

# Detector construction

Mihaly Novak (CERN, EP-SFT)

Getting Started with **Geant4** at CERN, Geneva (Switzerland)

- Material definition
  - The **Geant4** material model
  - Material definition
  - The **NIST** material data base
- Geometry definition
  - The **Geant4** geometry description
  - Solid, Logical - , Physical - volume

Material definition

# **THE GEANT4 MATERIAL MODEL**

- The **Geant4 material** model follows the natural one: materials are made of **elements** and elements are made of **isotopes**
- The 3 main classes to describe these objects are
  - **G4Isotope**: describes the properties of atoms ( $Z$  - atomic number,  $N$  - number of nucleons and  $A$  - molar mass) with unique `name` and `index`
  - **G4Element**: describes the properties of elements ( $Z$  - effective atomic number,  $N$  - effective number of nucleons and  $A$  - effective molar mass, number of isotopes, etc.) with unique `name`, `symbol` and `index`
  - **G4Material**: describes the macroscopic properties of matter (**density, state, temperature, pressure**, etc.) with unique `name` and `index`
- Unique index: a pointer to the created object is automatically stored in global table (isotope, element and material tables)

- The **material density** must be set ( $> \text{zero}$ ) by the user at definition (except NIST materials) !
- The **material temperature** and **pressure** can optionally be set:
  - default: Normal Temperature and Pressure(NTP) 293.15 [K], 1 [atm] = 101.325 [kPascal]
- The **material state** can be **solid**, **liquid** or **gas**:
  - default is either solid or gas depending on the density (`kGasThreshold = 10 [mg/cm3]` )
  - non-crystalline (i.e. amorphous) solid by default (special extension for incorporating some information on the crystal structure)
- Special set of pre-defined materials: **NIST** material composition data base with some frequently used HEP materials
- **Geant4 material documentation:** [Material Documentation](#)

Material definition

# **MATERIAL DEFINITION**

- Elements and isotopes:

- **G4Element** object without specifying the isotope composition:

```
// simple way of Carbon element definition
G4Element* elC = new G4Element(name="Carbon", symbol="C", z = 6., a = 12.01*g/mole);
```

- need to give: name, symbol, Z and A (effective atomic number and molar mass)
- isotopes will be automatically added with natural abundances (A won't be updated)

- **G4Element** object by specific (non-natural) isotope composition:

```
// Define "enriched uranium" element as 90 % of U235 and 10 % of U 238:
//
// create the isotopes: iz = number of protons and n = number of nucleons
G4Isotope* U5 = new G4Isotope(name="U235", iz=92, n=235);
G4Isotope* U8 = new G4Isotope(name="U238", iz=92, n=238);
// create the element and build up by adding the isotopes with their abundance
G4Element* elU=new G4Element(name="enriched uranium",symbol="U",numisotopes=2);
elU->AddIsotope(U5, abundance= 90.*perCent);
elU->AddIsotope(U8, abundance= 10.*perCent);
```

- element object must be created: name, symbol, number of isotopes
- isotope objects must be created: name, number of protons and nucleons
- isotopes need to be added by their relative abundance

- Simple **G4Material** object definition:
  - “simple”: the material contains only one element and the corresponding **G4Element** object is not provided:

```
// single element "Uranium" material without giving the uranium element object
G4Material* matU = new G4Material(name    = "Uranium",
                                   _z      = 92.0,
                                   _a      = 238.03 * g/mole,
                                   density= 18.950 * g/cm3);
```

- the corresponding **G4Element** object will be automatically created (with natural isotope abundance)
- need to give: `name`, `density` of the material, `Z` and `A` (effective atomic number and molar mass) of the single **G4Element**
- what happens if we want the single element to have non-natural isotope abundance e.g. the previously created `enriched uranium` (see later)



- **G4Material** object definition as chemical molecule:
  - molecules build up from (several) elements with composition specified by the number of element (e.g. water = H<sub>2</sub>O)
  - accordingly, **G4Material** object can be created by adding **G4Element** objects to it together with their composition number:

```
// Create water material as molecule based on its chemical formula (H2O)
//
// create the necessary H and O elements (natural isotope abundance):
G4Element* elH = new G4Element(name = "Hydrogen",
                                symbol = "H",
                                z = 1.00,
                                a = 1.01 * g/mole);
G4Element* elO = new G4Element(name = "Oxygen",
                                symbol = "O",
                                z = 8.00,
                                a = 16.00 * g/mole);
// create the water material (name, density, number of components):
G4Material* matH2O = new G4Material(name = "Water",
                                    density = 1.0 * g/cm3,
                                    ncomponents = 2);
// add the elements to the material with their composition number
matH2O->AddElement(elH, numberOfatoms = 2);
matH2O->AddElement(elO, numberOfatoms = 1);
```

- **G4Material** object definition as mixture:
  - mixture of elements (**G4Element**), mixture of other materials (**G4Material**) or even mixture of elements and materials
  - similar to molecules with the differences:
    - components can be other materials not only elements
    - the ratio of the components must be given as “**fractional mass**” not as “**number of atoms**”
  - **mixture of elements** example: using the `AddElement` method

```
// Create air material as 70-30 % mixture of N and O elements
// (assuming that N and O Geant4 elements have already been created as
//  elN and elO)
//
// create the air material (name, density, number of components):
G4Material* matAir = new G4Material(name      = "Air",
                                   density     = 1.290 * mg/cm2,
                                   ncomponents = 2);
// add the elements to the material with their fractional mass
matAir->AddElement(elN, fractionmass = 0.7);
matAir->AddElement(elO, fractionmass = 0.3);
```

- **G4Material** object definition as mixture:
  - **mixture of element(s) and material(s)** example: using the `AddElement` and `AddMaterial` methods

```
// Create aerogel material as 62.5 % silicon dioxide (SiO2), 37.4 % water (H2O)
// materials and 0.1 % carbon element. Assuming that the materials (matSiO2 and
// matH2O) as well as the carbon element (elC) have already been created.
//
// create the aerogel material (name, density, number of components):
G4Material* matAerog = new G4Material(name      = "Aerogel",
                                     density     = 0.2 * g/cm2,
                                     ncomponents = 3);
// add the elements to the material with their fractional mass
matAerog->AddMaterial(matSiO2, fractionmass = 62.5 * perCent);
matAerog->AddMaterial(matH2O , fractionmass = 37.4 * perCent);
matAerog->AddElement (elC      , fractionmass = 0.1 * perCent);
```

Material definition

# **THE NIST MATERIAL DATA BASE**

- The data base includes more than 3000 **isotopes**
- Isotopic composition of **elements** ( $Z = [1-108]$ ) with their **natural isotopic abundance**: using the **NIST Atomic Weights and Isotopic Compositions** data base
- **NIST elements** can be obtained easily from the **Geant4 NIST** data base **by** using **their** `symbol` or `Z - atomic number`:

- the corresponding **G4Isotope** objects will be automatically built
- “find or build” i.e. avoids duplication of element objects

```
// get the carbon G4Element object from the NIST data base: by its symbol
G4Element* elC = G4NistManager::Instance()->FindOrBuildElement("C");
// get the silicon G4Element object from the NIST data base: by its Z
G4Element* elSi = G4NistManager::Instance()->FindOrBuildElement(14);
```

- Large collection of **pre-defined materials**:
  - pre-defined: density, elemental composition (with the pre-defined natural isotopic composition), mean ionization energy, density effect parameters, etc.

- **Use these pre-defined materials whenever possible:**
  - guarantees high accuracy for many derived parameters (consistency)
- **NIST and more pre-defined materials** (318 at the moment):
  - **single element NIST** materials with  $Z = [1-98]$  and named after the atomic symbol:
    - aluminum (“G4\_Al”), silicon (“G4\_Si”), gold (“G4\_Au”), etc.
  - **compound NIST** materials:
    - “G4\_AIR”, “G4\_ALUMINUM\_OXIDE”, “G4\_MUSCLE\_SKELETAL\_ICRP”, etc.
  - **HEP** and nuclear materials:
    - liquid argon “G4\_lAr”, lead tungstate “G4\_PbWO4”, “G4\_STAINLESS-STEEL”, etc.
  - **space** materials:
    - “G4\_KEVLAR”, “G4\_NEOPRENE”, etc.
  - **bio-chemical** materials:
    - the DNA bases “G4\_ADENINE”, “G4\_GUANINE”, “G4\_CYTOSINE”, “G4\_THYMINE”, etc.

- How to access these pre-define materials:
  - can be obtained from the **Geant4** NIST data base **by** using **their name**
  - their name starts with the “**G4\_**” **prefix** (see in the previous

```
// Use the NIST data base to get predefined materials: carbon, silicon
//
// get the simple pre-defined carbon material from the NIST data base
G4Material* matC = G4NistManager::Instance()->FindOrBuildMaterial("G4_C");
// get the simple pre-defined silicon material from the NIST data base
G4Material* matSi = G4NistManager::Instance()->FindOrBuildMaterial("G4_Si");
```

```
// Use the NIST data base to get pre-defined materials:
//
// get the NIST manager (just to simplify)
G4NISTManager* nistMGR = G4NistManager::Instance();
// get the pre-defined liquid argon ("G4_LAr") from the NIST DB
G4Material* matLAr = nistMGR->FindOrBuildMaterial("G4_LAr");
// get the pre-defined concrete ("G4_CONCRETE") from the NIST DB
G4Material* matConcr = nistMGR->FindOrBuildMaterial("G4_CONCRETE");
```

- List available pre-define **NIST** elements, materials from the data base with their composition:
  - user interface command:
    - /material/nist/printElement <SYMBOL>
    - /material/nist/listMaterials <CATEGORY>
  - directly from C++ code:

```
// List the pre-defined NIST ELEMENT(S) with its(their) isotope composition:
//
// element name can be: the element SYMBOL i.e. "Al" or "all"
const G4String nistElementName = "Al";
G4NistManager::Instance()->PrintElement(nistElementName);
//
// List the pre-defined NIST MATERIALS with their element composition:
//
// category name can be: "simple", "compound", "hep", "space", "bio", "all"
const G4String nistMatCategoryName = "simple";
G4NistManager::Instance()->ListMaterials(nistMatCategoryName);
```



- List available pre-define **NIST** elements, materials from the data base with their composition:
  - user interface command:

```
• /material /nist /printElement <SYMBOL>
```

**We will keep it simple and use NIST materials in our application**  
**Try these out with the simple main that we wrote to check the installation!**

Geometry definition

# **THE GEANT4 GEOMETRY DESCRIPTION**

- **Geant4** detector **geometry** description is composed of **three** conceptual **layers**: **Solid**, **Logical-Volume**, **Physical-Volume**
- **users** need to **construct them** directly in their user code (Detector Construction) by “`new`”, they get **registered** at construction in the corresponding store (**`G4SolidStore`**, **`G4LogicalVolumeStore`**, **`G4PhysicalVolumeStore`**) which will take care of deallocation of the corresponding memory at the end (if needed)
- geometry description can be rather complex but we will keep it simple now and focus only on the parts that we need
- more information on the detector geometry description can be found in the corresponding documentation: [Detector Geometry](#)

Geometry definition

**SOLID**

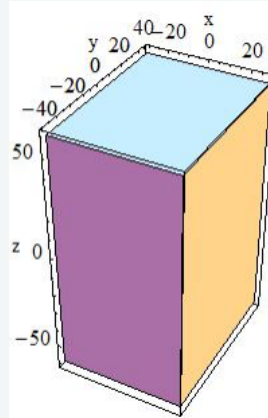
## G4VSolid:

- the **shape** of the **Geant4** detector geometry builds up from **geometrical primitives**, all derived from the **G4VSolid** base class that provides interface to:
  - compute distances between the shape and a given point
  - check whether a point is inside the shape
  - compute the extent of the shape
  - compute the surface normal to the shape at a given point
- **Geant4** makes use of **Constructed Solid Geometry** (CSG) to define these geometrical primitives: **G4Box**, **G4Tubes**, **G4Trd**, **G4Para**, **G4Trap**, **G4Torus**, etc.. (special CSG-like solids e.g. **G4Polycone**, **G4Polyhedra**, **G4Ellipsoid**, etc., tessellated and boolean solids are also available. See the [Geometry: Solids](#) documentation).
- these three-dimensional primitives **described by a minimal set of parameter** to define the dimensions of the corresponding solid e.g. **G4Box**
- these implement the **G4VSolid** base class interface methods

## Box:

To create a **box** one can use the constructor:

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



*In the picture:*

$pX = 30$ ,  $pY = 40$ ,  $pZ = 60$

by giving the box a name and its half-lengths along the X, Y and Z axis:

$pX$

half length in X

$pY$

half length in Y

$pZ$

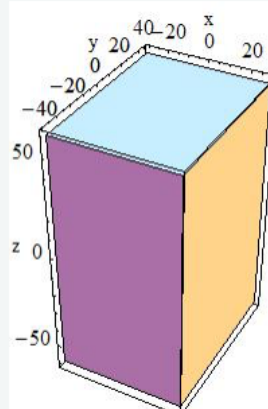
half length in Z

This will create a box that extends from  $-pX$  to  $+pX$  in X, from  $-pY$  to  $+pY$  in Y, and from  $-pZ$  to  $+pZ$  in Z.

## Box:

To create a **box** one can use the constructor:

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



```
// create solid (box) for the target  
G4Box* targetSolid = new G4Box("solid-Target",    // name  
                                0.5*targetXSize,   // half x-size  
                                0.5*targetYZSize,   // half y-size  
                                0.5*targetYZSize);  // half z-size
```

by giving the box a name and its half-lengths along the X, Y and Z axis:

pX	half length in X	pY	half length in Y	pZ	half length in Z
----	------------------	----	------------------	----	------------------

This will create a box that extends from  $-pX$  to  $+pX$  in X, from  $-pY$  to  $+pY$  in Y, and from  $-pZ$  to  $+pZ$  in Z.

Geometry definition

# **LOGICAL VOLUME**



## G4LogicalVolume:

- encapsulates **all information** of a detector volume element **except** its real physical **position** (position and rotation):
  - the **shape** and dimensions of the volume i.e. a **G4VSolid**
  - the **material** of the volume i.e. **G4Material** that is the minimally **required** additional information beyond the solid
  - additional, optional information such as magnetic field (**G4FieldManager**) or user defined limits (**G4UserLimits**), etc.
- it's **NOT** a base class! Its constructor:

```
G4LogicalVolume( G4VSolid*           pSolid,           // its Solid
                  G4Material*         pMaterial,        // its Material
                  const G4String&     Name,             // its Name
                  G4FieldMgr*         pFieldMgr=0,
                  G4VSensitiveDetector* pSDetector=0,
                  G4UserLimits*       pULimits=0,
                  G4bool               Optimise=true )
```

```
G4LogicalVolume* targetLogical = new G4LogicalVolume(targetSolid, // solid
                                                       materialTarget, // material
                                                       "logic-Target"); // name
```

## G4LogicalVolume:

- encapsulates **all information** of a detector volume element **except** its real physical **position** (position and rotation):
  - the **shape** and dimensions of the volume i.e. a **G4VSolid**
  - the **material** of the volume i.e. **G4Material** that is the minimally **required** additional information beyond the solid
  - additional, optional information such as magnetic field (**G4FieldManager**) or user defined limits (**G4UserLimits**), etc.
- it's **NOT** a base class! Its constructor:
 

```
G4LogicalVolume( G4VSolid*           pSolid,           // its Solid
                  G4Material*         pMaterial,         // its Material
                  const G4String&      Name,              // its Name
                  G4FieldManager*      pFieldMgr=0,
                  G4VSensitiveDetector* pSDetector=0,
                  G4UserLimits*        pULimits=0,
                  G4bool                Optimise=true )
```
- see the [Geometry: Logical Volumes](#) documentation

Geometry definition

# **PHYSICAL VOLUME**

## G4VPhysicalVolume:

- the abstract base class for representation of physically **positioned volumes**
- a volume is **positioned in a mother volume** relative to its coordinate system
- the positioning can be:
  - **placement volume: one positioned volume**, i.e. **one G4VPhysicalVolume** object represents **one “real” volume**
  - **repeated volume: one volume positioned many times**, i.e. **one G4VPhysicalVolume** object represents **multiple copies of “real” volumes** (reduces memory by exploiting symmetry)
    - **Replica volumes**: the multiple **copies** of the volume **are all identical**
    - **Parameterised volumes**: the multiple **copies of** a volume can be different in **size, solid type, or material** that can all be **parameterised** as a **function of the copy number**
- we will have a look only to the **placement** but see all at the **Geometry: Physical Volume** documentation

- represent **one positioned G4LogicalVolume**
- created by **associating a G4LogicalVolume with a Transformation** that defines the **position** of the volume **in the mother volume**
- the **Transformation** can be given either as a single **G4Transform3d** object or as combination of rotation **G4RotationMatrix** and translation **G4ThreeVector**
- a **mother volume must be specified** for all volumes **except** the “**world**”
- (one of the two) constructor with the rotation matrix and translation vector:

```
G4PVPlacement (      G4RotationMatrix*   pRot,           // rot.-matrix
                    const G4ThreeVector&   tlate,         // translation
                    G4LogicalVolume*       pCurrentLogical, // logical volume
                    const G4String&        pName,         // name
                    G4LogicalVolume*       pMotherLogical, // mother logical volume
                    G4bool                  pMany,         //
                    G4int                   pCopyNo,       // unique identifier
                    G4bool                  pSurfChk=false ) // check overlap ?
```

```
G4VPhysicalVolume* targetPhyscal = new G4PVPlacement(nullptr, // (no) rotation
G4ThreeVector(0.,0.,0.), // translation
targetLogical, // its logical volume
"Target", // its name
worldLogical, // its mother volume
false, // not used
0); // cpy number
```

Everything has shown that is necessary to write  
**YourDetectorConstruction**

**START TO DEVELOP OUR APPLICATION**