jest obliczana dokładnie (ze wzoru); dla bardziej złożonych jest szacowana przez całkowanie metodą Monte Carlo
- Całkowita masa układu (lub poszczególnych elementów) może być obliczona za pomocą metody składowej **GetMass(...)** klasy G4LogicalVolume:
GetMass(G4bool forced=false, G4bool propagate=true, G4Material* pMaterial=0);
 - Obliczenia mogą być długotrwałe w zależności od złożoności geometrii (zaleca się raczej wywoływać GetMass dla poszczególnych objętości logicznych a nie dla objętości logicznej „świat”),
 - Zwrócona masa jest zapamiętywana i zwracana bez ponownego obliczania przy kolejnych wywołaniach (cache) chyba, że **forced=true**,
 - Objętości córki będą ignorowane jeśli ustawi się **propagate=false**,
 - Podając wskaźnik do wybranego materiału, można sprawdzić „co by było gdyby”.

Geometria - G4VPhysicalVolume

Bryła umiejscowiona w geometrii składa się z trzech warstw:

- **G4VSolid** – opisuje kształt i rozmiary
- **G4LogicalVolume** – opisuje materiał, zawiera wskaźniki do brył córek, aktywnych detektorów, limitów użytkownika, itp.
- **G4VPhysicalVolume** – opisuje pozycję i rotację w przestrzeni.



Geometria – tworzenie brył

```
G4VSolid* pBoxSolid = new G4Box("aBoxSolid", 1.*m, 2.*m, 3.*m);
```

Kształt *Nazwa (string)* *Rozmiary. Uwaga! Podajemy połowy długości boków!
Zawsze pamiętać o jednostkach!*

<u>G4VPhysicalVolume</u> *	aBoxPhys = new <u>G4PVPlacement</u> (pRotation,	Macierz obrotu (wskaźnik)
		G4ThreeVector(X, Y, Z),	Wektor translacji (referencja)
<i>Tutaj fizyczną bryłę konstruujemy i umieszczamy w geometrii za pomocą klasy G4PVPlacement (Physical Volume Placement).</i>		pBoxLog,	Wskaźnik do LogicalVolume
<i>Powstanie jeden egzemplarz.</i>		“aBoxPhys”,	Nazwa (string)
		pMotherLog,	Wskaźnik do LogicalVolume matki
		0,	Currently NOT used. For future use
		copyNo,	Numer kopii.
		nSurfChk=false):	Sprawdzanie kolizji (opcjonalnie)

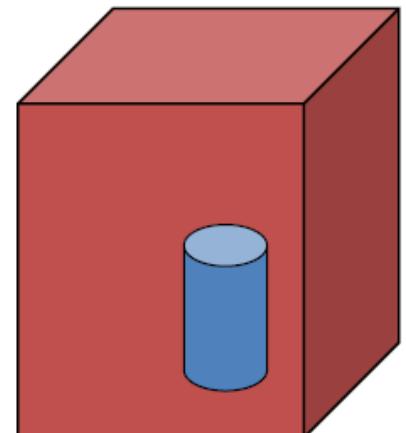
Tutaj fizyczną bryłę konstruujemy i umieszczamy w geometrii za pomocą klasy G4PVPlacement (Physical Volume Placement). Powstanie jeden egzemplarz.

Powstanie jeden egzemplarz.

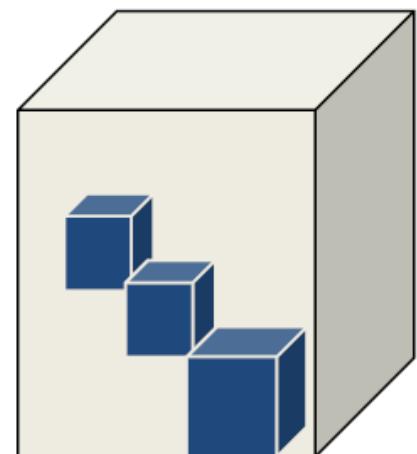


Geometria – Sposoby tworzenia i umiejscawiania brył fizycznych (PhysicalVolume)

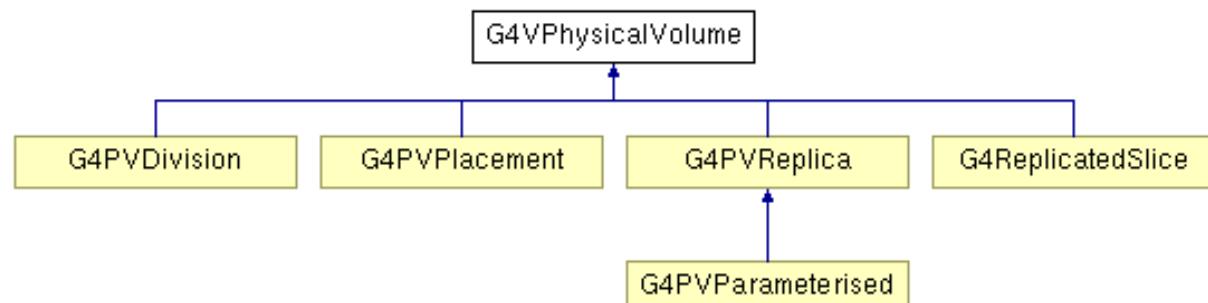
- „Placement volume” ([G4PVPlacement](#)) – pojedyncza objętość (bryła) umieszczona jednokrotnie w geometrii (może w swoim wnętrzu zawierać bryły córki)
 - Jedna objętość fizyczna (G4PhysicalVolume) reprezentuje jedną „rzeczywistą” bryłę.
- „Repeated volume” – jedna objętość (bryła) umieszczona w geometrii wielokrotnie
 - Jedna objętość fizyczna (G4PhysicalVolume) reprezentuje dowolną liczbę „rzeczywistych” brył.
 - mniejsze użycie pamięci, szybsza nawigacja.
 - Rodzaje „repeated volumes”:
 - „Replica” ([G4PVReplica](#)) i „Division” ([G4PVDivision](#))
proste powielanie lub podział wzdłuż wybranej osi (również w układzie cylindrycznym/sferycznym),
 - „Parameterised” ([G4PVParameterised](#))
powielanie bryły w „sprytny” sposób, tzn. kształt, rozmiar, materiał, czułość (sensitivity), pozycja, rotacja, atrybuty wizualizacji są obliczane w zadany sposób (parametryzacja) na podstawie numeru kopii,
 - [G4ReflectionFactory](#) – bryła i jej odbicie,
 - [G4AssemblyVolume](#) – zbiór wielu „Placement volumes”.
- Jedna bryła „matka” może w sobie zawierać albo
 - dowolnie wiele brył typu „placement volume”,
 - albo, jedną bryłę typu „repeated volume”.



placement



repeated



Geometria – dwa sposoby pozycjonowania brył

Rotacja względem środka lokalnego układu współrzędnych (córki)

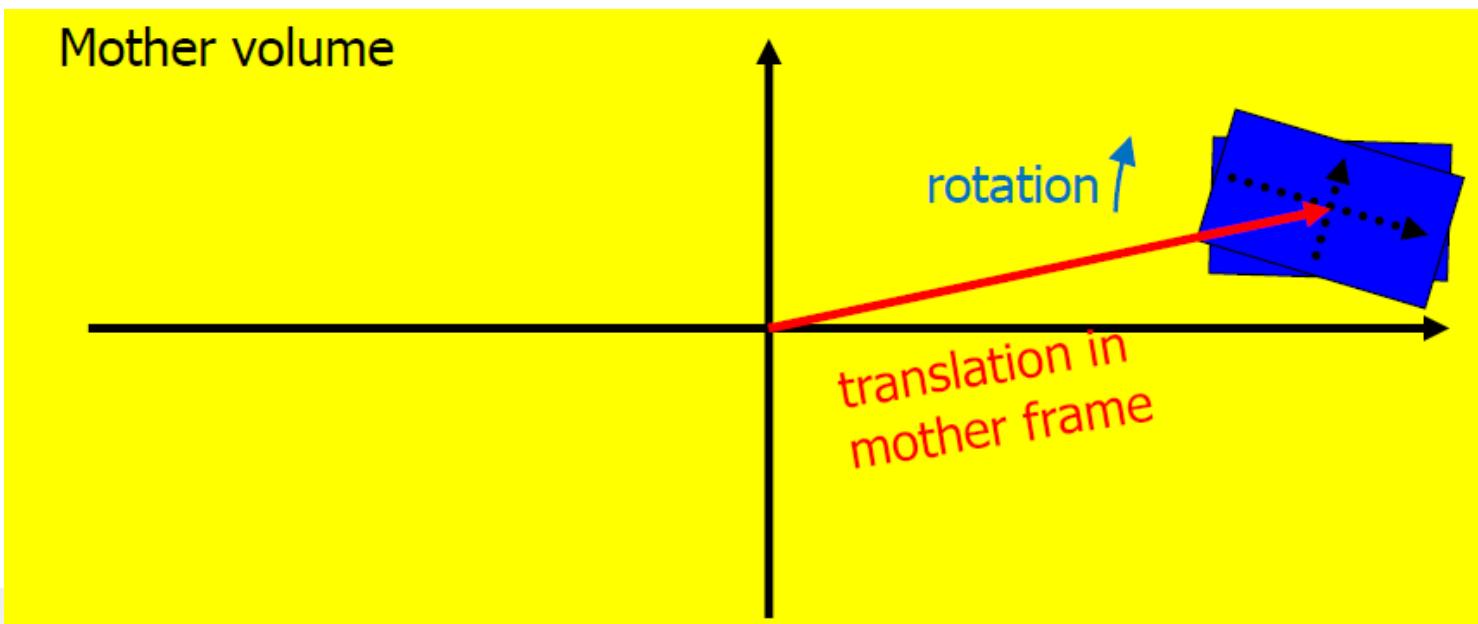
+

Translacja względem środka globalnego układu współrzędnych (matki)

Sposób najbardziej intuicyjny

G4PVPlacement(

```
G4Transform3D(G4RotationMatrix &pRot, // rotation of daughter volume
const G4ThreeVector &tlate), // position in mother frame
G4LogicalVolume *pDaughterLogical,
const G4String &pName,
G4LogicalVolume *pMotherLogical,
G4bool pMany, // not supported/for future use...
G4int pCopyNo, // unique arbitrary integer
G4bool pSurfChk=false); // optional boundary check
```



Geometria – dwa sposoby pozycjonowania brył

Rotacja globalnego układu współrzędnych (matki)

+

Translacja względem środka globalnego układu współrzędnych (matki)

```
G4PVPlacement(G4RotationMatrix* pRot, // rotation of mother frame  
               const G4ThreeVector &tlate, // position in mother frame  
               G4LogicalVolume *pDaughterLogical,  
               const G4String &pName,  
               G4LogicalVolume *pMotherLogical,  
               G4bool pMany, // not supported/for future use...  
               G4int pCopyNo, // unique arbitrary integer  
               G4bool pSurfChk=false); // optional boundary check
```

- *Mniej intuicyjna metoda*
- *Bez obrotu nie ma różnicy względem poprzedniej*
- *Zaleta (chyba jedyna): przekazywany wskaźnik do macierzy obrotu a nie sama macierz -> można zmienić macierz bez dostępu do bryły.*

Dziękuję za uwagę



NARODOWE
CENTRUM
BADAŃ
JĄDROWYCH
ŚWIERK

www.ncbj.gov.pl

