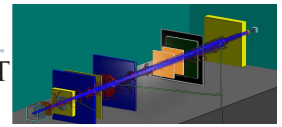


XIII International GEANT4 SCHOOL

Materials & Geometry

Describing your detector in Geant4 — units, materials and the volume hierarchy

Alberto Sciuto

INFN – Laboratori Nazionali del Sud

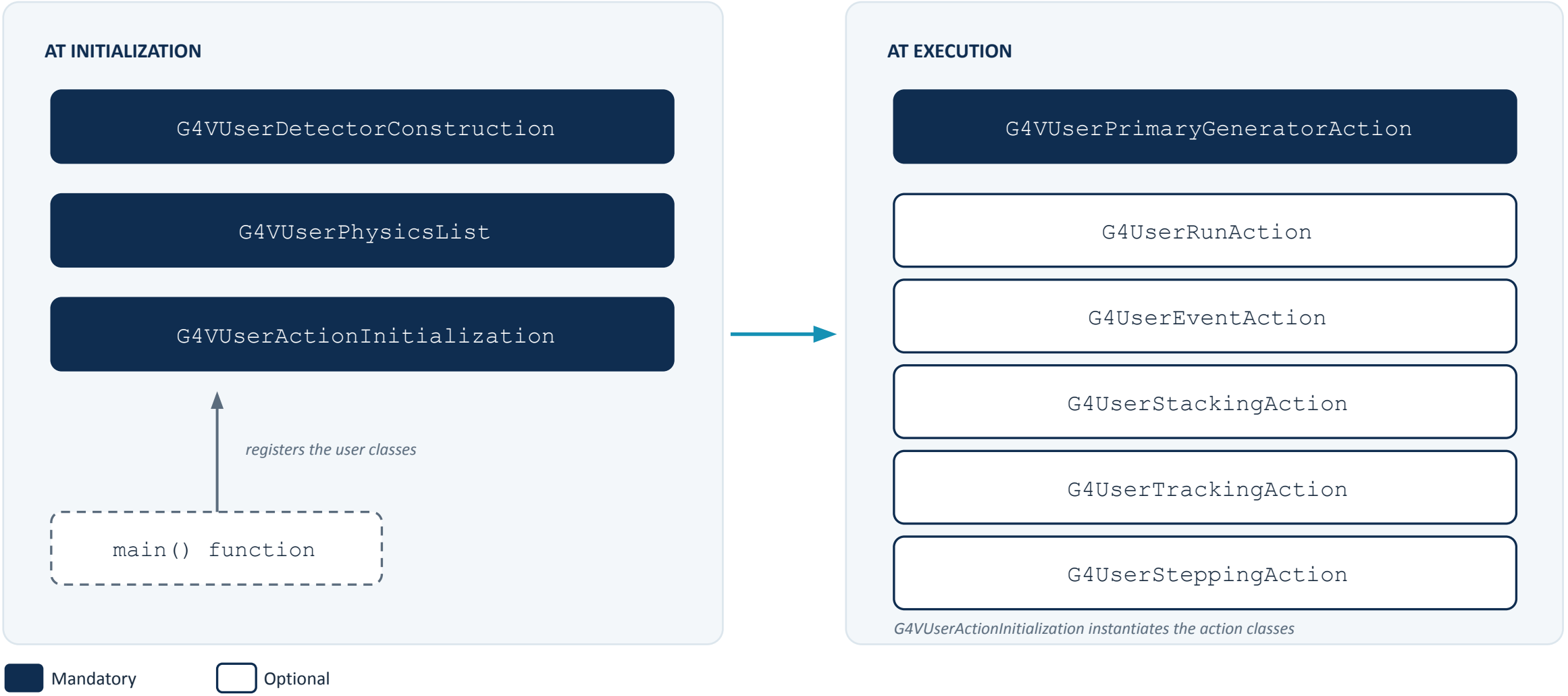
alberto.sciuto@lns.infn.it

A lot of material by J. Pipek

Original slides by G.A.P. Cirrone and Luciano Pandola

27-31 July 2026 ELI Beamlines Facility Czech Republic

Mandatory (and optional) user classes



The Detector Construction

The user class that describes the geometry **must inherit** from `G4VUserDetectorConstruction` …

… and must be **registered** in the Run Manager

- It defines the geometry of your model:

All **materials**

All **volumes** and their placements

– (Optionally) add fields

(Optionally) define volumes for read-out — **sensitive detectors**

THE BASE CLASS

```
class G4VUserDetectorConstruction
{
  public:
    G4VUserDetectorConstruction();
    virtual ~G4VUserDetectorConstruction();

    virtual G4VPhysicalVolume* Construct() = 0;
    virtual void ConstructSDandField();
    // ...
};
```

i `Construct()` is **pure virtual**: every application has to implement it and return the world physical volume.

PART I

Units

How Geant4 talks about physical quantities

01

Geant4 basic types

Aliases for the primitive data types provide **cross-platform compatibility**:

`G4double` `G4float` `G4int` `G4bool` `G4long`

An enhanced version of string, `G4String`
inherits from `std::string` ⇒ all its methods and operators
– plus several additional methods

`G4ThreeVector` — a three-component class corresponding to a real physics vector (examples later)

```
G4ThreeVector dimensions {1.0, 2.0, 3.0};
```

i Please use these types for best compatibility — e.g. `G4int` instead of `int`, `G4ThreeVector` when it makes sense, etc.

Units in Geant4

Don't use default units!

When specifying dimensions, **always multiply** by an appropriate unit:

```
G4double width   = 12.5 * m;
G4double density = 2.7 * g/cm3;
```

Most common units are defined in the **CLHEP** library (included in Geant4):

`G4SystemOfUnits.hh`

`CLHEP/SystemOfUnits.hh`

… and you can define new units (next slides)

Output data in terms of a specific unit: divide the value by the unit —

```
G4cout << dE / MeV << " (MeV)" << G4endl;
```

WHY IT MATTERS

- Internal units are hidden from user code — your numbers stay unambiguous
- The same code works unchanged if the internal system of units changes
- Multiplying on input and dividing on output is all you ever need

The system of units in Geant4

`mm` millimeter	**`ns`** nanosecond	**`MeV`** megaelectronvolt	**`eplus`** unit charge
`K` kelvin	**`cd`** candela	**`rad`** radian	**`sr`** steradian

All other units are derived from the basic ones.

Useful feature — Geant4 can select the most appropriate unit for you: just specify the category of the data (`Length`, `Time`, `Energy`, …)

```
G4cout << G4BestUnit(StepSize, "Length");
```

i `StepSize` will be printed in km, m, mm or … fermi, depending on its actual value.

Defining new units

New units can be defined directly as constants or — **suggested way** — via `G4UnitDefinition`:

```
G4UnitDefinition ( "name", "symbol", "category", value )
G4UnitDefinition ( "grammpercm2", "g/cm2", "MassThickness", g/cm2 );
```

The new category `MassThickness` is then registered in the kernel in `G4UnitsTable`

To print the list of units:

from the code

```
G4UnitDefinition::PrintUnitsTable();
```

at run-time (UI command)

```
Idle> /units/list
```

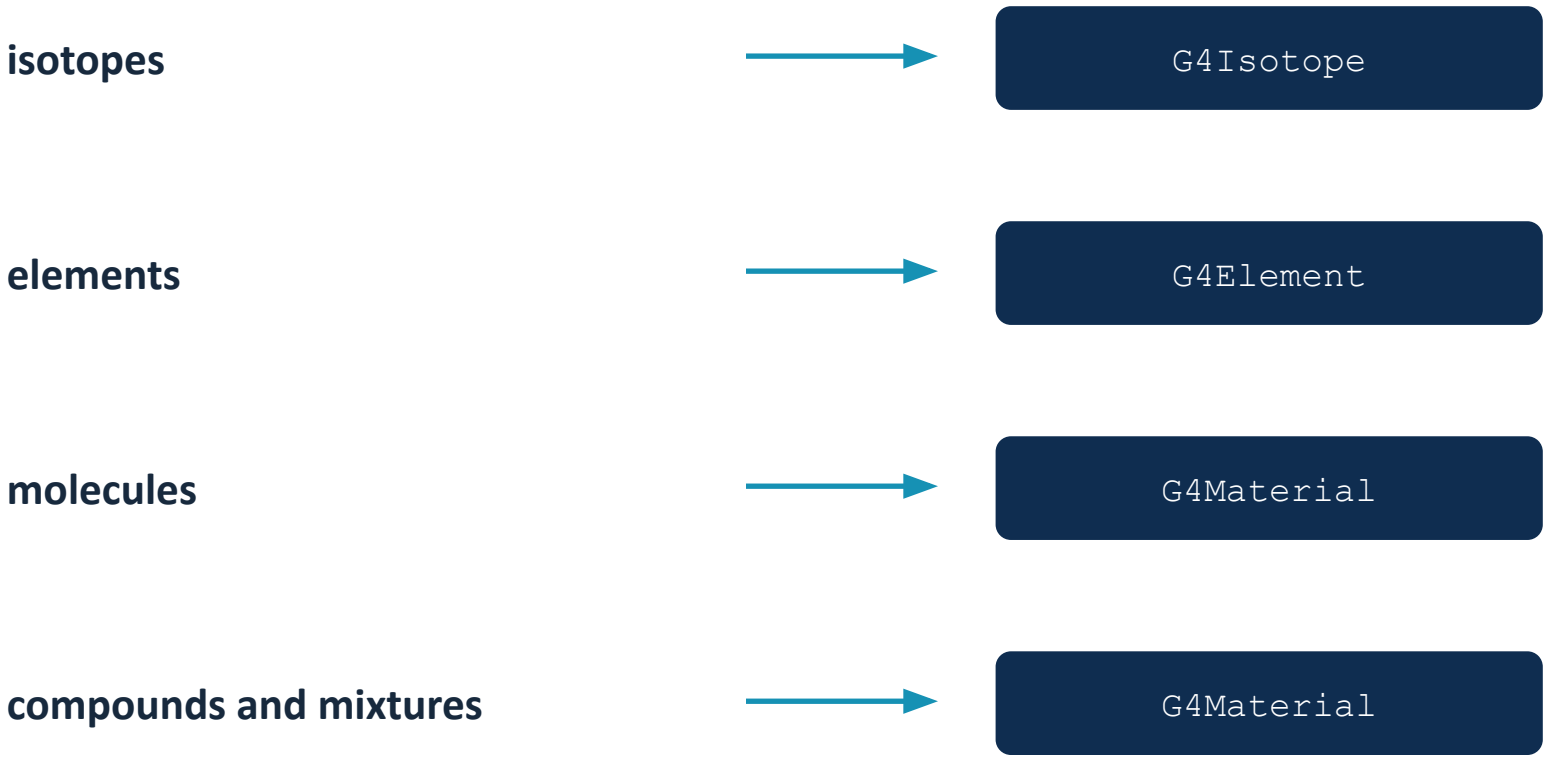
PART II

Materials

From isotopes to mixtures

02

Different levels of material description



ATTRIBUTES

Density (mandatory)

- Temperature
- Pressure
- State (gas, liquid, …)

Atoms vs. matter

G4ISOTOPE · G4ELEMENT

properties of the atoms

- Atomic number, number of nucleons, mass of a mole
- Shell energies
- Cross-sections per atom, etc.

G4MATERIAL

macroscopic properties of the matter

- Temperature, pressure, state, density
- Radiation length, absorption length, etc.

Used by **tracking, geometry and physics** in Geant4

i Material properties are computed from the elemental properties — assumption of a **linear combination**.

Elements and isotopes

Need an element with a **non-natural isotopic composition** (e.g. enriched Ge)?

1. Build the isotopes …

2. … and assemble them into elements

ISOTOPE

```
G4Isotope(const G4String& name,
                G4int     z,    // atomic number
                G4int     n,    // number of nucleons
                G4double  a );  // mass of mole
```

ELEMENT

```
G4Element(const G4String& name,
          const G4String& symbol,  // element symbol
                G4int     nIso );  // n. of isotopes
```

ASSEMBLE

```
G4Element::AddIsotope(G4Isotope* iso, // isotope
      G4double relAbund);  // fraction of nuclei
```

… for instance: enriched uranium

Build enriched U — isotopes first, then the element:

```
G4Isotope* U5 = new G4Isotope(name="U235", iz=92, n=235, a=235.01*g/mole);
G4Isotope* U8 = new G4Isotope(name="U238", iz=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);
```

For an element with **natural isotopic composition** the definition is easier:

```
a = 16.00*g/mole;
G4Element* elO = new G4Element("Oxygen", symbol="O", z=8., a);
G4cout << elO << G4endl;  // printout of element info
```

Elements and molecules

Single-element materials:

```
G4double z, a, density;
density = 1.390*g/cm3;
a = 39.95*g/mole;
G4Material* lAr =
    new G4Material("liquidAr", z=18, a, density);
```

Molecules — composition by number of atoms:

```
a = 1.01*g/mole;
G4Element* elH =
    new G4Element("Hydrogen", symbol="H", z=1., a);
a = 16.00*g/mole;
G4Element* elO =
    new G4Element("Oxygen", symbol="O", z=8., a);

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

i Elements are the building blocks: define them once, reuse them in every material.

Materials: compounds

Composition by **fraction of mass**:

```
a = 14.01*g/mole;
G4Element* elN = new G4Element(name="Nitrogen", symbol="N", z=7., a);
a = 16.00*g/mole;
G4Element* elO = new G4Element(name="Oxygen",   symbol="O", z=8., a);

density = 1.290*mg/cm3;
G4Material* Air = new G4Material(name="Air", density, ncomponents=2);
Air->AddElement(elN, 70.0*perCent);
Air->AddElement(elO, 30.0*perCent);
```

i `AddElement` with `perCent` fractions — they must add up to 100 %.

Materials: mixtures

Materials can also be combined — **composition of mixtures**:

```
G4Element*  elC  = ...;  // define "carbon" element
G4Material* SiO2 = ...;  // define "quartz" material
G4Material* H2O  = ...;  // define "water"  material

density = 0.200*g/cm3;
G4Material* aerogel = new G4Material("Aerogel", density, ncomponents=3);
aerogel->AddMaterial(SiO2, fractionmass=62.5*perCent);
aerogel->AddMaterial(H2O,  fractionmass=37.4*perCent);
aerogel->AddElement (elC,  fractionmass= 0.1*perCent);
```

i A mixture can combine **elements and full materials** in the same definition.

Example: a gas

It may be necessary to specify **temperature** and **pressure**
The **dE/dx** computation is affected

```
G4double density     = 27.  * mg/cm3;
G4double temperature = 325. * kelvin;
G4double pressure    = 50.  * atmosphere;

G4Material* CO2 = new G4Material("CO2Gas", density,
    ncomponents=2, kStateGas, temperature, pressure);
CO2->AddElement(C, natoms = 1);
CO2->AddElement(O, natoms = 2);
```

`kStateGas` *state flag passed to the constructor*

“Vacuum”

> **Absolute vacuum does not exist** — model it as a gas at very low density!

You **cannot** define materials composed of multiple elements through Z or A, or with ρ = 0

- A single-element, ultra-low-density gas does the job

```
G4double atomicNumber = 1.;
G4double massOfMole   = 1.008*g/mole;
G4double density      = 1.e-25*g/cm3;
G4double temperature  = 2.73*kelvin;
G4double pressure     = 3.e-18*pascal;

G4Material* Vacuum = new G4Material(
    "interGalactic", atomicNumber, massOfMole,
    density, kStateGas, temperature, pressure);
```

The NIST material database

All elements and many commonly-used materials are available in Geant4 through the **NIST database**
No need to predefine elements and materials — retrieve them from the NIST manager:

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

G4Material* H2O    = manager->FindOrBuildMaterial("G4_WATER");
G4Material* vacuum = manager->FindOrBuildMaterial("G4_Galactic");
```

UI commands:

`/material/nist/printElement`	print defined elements
`/material/nist/listMaterials`	print defined materials

The NIST database: elements

The NIST database for elements and materials is **imported in Geant4**
- physics.nist.gov/PhysRefData

- UI commands specific for handling materials

The **best accuracy** for the most relevant parameters is guaranteed:
- Density
- Mean excitation potential
- Element and isotope composition
- Various corrections

Natural isotope compositions, more than **3000 isotope masses**
- Full list: Geant4 Application Developer Guide, appendix “NIST materials”

Z		A	m	error	(%)	A_{eff}
14	Si	22	22.03453	(22)		28.0855(3)
		23	23.02552	(21)		
		24	24.011546	(21)		
		25	25.004107	(11)		
		26	25.992330	(3)		
		27	26.98670476	(17)		
		28	27.9769265327	(20)	92.2297 (7)	
		29	28.97649472	(3)	4.6832 (5)	
		30	29.97377022	(5)	3.0872 (5)	
		31	30.97536327	(7)		
		32	31.9741481	(23)		
		33	32.978001	(17)		
		34	33.978576	(15)		
		35	34.984580	(40)		
		36	35.98669	(11)		
		37	36.99300	(13)		
		38	37.99598	(29)		
		39	39.00230	(43)		
		40	40.00580	(54)		
		41	41.01270	(64)		
		42	42.01610	(75)		

NIST materials

NIST elements: H ⟶ Cf (Z = 1 ⟶ 98)
NIST compounds: e.g. `G4_ADIPOSE_TISSUE_ICRP`
HEP and nuclear materials: e.g. liquid Ar, PbWO4
Possible to build **mixtures of NIST and user-defined** materials

```
=====================================
###  Compound Materials from the NIST Data Base
=====================================
 N Name     ChFormula      density(g/cm^3)  I(eV)
=====================================
13  G4_Adipose_Tissue            0.92          63.2
          1    0.119477
          6    0.63724
          7    0.00797
          8    0.232333
         11    0.0005
         12    2e-05
         15    0.00016
         16    0.00073
         17    0.00119
         19    0.00032
         20    2e-05
         26    2e-05
         30    2e-05
4  G4_Air                       0.00120479    85.7
          6    0.000124
          7    0.755268
          8    0.231781
         18    0.012827
2  G4_CsI                       4.51          553.1
         53    0.47692
         55    0.52308
```

```
=====================================
###  Elementary Materials from the NIST Data
=====================================
 Z Name  ChFormula      density(g/cm^3)  I(eV)
=====================================
1   G4_H   H_2            8.3748e-05     19.2
2   G4_He                  0.000166322    41.8
3   G4_Li                  0.534          40
4   G4_Be                  1.848          63.7
5   G4_B                   2.37           76
6   G4_C                   2              81
7   G4_N   N_2             0.0011652      82
8   G4_O   O_2             0.00133151     95
9   G4_F                   0.00158029     115
10  G4_Ne                  0.000838505    137
11  G4_Na                  0.971          149
```

`/material/nist/listMaterials` — output

PART III

Geometry

Solids, logical and physical volumes

03

Describe your detector

A detector geometry is made of a number of **volumes**
The largest volume is the **World volume** — it must contain all other volumes
Derive your own concrete class from the `G4VUserDetectorConstruction` abstract base class
Implement the virtual methods `Construct()` (**pure virtual**) and `ConstructSDandField()`

INSIDE CONSTRUCT()

- Define the shapes / solids required to describe the geometry
- Construct all necessary materials
- Construct and place the volumes of your detector geometry
- *Define "sensitivity" properties associated to volumes*
- *Associate magnetic fields to detector regions*
- *Define visualization attributes for the detector elements*

in italics: optional

Geometry: implementation basics

Implement a class inheriting from `G4VUserDetectorConstruction`:

```
class MyDetector : public G4VUserDetectorConstruction {
public:
    virtual G4VPhysicalVolume* Construct();  // required

    virtual void ConstructSDandField();      // optional
    // ...
};
```

Create an instance **in the main program**:

```
MyDetector* detector = new MyDetector();
runManager->SetUserInitialization(detector);
```

i **Split the implementation** into more classes and methods — good programming practice.

! Do **not** delete the `MyDetector` instance — the Run Manager does that automatically.

G4VUserDetectorConstruction

`G4VUserDetectorConstruction.hh`

CONSTRUCT()

- Define materials
- Define solids and volumes of the geometry
- Build the tree hierarchy of volumes
- Define visualization attributes
- **Return the world physical volume!**

CONSTRUCTSDANDFIELD()

MT

- Assign magnetic fields to volumes / regions
- Define sensitive detectors and assign them to volumes

called per worker thread in multi-threaded mode

Three conceptual layers

`G4VSolid`

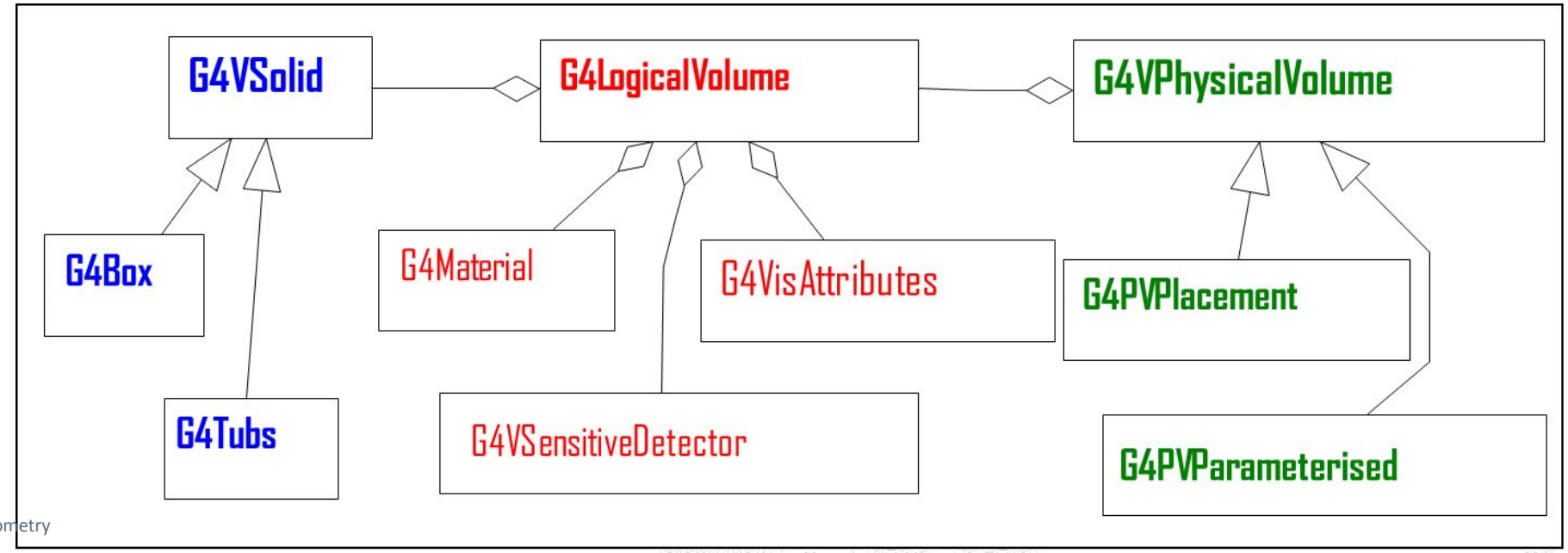
Shape and size

`G4LogicalVolume`

Hierarchy of volumes, material, sensitivity, magnetic field

`G4VPhysicalVolume`

Position and rotation. The same logical volume can be placed many times (repeated modules)



Define the detector geometry — basic strategy

1. Create the geometry object — the **solid**: shape and size

2. Assign properties to the object — the **material**

3. Place it in the coordinate system of the **mother volume**

SOLID → LOGICAL → PHYSICAL

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

G4LogicalVolume* pBoxLog =
    new G4LogicalVolume(pBoxSolid, pBoxMaterial,
                        "aBoxLog", 0, 0, 0);

G4VPhysicalVolume* pBoxPhys =
    new G4PVPlacement(pRotation,
        G4ThreeVector(posX, posY, posZ),
        pBoxLog, "aBoxPhys", pMotherLog,
        0, copyNo);
```

Solids

- **CSG (Constructed Solid Geometry) solids**

`G4Box`, `G4Tubs`, `G4Cons`, `G4Trd`, ...
  – analogous to the simple GEANT3 CSG solids

- **Specific solids (CSG-like)**

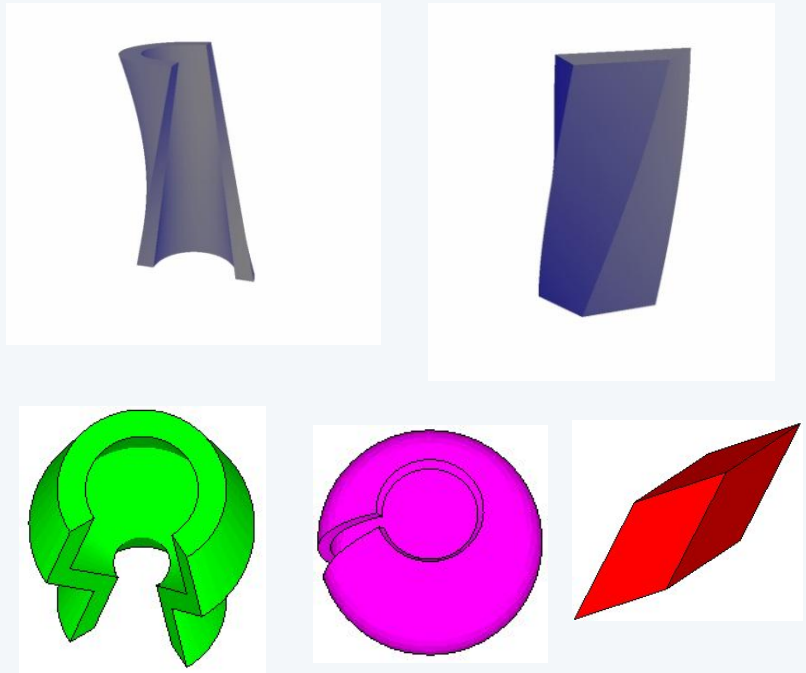
`G4Polycone`, `G4Polyhedra`, `G4Hype`, ...
`G4TwistedTubs`, `G4TwistedTrap`, ...

- **BREP (Boundary REPresented) solids**

`G4BREPSolidPolycone`, `G4BSplineSurface`, ...
  – any-order surfaces

- **Boolean solids**

`G4UnionSolid`, `G4SubtractionSolid`, ...

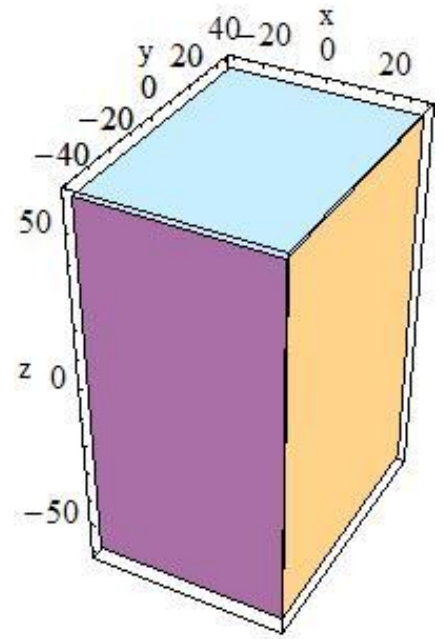


a few of the available shapes

CSG: G4Box

```
G4Box(const G4String& pname, // name
            G4double  pX,    // half-length in X
            G4double  pY,    // half-length in Y
            G4double  pZ );  // half-length in Z
```

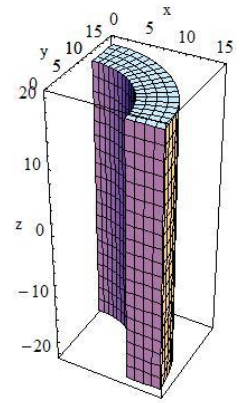
> **Note the half-length!** A `G4Box(…, 1.*m, …)` is 2 m across in X.



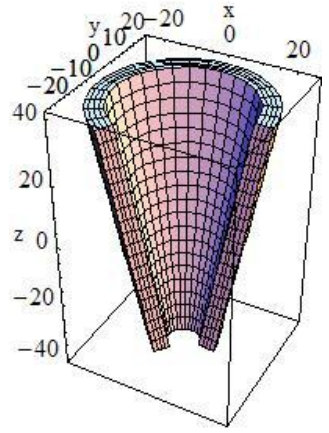
pX, pY, pZ = half-lengths

CSG: G4Tubs & G4Cons

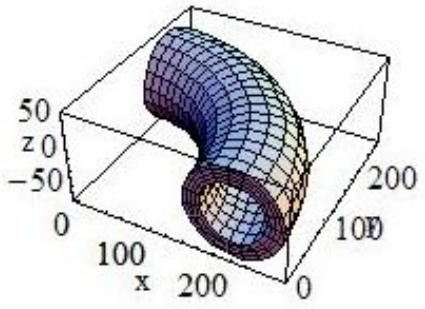
```
G4Tubs(const G4String& pname, // name
             G4double  pRmin,  // inner radius (0)
             G4double  pRmax,  // outer radius
             G4double  pDz,    // Z half-length!
             G4double  pSphi,  // starting Phi (0)
             G4double  pDphi); // segment angle (twopi)
```



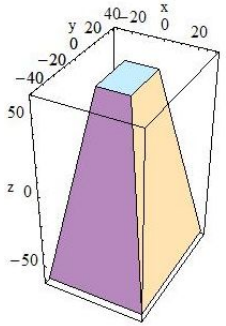

```
G4Cons(const G4String& pname, // name
             G4double  pRmin1, // inner radius at -pDz
             G4double  pRmax1, // outer radius at -pDz
             G4double  pRmin2, // inner radius at +pDz
             G4double  pRmax2, // outer radius at +pDz
             G4double  pDz,    // Z half-length
             G4double  pSphi,  // starting Phi
             G4double  pDphi); // segment angle
```



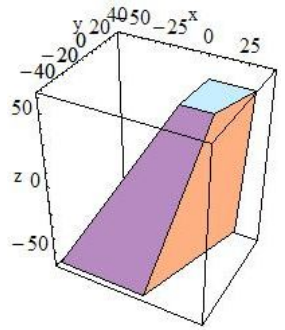
Other CSG solids



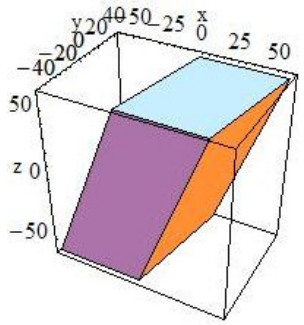
G4Torus



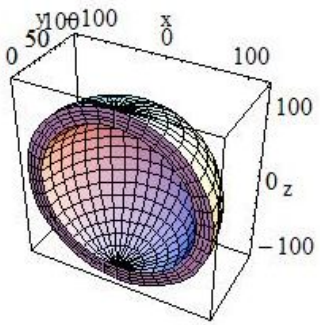
G4Trd



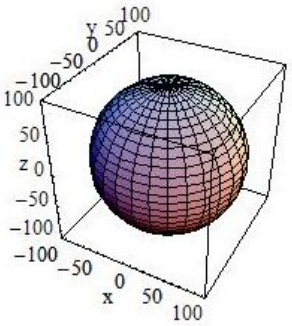
G4Trap



G4Para · parallelepiped



G4Sphere



G4Orb · full solid sphere

Check Section 4.1.2 of the Geant4 Application Developer Guide for all available shapes.

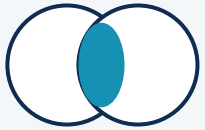
Boolean solids

Solids can be combined using **boolean operations**:
`G4UnionSolid`, `G4SubtractionSolid`, `G4IntersectionSolid`
Requires: **2 solids, 1 boolean operation**, and an (optional) transformation for the 2nd solid
The 2nd solid is positioned **relative to the coordinate system of the 1st** solid
The result becomes a solid itself — **re-usable** in a further boolean operation

- The solids to be combined can be either CSG or other Boolean solids

`G4UnionSolid`

`G4SubtractionSolid`



`G4IntersectionSolid`

! Tracking cost for navigating a complex Boolean solid is proportional to the number of constituent CSG solids.

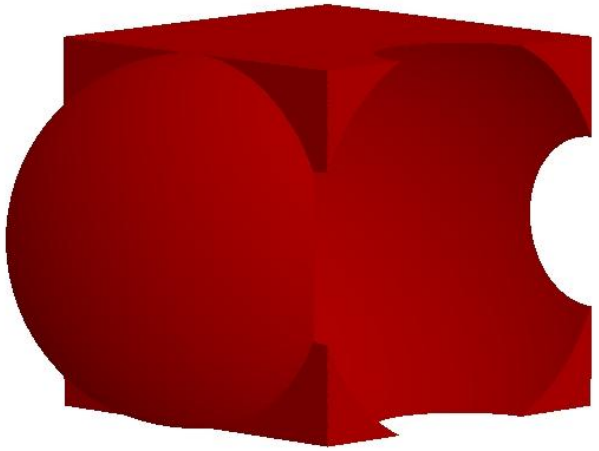
Boolean solids — an example

```
G4VSolid* box = new G4Box("Box", 50*cm, 60*cm, 40*cm);
G4VSolid* cylinder =
    new G4Tubs("Cylinder", 0., 50.*cm, 50.*cm, 0., twopi);

G4VSolid* unionSolid =
    new G4UnionSolid("Box+Cylinder", box, cylinder);

G4VSolid* subtract =
    new G4SubtractionSolid("Box-Cylinder", box, cylinder,
        0, G4ThreeVector(30.*cm, 0., 0.));

G4RotationMatrix* rm = new G4RotationMatrix();
rm->RotateX(30.*deg);
G4VSolid* intersect =
    new G4IntersectionSolid("Box&&Cylinder",
        box, cylinder, rm, G4ThreeVector(0.,0.,0.));
```



visualisation of a resulting solid

Logical volumes

Contain **all information of the volume except position**:
Shape and dimension (`G4VSolid`)

- Material, sensitivity, visualization attributes
- Hierarchy of daughter volumes
- Magnetic field, user limits

Physical volumes of the same type can **share** a logical volume
The pointers to solid and material must **not** be `nullptr`

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

optional }

one logical volume → many placed copies

Physical volumes

A physical volume is a **positioned instance** of a logical volume inside another logical volume (its **mother volume**)

Placement (`G4PVPlacement`)
- one positioned volume

Repeated: a volume placed many times
- can represent any number of volumes — reduces memory use

`G4PVReplica` — simple repetition

`G4PVParameterised` — more complex patterns
- `G4PVDivision`

placement — one positioned volume

repeated — same logical volume, n copies

Geometry hierarchy: mothers and daughters

A volume is placed **in its mother volume**
Position and rotation of the daughter are described **with respect to the local coordinate system of the mother**
The origin of the mother's local coordinate system is at the **center of the mother volume**
Daughter volumes **cannot protrude** from the mother volume
Daughter volumes **cannot overlap**
The logical volume of the mother knows the daughter volumes it contains — it is **uniquely defined** to be their mother volume

Geometry hierarchy: the world volume

One logical volume can be **placed more than once**; one or more volumes can be placed in a mother volume
The mother–daughter relationship is an information of `G4LogicalVolume`
  – if the mother is placed more than once, all daughters by definition appear in each placed physical volume

The **World volume** must be a unique physical volume which fully contains all the other volumes — the **root** of the hierarchy
The World volume defines the **global coordinate system**; its origin is at the center of the world
The position of a track is given **with respect to the global coordinate system**

G4PVPlacement

A single volume positioned **relatively to the mother volume** — in a frame rotated and translated relative to the mother's coordinate system

- A few variants are available:

using `G4Transform3D` to represent the **direct rotation and translation of the solid** instead of the frame (alternative constructor)
specifying the mother volume as a pointer to its **physical** volume instead of its logical volume

FOUR CONSTRUCTORS AVAILABLE

logical OR physical volume
as the mother volume

active OR passive transformation
of the coordinate system

Rotation of vs. in the mother frame

Passive transformation — rotation **of the mother frame**:

```
G4PVPlacement(G4RotationMatrix* pRot,       // rotation of mother frame
        const G4ThreeVector& tlate,        // position in mother frame
        G4LogicalVolume* pCurrentLogical,
        const G4String& pName,
        G4LogicalVolume* pMotherLogical,
        G4bool pMany,                      // not used, set it to false
        G4int  pCopyNo,                    // unique arbitrary index
        G4bool pSurfChk = false);          // optional overlap check
```

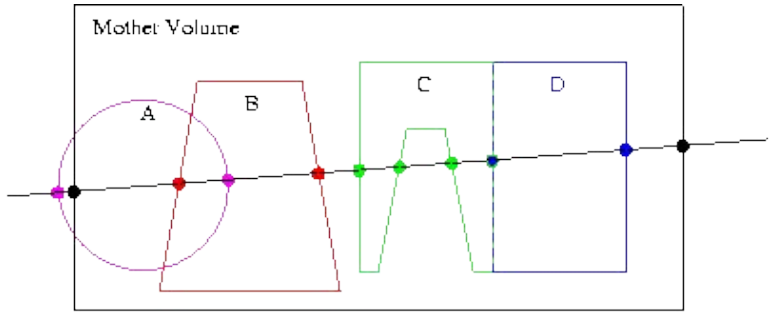
Active transformation — rotation **in the mother frame** (via `G4Transform3D`):

```
G4PVPlacement(G4Transform3D(
            G4RotationMatrix& pRot,        // rotation in daughter frame
            const G4ThreeVector& tlate),   // position in mother frame
        G4LogicalVolume* pDaughterLogical,
        const G4String& pName,
        G4LogicalVolume* pMotherLogical,
        G4bool pMany, G4int pCopyNo,
        G4bool pSurfChk = false);          // optional overlap check
```

Geometry problems

Geant4 does **not** allow malformed geometries — neither **protruding** (daughter / mother) nor **overlapping** (sisters)
The behavior of navigation is **unpredictable** in such cases

- Detecting overlaps is bounded by the complexity of the solid model description
- Utilities are provided for detecting wrong positioning:
  - optional checks at construction
  - kernel run-time commands
  - graphical tools (DAVID)



protruding **overlapping**

Tools for geometry check

Constructors of `G4PVPlacement` and `G4PVParameterised` have an optional argument `pSurfChk`
If this flag is **true**, an overlap check is done at construction:
- points are randomly sampled on the surface of the volume being created

CPU-expensive — but **worth trying at least once**

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

Built-in run-time commands activate verification tests for the user geometry:

`/geometry/test/run` *or* `/geometry/test/grid_test`

start verification of the geometry for overlapping regions, based on a standard grid setup — limited to the first depth level

`/geometry/test/recursive_test` the same test on all depth levels (CPU intensive!)

Regions

A region is a **sub-set of the geometry**

- It may have its own specific:

production thresholds (cuts)
user limits — artificial limits affecting tracking, e.g. max step length, max number of steps, min kinetic energy left, …
  – field manager

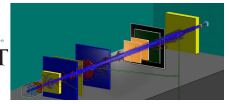
i The **World logical volume** is recognized as the default region.

! The user is **not allowed** to define a region for the World logical volume.

AND NOW

Hands-on...

Materials & geometry in practice



Alberto Sciuto · alberto.sciuto@lns.infn.it