Geant4 Tutorial

SciNeGHE – Trieste 2010

An hands-on course based on Geant4 with emphasis on high energy astroparticle physics.

Lectures will cover all aspects of Geant4 from basic installation through advanced topics and will be interspersed with examples that build a progressively more complex application.

The course should be of interest both to complete novices and to those who already have some basic familiarity with Geant4. Participants are expected to have a reasonable knowledge of C++.

Based on Lectures by the G4 collaboration

Outline of the G4 course

- General Introduction to G4
  - What is G4 ?
  - Review of user documentation
  - Geant4 as a toolkit
- Geant4 Kernel and basics of the toolkit
  - Run, Event, Step
  - Particle and Physics processes
  - User classes
- Geant4 geometry construction
  - Materials
  - Volumes
- Particle Generation in G4
- Geant4 physics
- Astroparticle Examples

User classes

To use Geant4, you have to …

- Geant4 is a toolkit. You have to build an application.
- To make an application, you have to
  - Define your geometrical setup
    - Material, volume
  - Define physics to get involved
    - Particles, physics processes/models
    - Production thresholds
  - Define how an event starts
    - Primary track generation
  - Extract information useful to you
- You may also want to
  - Visualize geometry, trajectories and physics output
  - Utilize (Graphical) User Interface
  - Define your own UI commands
  - etc.

User classes

- main()
  - Geant4 does not provide *main().*
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
- Action classes
  - Use G4RunManager::SetUserAction() to define.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

Note : classes written in yellow are mandatory.

The main program

- Geant4 does not provide the *main*().
- In your *main(),* you have to
  - Construct G4RunManager (or your derived class)
  - Set user mandatory classes to RunManager
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
    - G4VUserPrimaryGeneratorAction
- You can define VisManager, (G)UI session, optional user action classes, and/or your persistency manager in your *main()*.

Describe your detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class.
- In the virtual method *Construct()*,
  - Instantiate all necessary materials
  - Instantiate volumes of your detector geometry
  - Instantiate your sensitive detector classes and set them to the corresponding logical volumes
- Optionally you can define
  - Regions for any part of your detector
  - Visualization attributes (color, visibility, etc.) of your detector elements

Select physics processes

- Geant4 does not have any default particles or processes.
  - Even for the particle transportation, you have to define it explicitly.
- Derive your own concrete class from G4VUserPhysicsList abstract base class.
  - Define all necessary particles
  - Define all necessary processes and assign them to proper particles
  - Define cut-off ranges applied to the world (and each region)
- Geant4 provides lots of utility classes/methods and examples.
  - "Educated guess" physics lists for defining hadronic processes for various use-cases.

Generate primary event

- Derive your concrete class from G4VUserPrimaryGeneratorAction abstract base class.
- Pass a G4Event object to one or more primary generator concrete class objects which generate primary vertices and primary particles.
- Geant4 provides several generators in addition to the G4VPrimaryParticlegenerator base class.
  - G4ParticleGun
  - G4HEPEvtInterface, G4HepMCInterface
    - Interface to /hepevt/ common block or HepMC class
  - G4GeneralParticleSource
    - Define radioactivity

Optional user action classes

- All user action classes, methods of which are invoked during “Beam On”, must be constructed in the user’s *main*() and must be set to the RunManager.
- G4UserRunAction
  - G4Run* GenerateRun()
    - Instantiate user-customized run object
  - void BeginOfRunAction(const G4Run*)
    - Define histograms
  - void EndOfRunAction(const G4Run*)
    - Analyze the run
    - Store histograms
- G4UserEventAction
  - void BeginOfEventAction(const G4Event*)
    - Event selection
  - void EndOfEventAction(const G4Event*)
    - Output event information

Optional user action classes

- G4UserTrackingAction
  - void PreUserTrackingAction(const G4Track*)
    - Decide trajectory should be stored or not
    - Create user-defined trajectory
  - void PostUserTrackingAction(const G4Track*)
    - Delete unnecessary trajectory
- G4UserSteppingAction
  - void UserSteppingAction(const G4Step*)
    - Kill / suspend / postpone the track
    - Draw the step (for a track not to be stored as a trajectory)

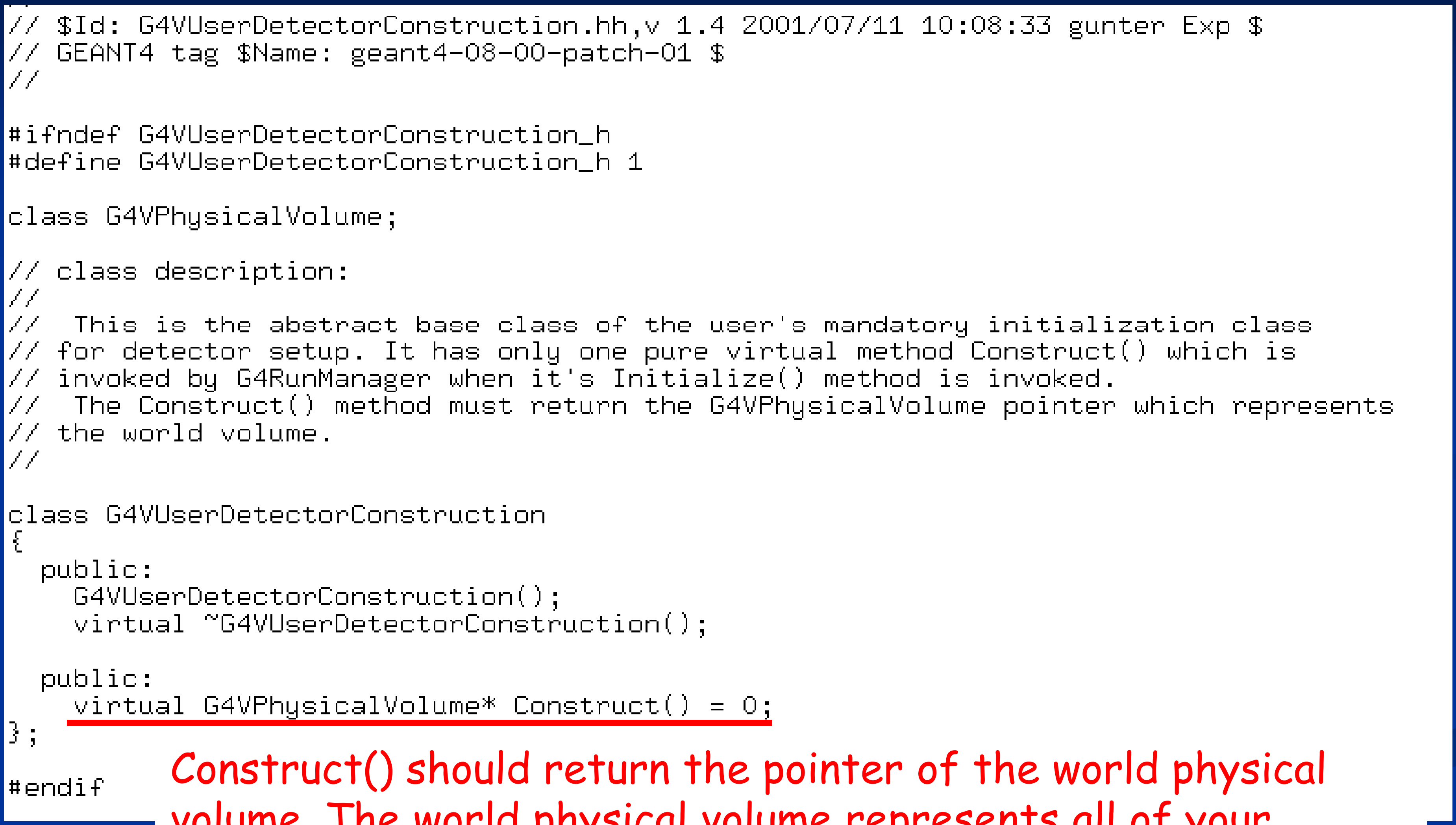
G4VUserDetectorConstruction

User classes

- main()
  - Geant4 does not provide *main().*

  Note : classes written in yellow are mandatory.
- Initialization classes
  - Use G4RunManager::SetUserInitialization() to define.
  - Invoked at the initialization
    - G4VUserDetectorConstruction
    - G4VUserPhysicsList
- Action classes
  - Use G4RunManager::SetUserAction() to define.
  - Invoked during an event loop
    - G4VUserPrimaryGeneratorAction
    - G4UserRunAction
    - G4UserEventAction
    - G4UserStackingAction
    - G4UserTrackingAction
    - G4UserSteppingAction

G4VUserDetectorConstruction



```
// $Id: G4VUserDetectorConstruction.hh,v 1.4 2001/07/11 10:08:33 gunter Exp $
// GEANT4 tag $Name: geant4-08-00-patch-01 $
//

#ifndef G4VUserDetectorConstruction_h
#define G4VUserDetectorConstruction_h 1

class G4VPhysicalVolume;

// class description:
//
//  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.
//

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

  public:
    virtual G4VPhysicalVolume* Construct() = 0;
};

#endif
```

Construct() should return the pointer of the world physical volume. The world physical volume represents all of your geometry setup.

Your detector construction

```
#ifndef MyDetctorConstruction_h
#define MyDetctorConstruction_h 1
#include “G4VUserDetectorConstruction.hh”
class MyDetctorConstruction
     : public G4VUserDetectorConstruction
{
 public:
  G4VUserDetectorConstruction();
  virtual ~G4VUserDetectorConstruction();
  virtual G4VPhysicalVolume* Construct();
 public:
  // set/get methods if needed
 private:
  // granular private methods if needed
  // data members if needed
};
#endif
```

Describe your detector

- Derive your own concrete class from G4VUserDetectorConstruction abstract base class.
- Implement the method Construct()
  1) Construct all necessary materials
  2) Define shapes/solids
  3) Define logical volumes
  4) Place volumes of your detector geometry
  5) Associate (magnetic) field to geometry *(optional)*
  6) Instantiate sensitive detectors / scorers and set them to corresponding volumes *(optional)*
  7) Define visualization attributes for the detector elements *(optional)*
  8) Define regions *(optional)*
- Set your construction class to G4RunManager
- It is suggested to modularize Construct() method w.r.t. each component or sub-detector for easier maintenance of your code.

Lecture 2
Geometry - Materials

Definition of material

Definition of Materials

- Different kinds of materials can be described:
  - isotopes <-> G4Isotope
  - elements <-> G4Element
  - molecules, compounds and mixtures <-> G4Material
- Attributes associated to G4Material:
  - temperature, pressure, state, density
- Single element material

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

Material: molecule

- A Molecule is made of several elements (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,ncomp=2);
G4int natoms;
H2O->AddElement(elH, natoms=2);
H2O->AddElement(elO, natoms=1);
```

Material: compound

- Compound: 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);
G4double fracMass;
Air->AddElement(elN, fracMass=70.0*perCent);
Air->AddElement(elO, fracMass=30.0*perCent);
```

Material: mixture

- Composition of compound materials

```
G4Element* elC  = …;    // define “carbon” element
G4Material* SiO2 = …;   // define “quartz” material
G4Material* H2O = …;    // define “water” material

density = 0.200*g/cm3;
G4Material* Aerog =
   new G4Material("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);
```

Element with user defined abundance

- An element can be created according to user defined abundances

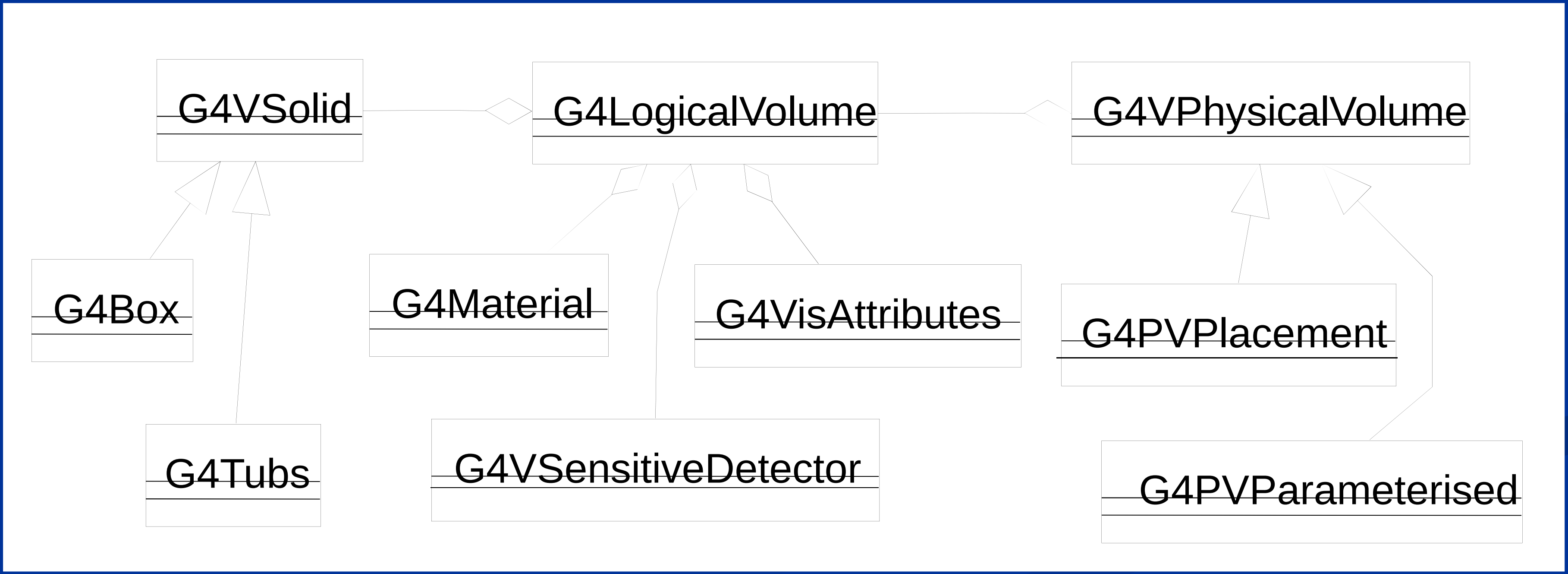
- Ex. Create an enriched Uranium for nuclear power generation

```
G4Isotope* isoU235 =
      new G4Isotope(“U235", iz=92, ia=235, a=235.0439242*g/mole);
 G4Isotope* isoU238 =
    new G4Isotope(“U238", iz=92, ia=238, a=238.0507847 *g/mole);

 G4Element* elenrichedU =
    new G4Element(“enriched U", symbol=“U" , ncomponents=2);
 elenrichedU->AddIsotope(isoU235, abundance=80.*perCent);
 elenrichedU->AddIsotope(isoU235, abundance=20.*perCent);

 G4Material* matenrichedU=
    new G4Material(“U for nuclear    power generation" , density=
      19.050*g/cm3 , ncomponents = 1 , kStateSolid );
 matenrichedU>AddElement( elenrichedU , fractionmass = 1 );
```

Lecture 2
Geometry - Volumes

Detector geometry

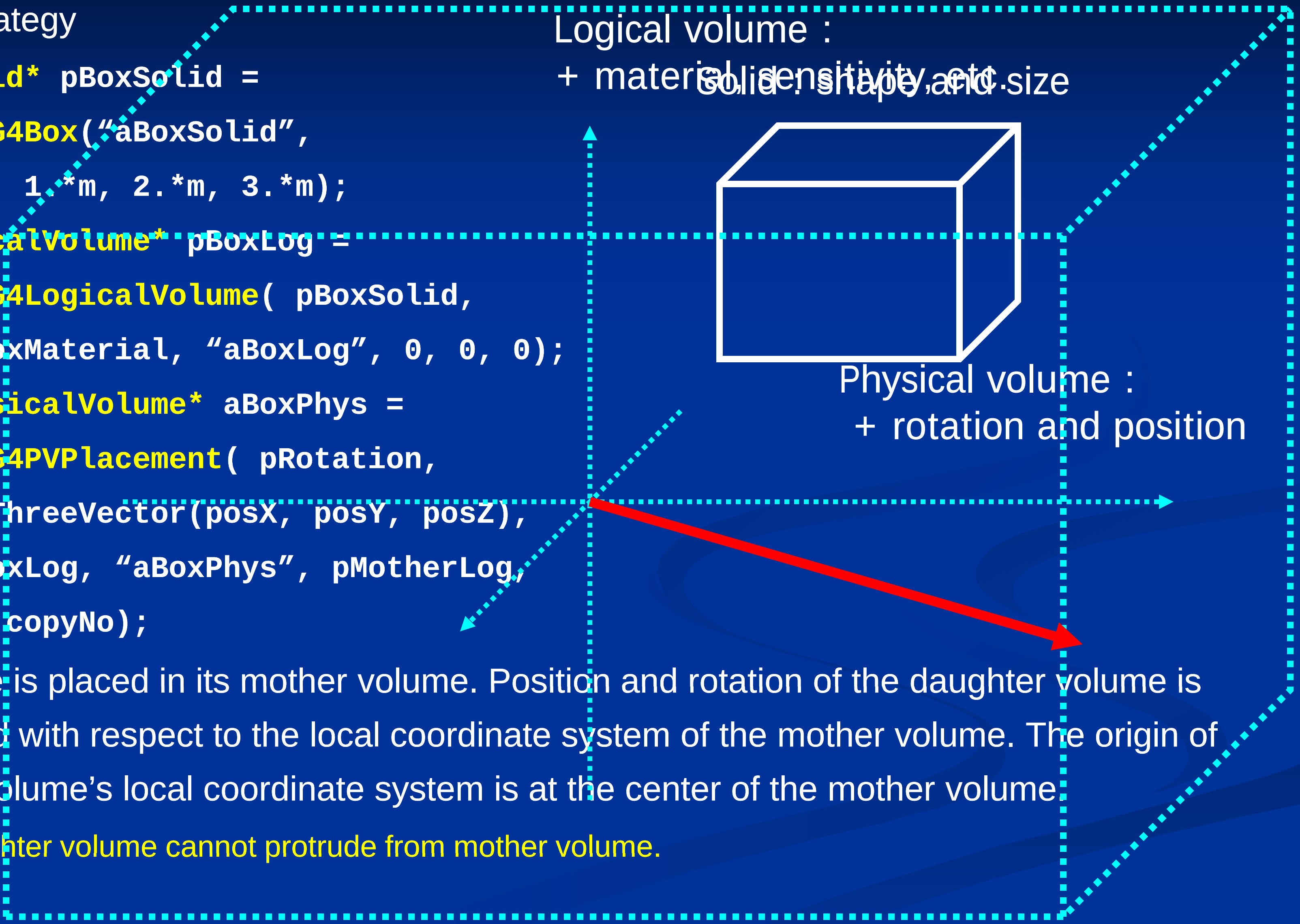
- Three conceptual layers
  - G4VSolid -- *shape, size*
  - G4LogicalVolume -- *daughter physical volumes,*
    material, sensitivity, user limits, etc.
  - G4VPhysicalVolume -- *position, rotation*



Define detector geometry

- Basic strategy

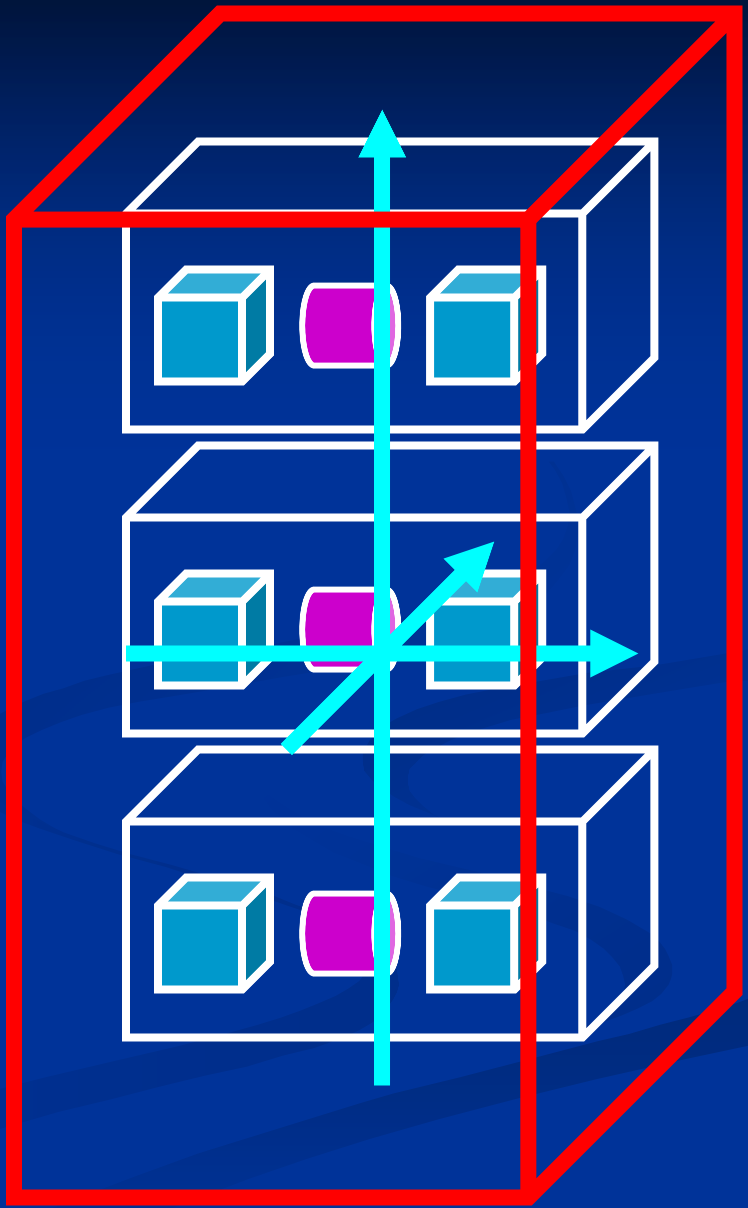
```
G4VSolid* pBoxSolid =
  new G4Box("aBoxSolid",
        1.*m, 2.*m, 3.*m);
G4LogicalVolume* pBoxLog =
  new G4LogicalVolume( pBoxSolid,
    pBoxMaterial, "aBoxLog", 0, 0, 0);
G4VPhysicalVolume* aBoxPhys =
  new G4PVPlacement( pRotation,
    G4ThreeVector(posX, posY, posZ),
    pBoxLog, "aBoxPhys", pMotherLog,
    0, copyNo);
```



- A volume is placed in its mother volume. Position and rotation of the daughter volume is described with respect to the local coordinate system of the mother volume. The origin of mother volume's local coordinate system is at the center of the mother volume.
  - Daughter volume cannot protrude from mother volume.

Geometrical hierarchy

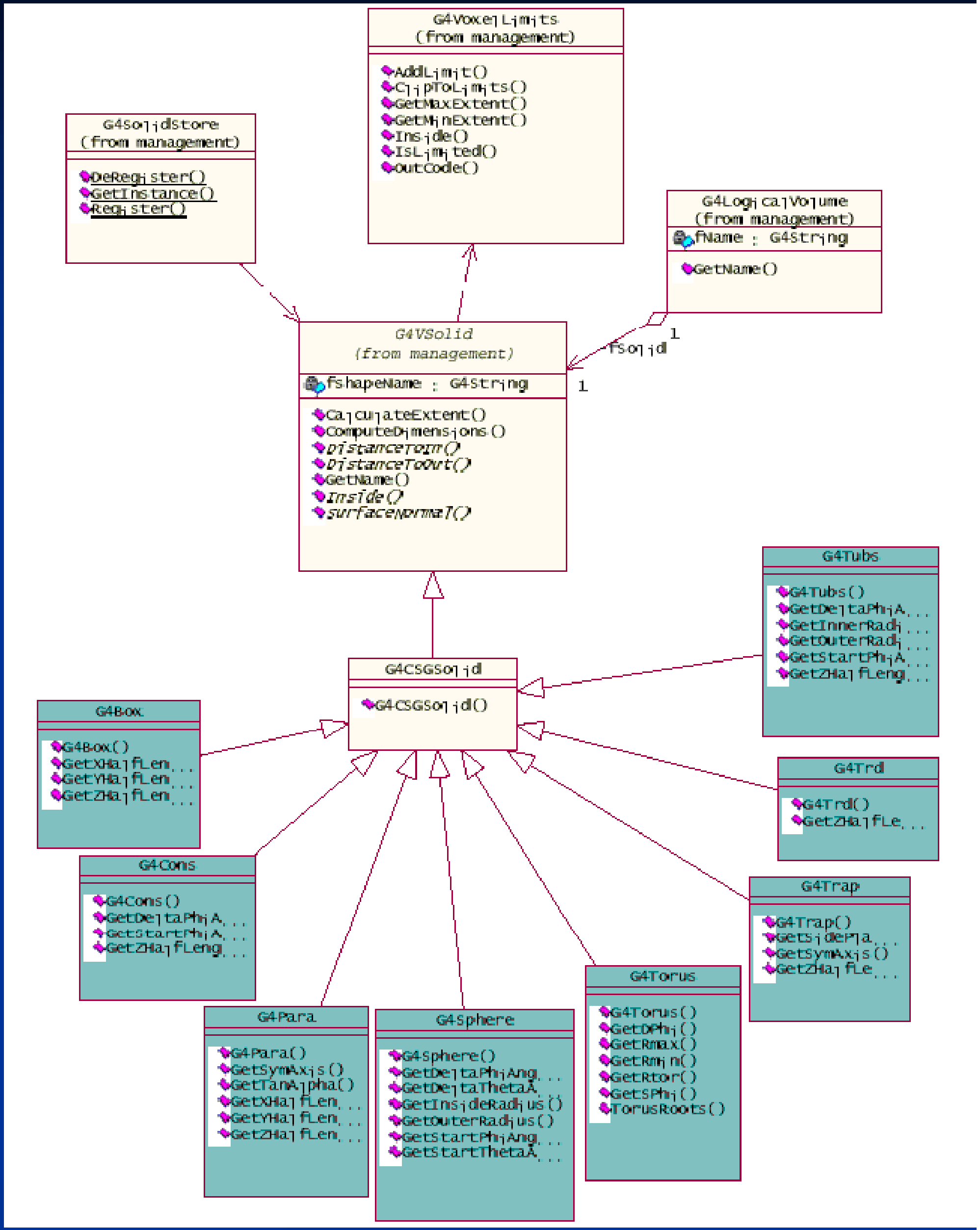
- One logical volume can be placed more than once. One or more volumes can be placed to a mother volume.
- Note that the mother-daughter relationship is an information of G4LogicalVolume.
  - If the mother volume is placed more than once, all daughters are by definition appear in all of mother physical volumes.
- The world volume must be a unique physical volume which fully contains all the other volumes.
  - The world volume defines the global coordinate system. The origin of the global coordinate system is at the center of the world volume.
  - Position of a track is given with respect to the global coordinate system.



Solid and shape

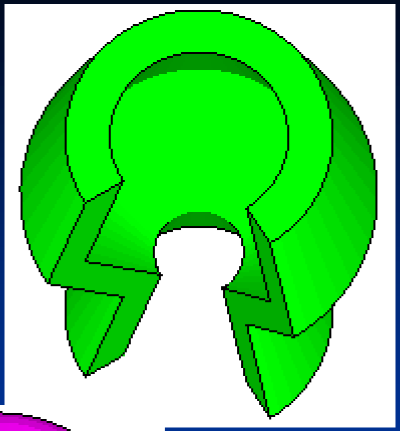
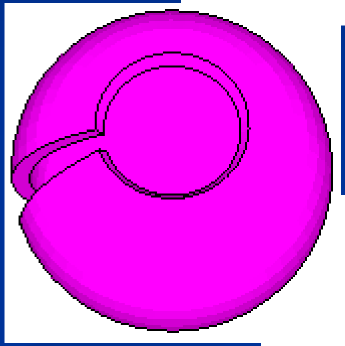
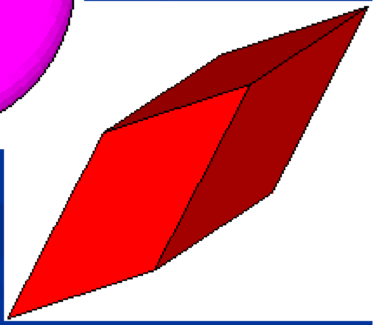
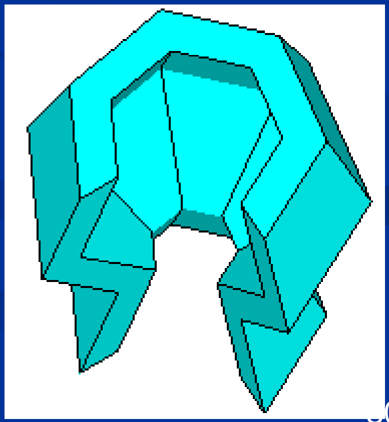
G4VSolid

- Abstract class. All solids in Geant4 are derived from it
- It defines but does not implement all functions required 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
- User can create his/her own solid class



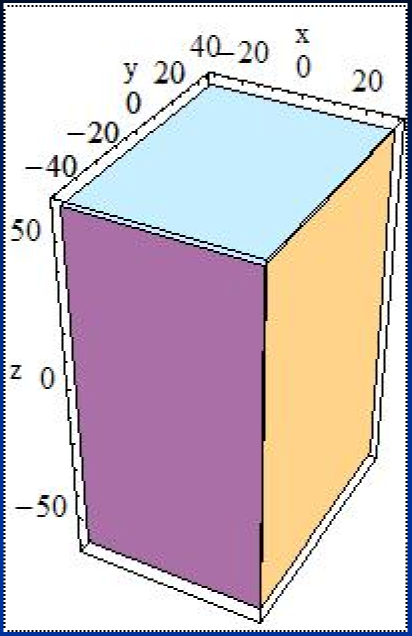
Solids

- Solids defined in Geant4:
  - CSG (Constructed Solid Geometry) solids
    - G4Box, G4Tubs, G4Cons, G4Trd, ...
    - Analogous to simple GEANT3 CSG solids
  - Specific solids (CSG like)
    - G4Polycone, G4Polyhedra, G4Hype, ...
  - BREP (Boundary REPresented) solids
    - G4BREPSolidPolycone, G4BSplineSurface, ...
    - Any order surface
  - Boolean solids
    - G4UnionSolid, G4SubtractionSolid, ...

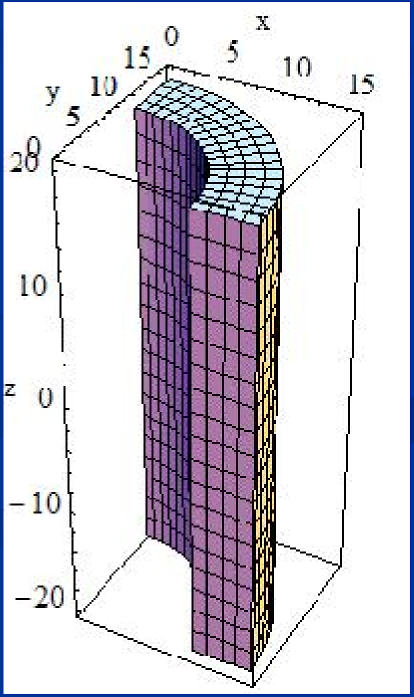





CSG: G4Box, G4Tubs

```
G4Box(const G4String &pname,   // name
            G4double half_x,   // X half size
            G4double half_y,   // Y half size
            G4double half_z);  // Z half size
```



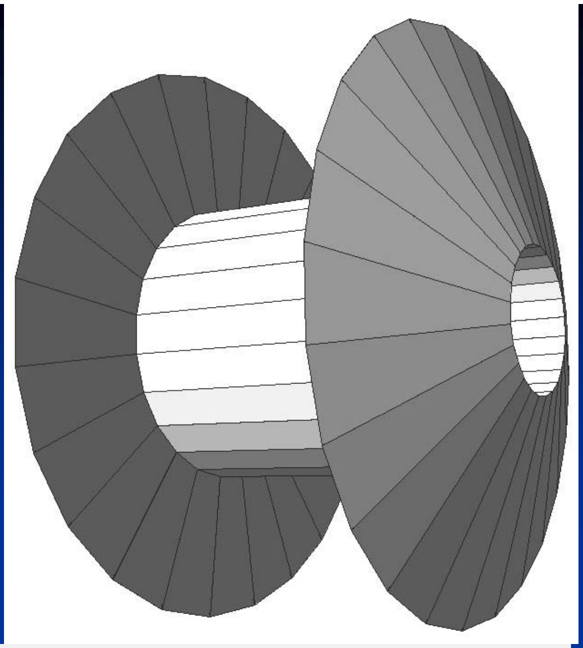
```
G4Tubs(const G4String &pname,  // name
             G4double  pRmin,  // inner radius
             G4double  pRmax,  // outer radius
             G4double  pDz,    // Z half length
             G4double  pSphi,  // starting Phi
             G4double  pDphi); // segment angle
```

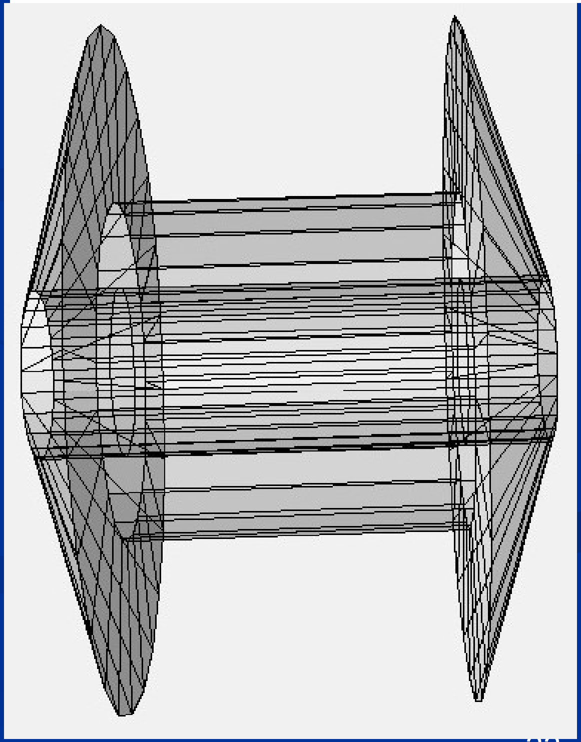


BREP Solids

- *BREP = Boundary REPresented Solid*
- Listing all its surfaces specifies a solid
  - e.g. 6 planes for a cube
- Surfaces can be
  - planar, 2nd or higher order
    - elementary BREPS
  - Splines, B-Splines,

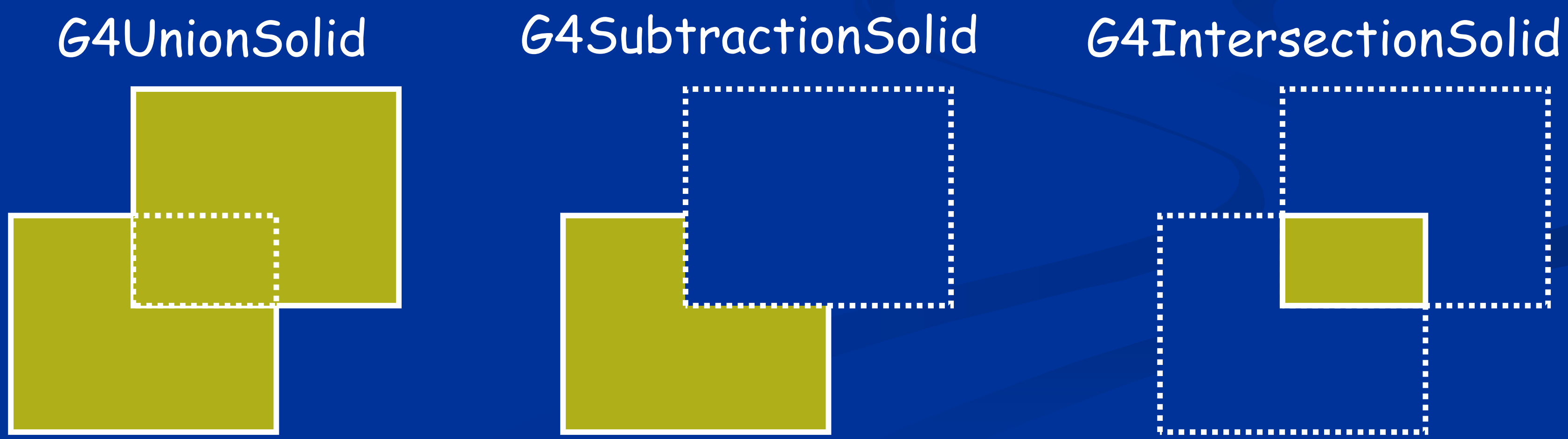
    NURBS (Non-Uniform B-Splines)
    - advanced BREPS
- Few elementary BREPS pre-defined
  - box, cons, tubs, sphere, torus, polycone, polyhedra
- Advanced BREPS built through CAD systems





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
  - 2nd solid is positioned relative to the coordinate system of the 1st solid
  - Result of boolean operation becomes a solid. Thus the third solid can be combined to the resulting solid of first operation.
- Solids to be combined can be either CSG or other Boolean solids.
- Note: tracking cost for the navigation in a complex Boolean solid is proportional to the number of constituent CSG solids



Boolean Solids - example

```
G4VSolid* box = new G4Box(“Box",50*cm,60*cm,40*cm);
G4VSolid* cylinder
 = new G4Tubs(“Cylinder”,0.,50.*cm,50.*cm,0.,2*M_PI*rad);
G4VSolid* union
 = 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.));
```

- The origin and the coordinates of the combined solid are the same as those of the first solid.

G4LogicalVolume

G4LogicalVolume

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

- Contains all information of volume except position and rotation
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits, Region
- Physical volumes of same type can share the common logical volume object

Define detector geometry

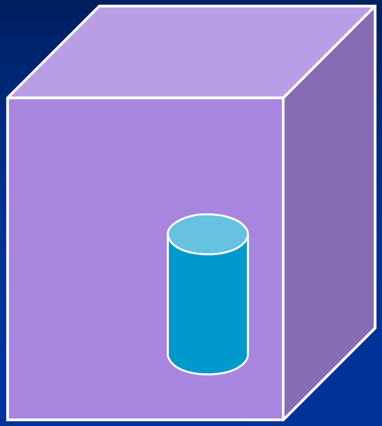
- Basic strategy

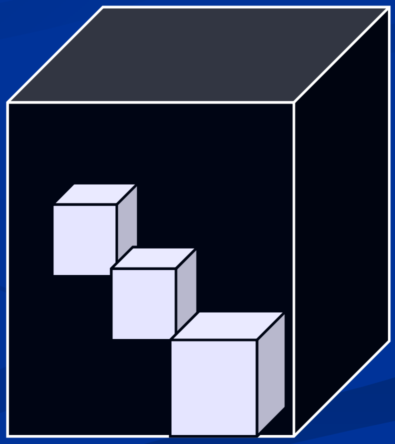
```
G4VSolid* pBoxSolid =
  new G4Box(“aBoxSolid”, 1.*m, 2.*m, 3.*m);
G4LogicalVolume* pBoxLog =
  new G4LogicalVolume( pBoxSolid, pBoxMaterial,
                       “aBoxLog”, 0, 0, 0);
G4VPhysicalVolume* aBoxPhys =
  new G4PVPlacement( pRotation,
    G4ThreeVector(posX, posY, posZ), pBoxLog,
    “aBoxPhys”, pMotherLog, 0, copyNo);
```

Physical volume

Physical Volumes

- Placement volume : it is one positioned volume
  - One physical volume object represents one "real" volume.
- Repeated volume : a volume placed many times
  - One physical volume object <u>represents</u> any number of "real" volumes.
  - reduces use of memory.
  - Parameterised
    - repetition w.r.t. copy number
  - Replica and Division
    - simple repetition along one axis
- A mother volume can contain either
  - many placement volumes
  - or, one repeated volume


placement


repeated

Physical volume

- G4PVPlacement 1 Placement = One Placement Volume
  - A volume instance positioned once in its mother volume
- G4PVParameterised 1 Parameterized = Many Repeated Volumes
  - Parameterized by the copy number
    - Shape, size, material, sensitivity, vis attributes, position and rotation can be parameterized by the copy number.
    - You have to implement a concrete class of G4VPVParameterisation.
  - Reduction of memory consumption
  - Currently: parameterization can be used only for volumes that either

    a) have no further daughters, or

    b) are identical in size & shape (so that grand-daughters are safely fit inside).
  - By implementing G4PVNestedParameterisation instead of G4VPVParameterisation, material, sensitivity and vis attributes can be parameterized by the copy numbers of ancestors.

Physical volume

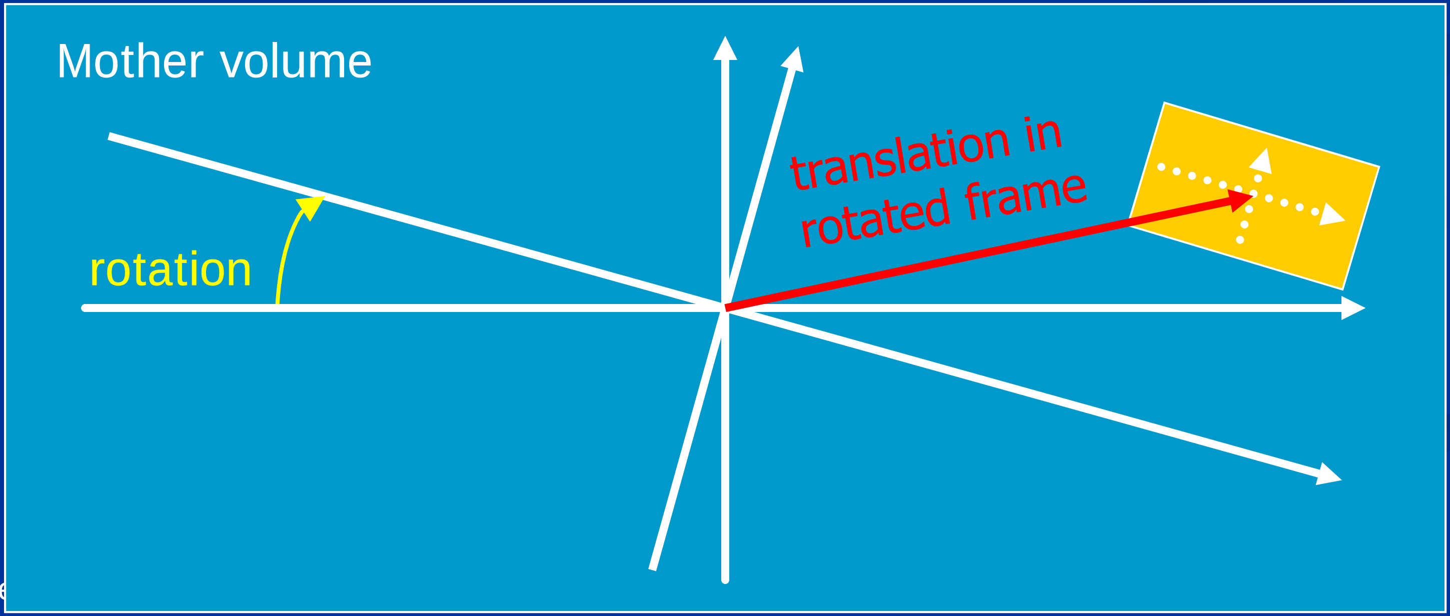
- G4PVReplica 1 Replica = Many Repeated Volumes
  - Daughters of same shape are aligned along one axis
  - Daughters fill the mother completely without gap in between.
- G4PVDivision 1 Division = Many Repeated Volumes
  - Daughters of same shape are aligned along one axis and fill the mother.
  - There can be gaps between mother wall and outmost daughters.
  - No gap in between daughters.
- G4ReflectionFactory 1 Placement = a pair of Placement volumes
  - generating placements of a volume and its reflected volume
  - Useful typically for end-cap calorimeter
- G4AssemblyVolume 1 Placement = a set of Placement volumes
  - Position a group of volumes

G4PVPlacement

G4PVPlacement

```
G4PVPlacement(G4RotationMatrix* pRot,    // rotation of mother frame
            const G4ThreeVector &tlate, // position in rotated frame
            G4LogicalVolume *pDaughterLogical,
            const G4String &pName,
            G4LogicalVolume *pMotherLogical,
            G4bool pMany, // ‘true’ is not supported yet…
            G4int pCopyNo, // unique arbitrary integer
            G4bool pSurfChk=false); // optional boundary check
```

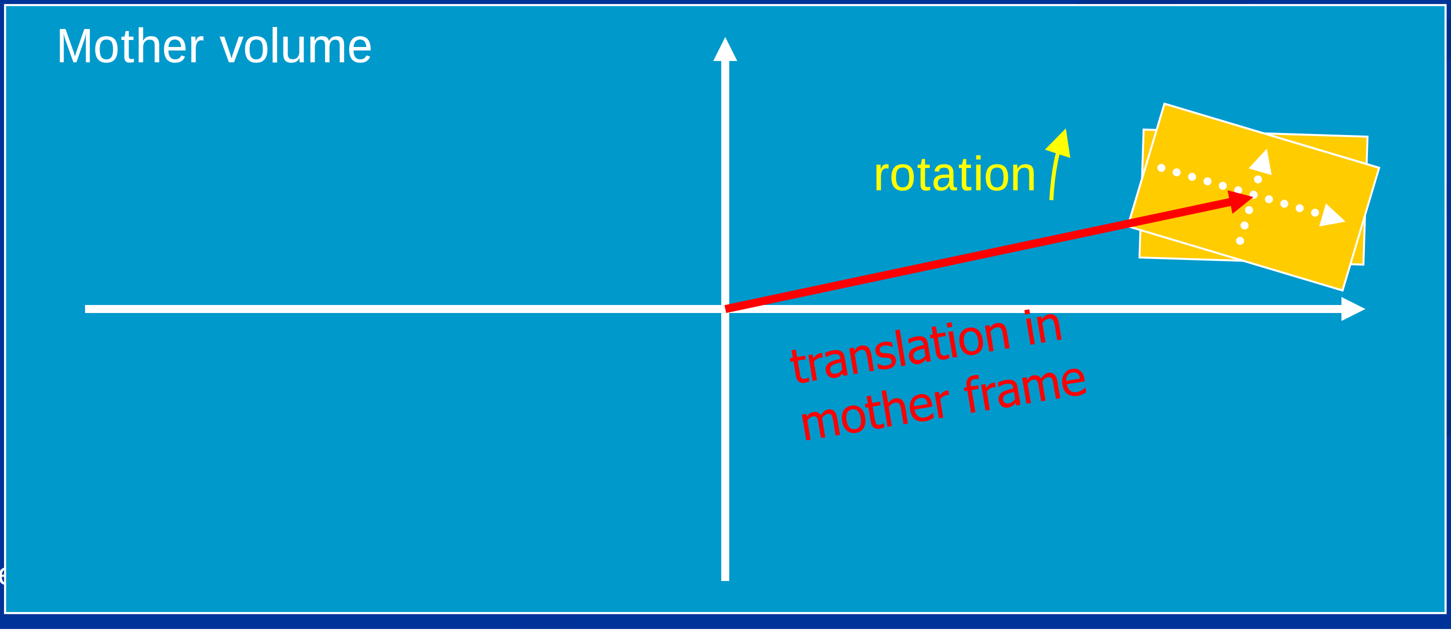
- Single volume positioned relatively to the mother volume.



Alternative G4PVPlacement

```
G4PVPlacement(
  G4Transform3D(G4RotationMatrix &pRot, // rotation of daughter frame
                const G4ThreeVector &tlate), // position in mother frame
  G4LogicalVolume *pDaughterLogical,
  const G4String &pName,
  G4LogicalVolume *pMotherLogical,
  G4bool pMany, // ‘true’ is not supported yet…
  G4int pCopyNo, // unique arbitrary integer
  G4bool pSurfChk=false); // optional boundary check
```

- Single volume positioned relatively to the mother volume.



Parameterized volume

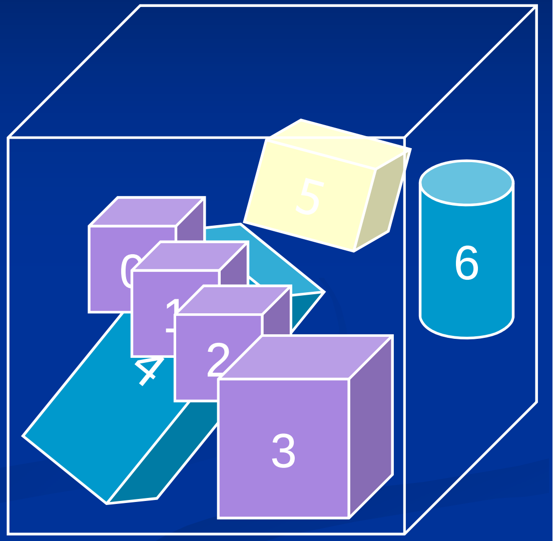
G4PVParameterised

```
G4PVParameterised(const G4String& pName,
                  G4LogicalVolume* pLogical,
                  G4LogicalVolume* pMother,
                  const EAxis pAxis,
                  const G4int nReplicas,
                  G4VPVParameterisation *pParam
                  G4bool pSurfChk=false);
```

- Replicates the volume **`nReplicas`** times using the parameterization **`pParam`**, within the mother volume **`pMother`**
- **`pAxis`** is a suggestion to the navigator along which Cartesian axis replication of parameterized volumes dominates.
  - kXAxis, kYAxis, kZAxis : one-dimensional optimization
  - kUndefined : three-dimensional optimization

Parameterized Physical Volumes

- User should implement a class derived from G4VPVParameterisation abstract base class and define following as a function of copy number
  - where it is positioned (transformation, rotation)
- Optional:
  - the size of the solid (dimensions)
  - the type of the solid, material, sensitivity, vis attributes
- All daughters must be fully contained in the mother.
- Daughters should not overlap to each other.
- Limitations:
  - Applies to simple CSG solids only
  - Granddaughter volumes allowed only for special cases
  - Consider parameterised volumes as “leaf” volumes
- Typical use-cases
  - Complex detectors
    - with large repetition of volumes, regular or irregular
  - Medical applications
    - the material in animal tissue is measured as cubes with varying material

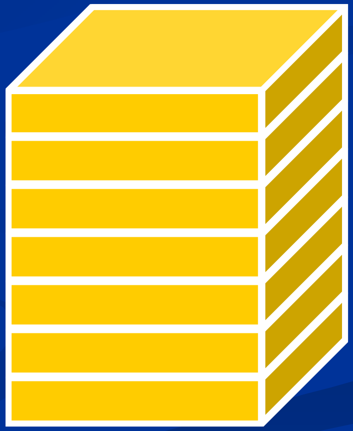


Replicated volume

Replicated Volumes

- The mother volume is completely filled with replicas, all of which are the same size (width) and shape.
- Replication may occur along:
  - Cartesian axes (X, Y, Z) – slices are considered perpendicular to the axis of replication
    - Coordinate system at the center of each replica
  - Radial axis (Rho) – cons/tubs sections centered on the origin and un-rotated
    - Coordinate system same as the mother
  - Phi axis (Phi) – phi sections or wedges, of cons/tubs form
    - Coordinate system rotated such as that the X axis bisects the angle made by each wedge

a daughter logical volume to be replicated



mother volume

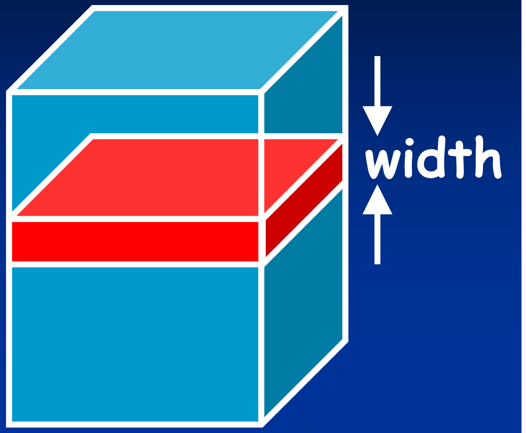
Replica - axis, width, offset

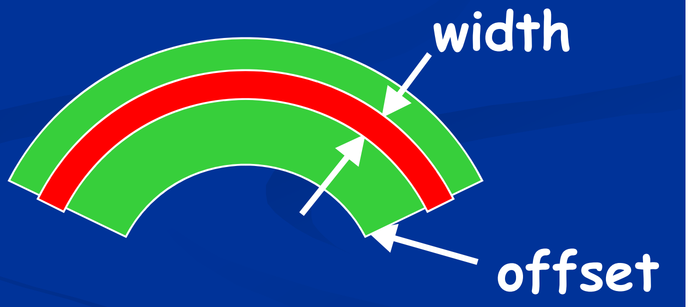
- Cartesian axes - **`kXaxis, kYaxis, kZaxis`**
  - Center of n-th daughter is given as

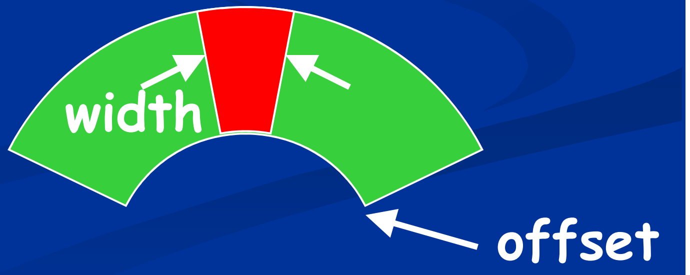
    `-width*(nReplicas-1)*0.5+n*width`
  - Offset shall not be used
- Radial axis - **`kRaxis`**
  - Center of n-th daughter is given as

    `width*(n+0.5)+offset`
  - Offset must be the inner radius of the mother
- Phi axis - **`kPhi`**
  - Center of n-th daughter is given as

    `width*(n+0.5)+offset`
  - Offset must be the starting angle of the mother







How To ?

- Create a new directory in your home
- Copy the example to your directory
- Source the env.sh command to set the env variables
- Be sure to have set G4WORKDIR env variable
- Make the example
- Run the example:
  - **$G4WORKDIR/bin/Linux-g++/exampleN03**