

Defining Material and Geometry

Contents

- › G4VUserDetectorConstruction class
- › Material
- › Solid and volume
- › Various ways of placement
- › Visualization attributes
- › Additional features in geometry
 - › Geometry checking tools

G4VUserDetectorConstruction

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 required to describe the geometry
 - 3) Construct and place volumes of your detector geometry
 - 4) Instantiate sensitive detectors and set them to corresponding volumes (optional)
 - 5) Associate magnetic field to detector (optional)
 - 6) Define visualization attributes for the detector elements (optional)
 - 7) Define regions (optional)
- › Set your construction class to `G4RunManager`
- › Modularize it w.r.t. each detector component or sub-detector for easier maintenance of your code

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
- ▶ Prefer low-density material to vacuum
- ▶ For hadronic processes, you have to have at least correct set of elements. You should not use "averaged material".
- ▶ 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);
```

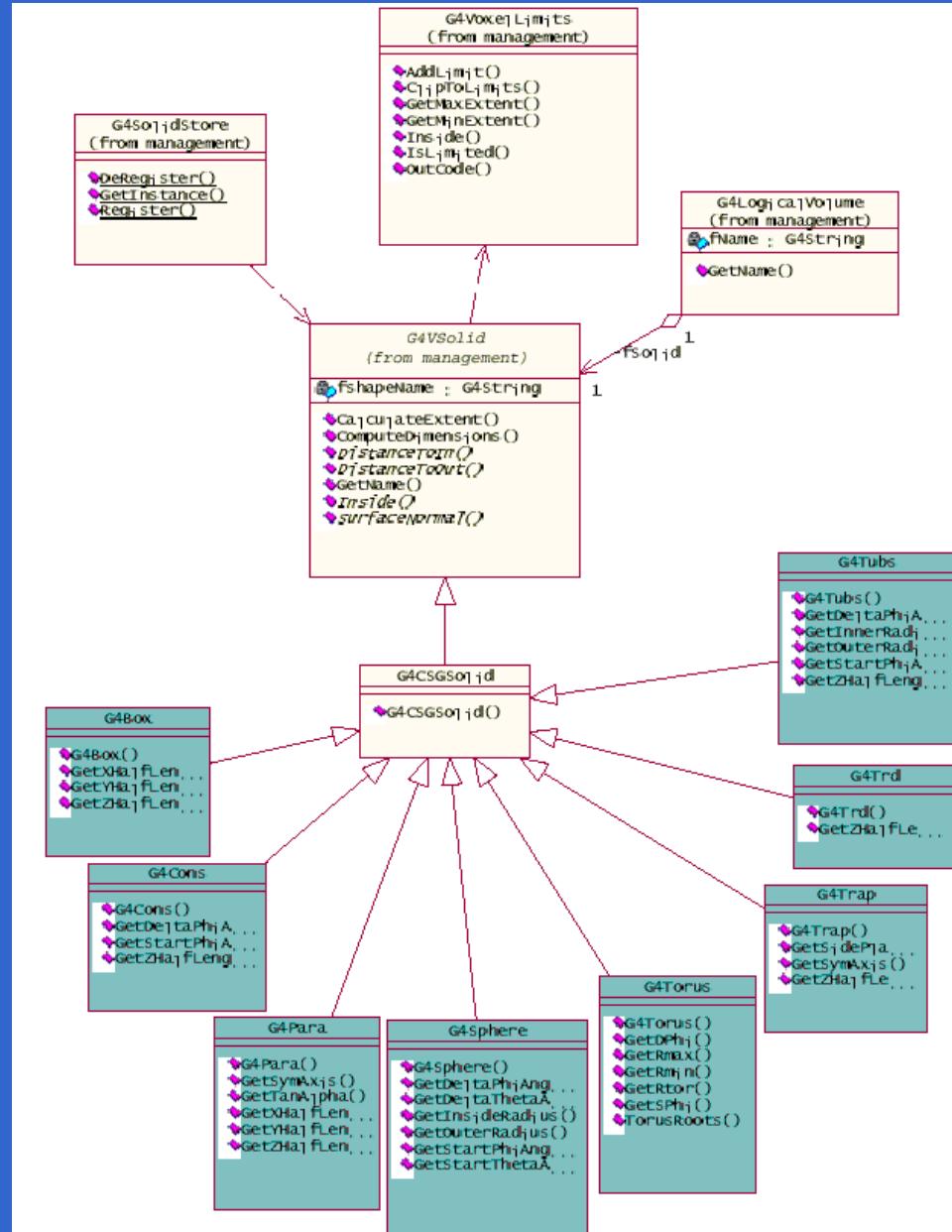
MATERIALS from Data Base

- › G4NistManager* nistMan = G4NistManager::Instance();
- › G4Material* Water = nistMan->FindOrBuildMaterial("G4_WATER");
- ›
- › Получаем указатель на базу данных (*nistMan) и запрашиваем материал "G4_WATER" (в начале всех материалов из базы Geant4 содержится префикс "G4_").

Solid and shape

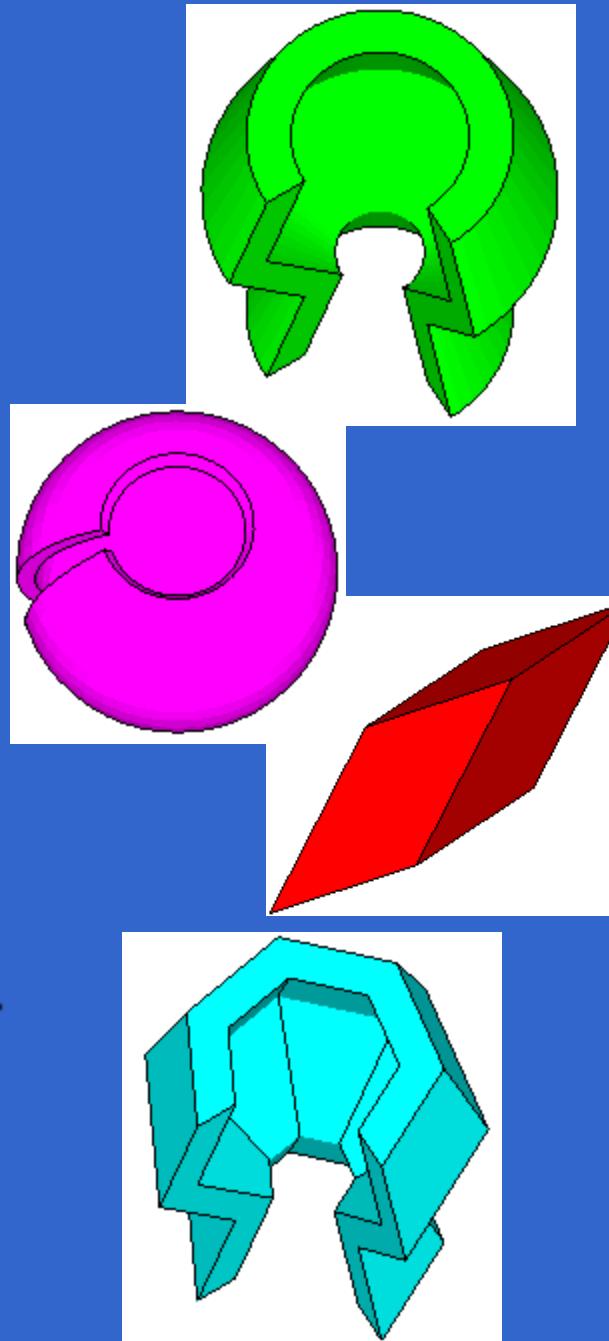
G4VSolid

- 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, ...



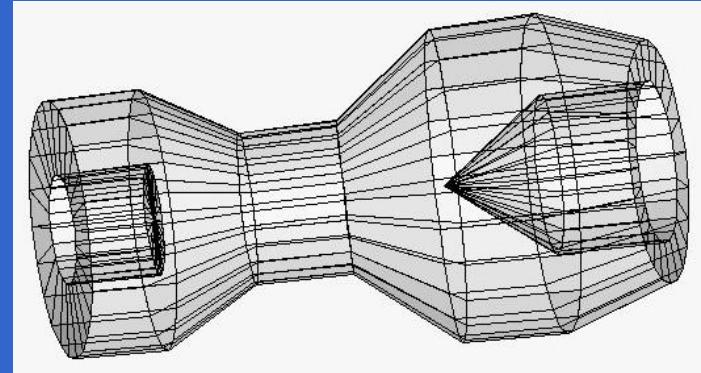
CSG: G4Tubs, G4Cons

```
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
```

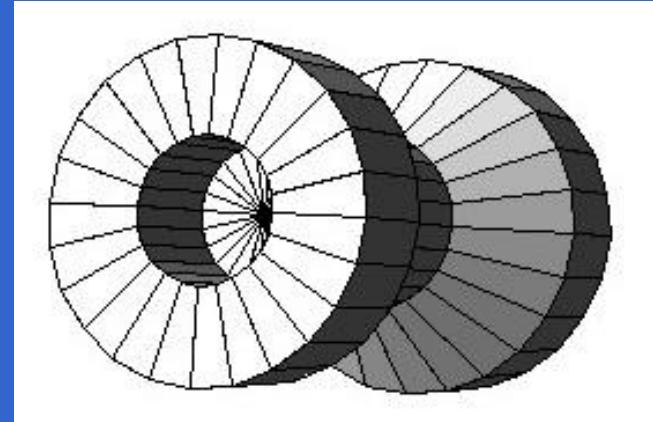
```
G4Cons(const G4String &pname, // name  
        G4double pRmin1, // inner radius -pDz  
        G4double pRmax1, // outer radius -pDz  
        G4double pRmin2, // inner radius +pDz  
        G4double pRmax2, // outer radius +pDz  
        G4double pDz, // z half length  
        G4double pSphi,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

Specific CSG Solids: G4Polycone

```
G4Polycone(const G4String& pName,  
           G4double phiStart,  
           G4double phiTotal,  
           G4int numRZ,  
           const G4double r[],  
           const G4double z[]);
```

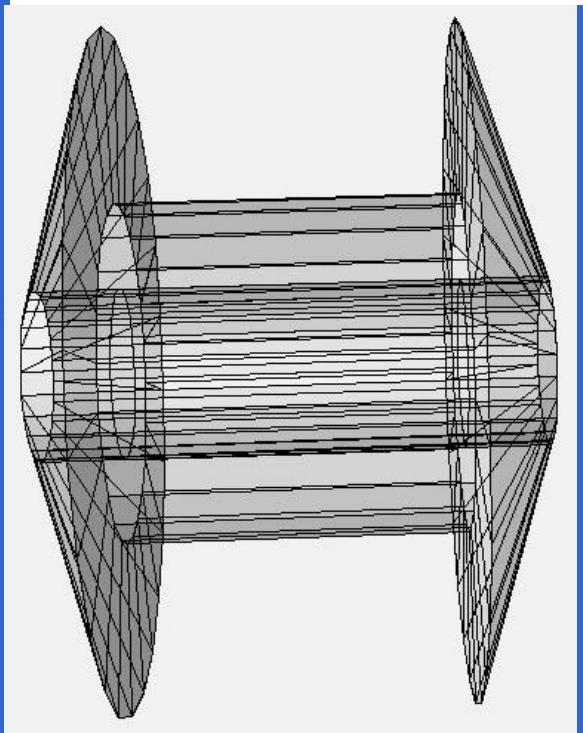
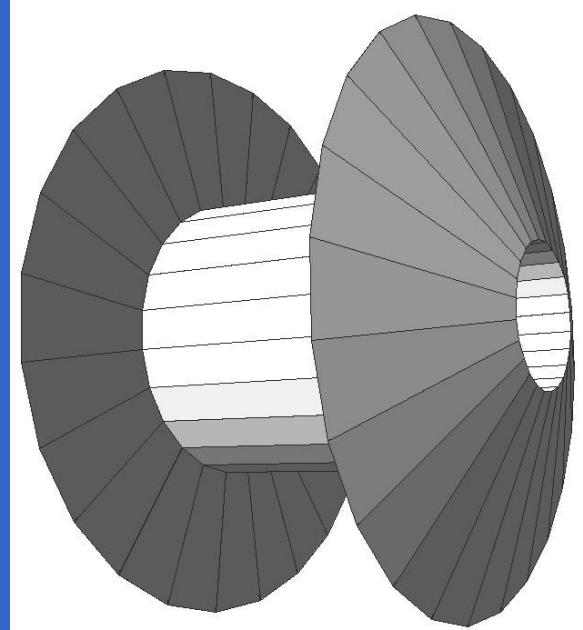


- **numRZ** - numbers of corners in the **r**, **z** space
- **r**, **z** - coordinates of corners
- Additional constructor using planes



BREP Solids

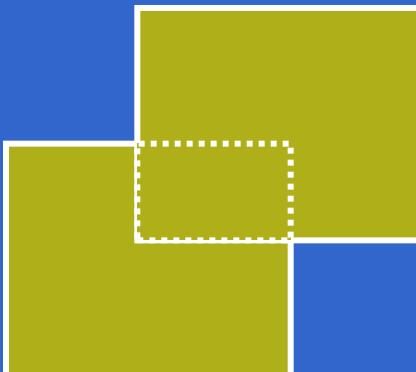
- ▶ *BREP = Boundary REPresented Solid*
- ▶ Listing all its surfaces specifies a solid
 - ▶ e.g. 6 planes for a cube
- ▶ Surfaces can be
 - ▶ planar, 2nd or higher order
 - ▶ elementary BREPS
 - ▶ Splines, B-Splines,
 - ▶ NURBS (Non-Uniform B-Splines)
 - ▶ advanced BREPS
- ▶ Few elementary BREPS pre-defined
 - ▶ box, cons, tubs, sphere, torus, polycone, polyhedra
- ▶ Advanced BREPS built through CAD systems



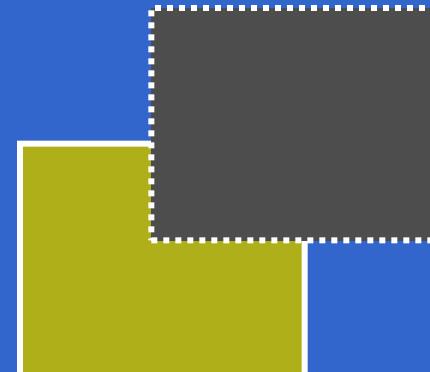
Boolean Solids

- › 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 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 solids

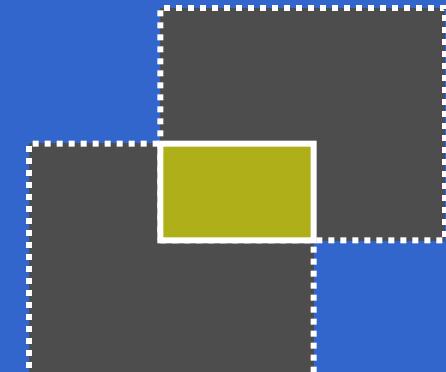
G4UnionSolid



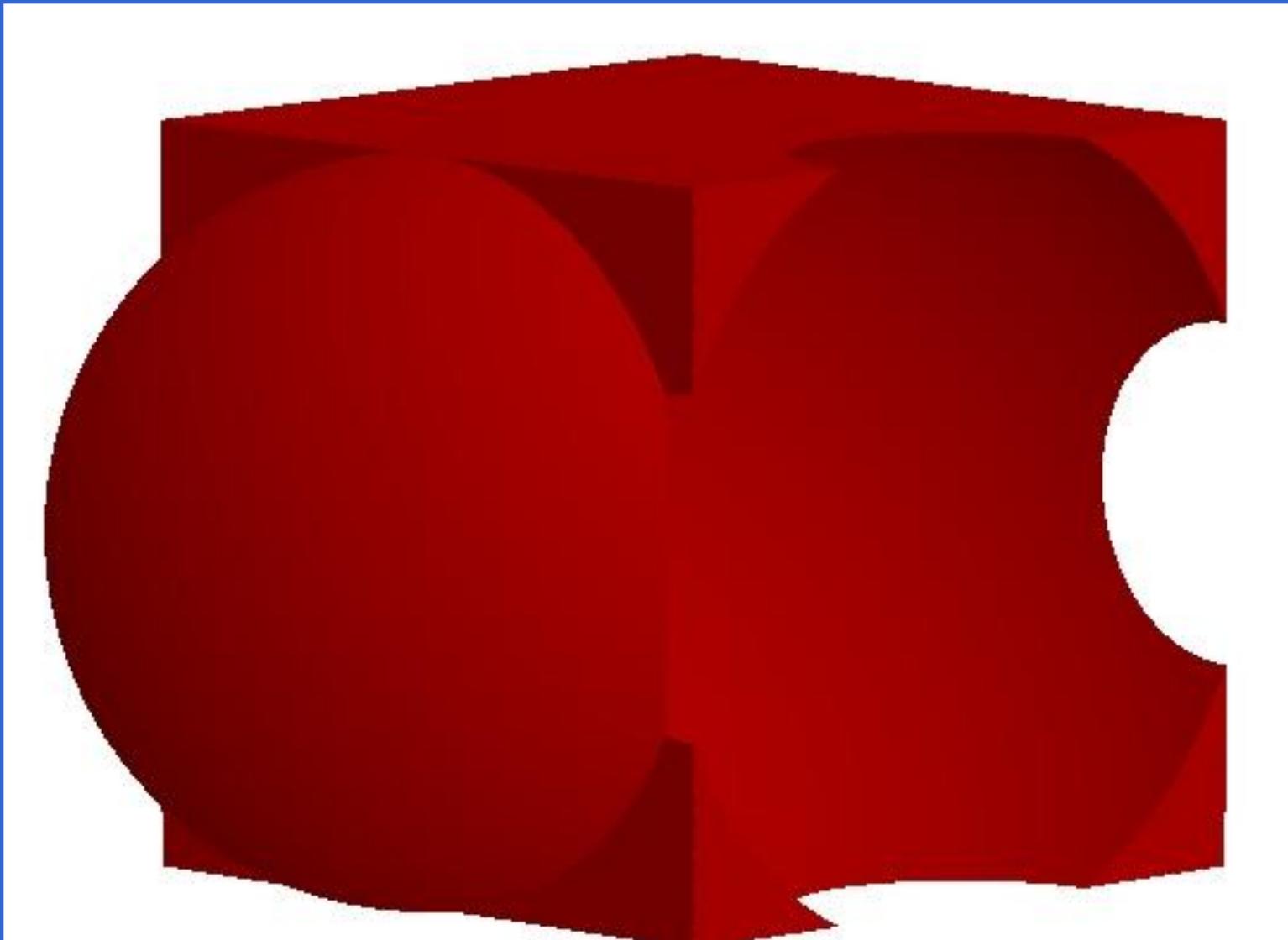
G4SubtractionSolid



G4IntersectionSolid



Boolean solid



Boolean Solids - example

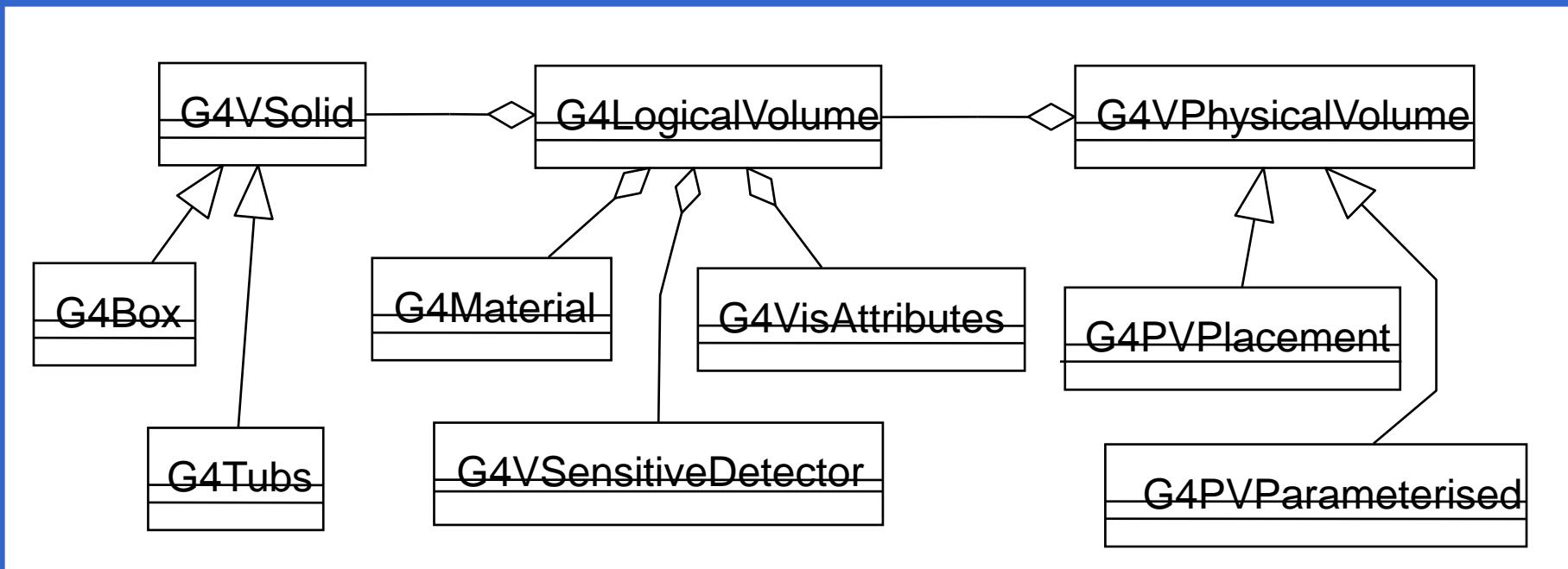
```
G4VSolid* box = new G4Box("Box",50*cm,60*cm,40*cm) ;  
G4VSolid* cylinder  
= new G4Tubs("Cylinder",0.,50.*cm,50.*cm,0.,2*M_PI*rad) ;  
G4VSolid* union  
= new G4UnionSolid("Box+Cylinder", box, cylinder) ;  
G4VSolid* subtract  
= new G4SubtractionSolid("Box-Cylinder", box, cylinder,  
    0, G4ThreeVector(30.*cm,0.,0.));  
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.

Defining a geometry

Define 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 unique physical volume which represents the experimental area must exist and fully contains all other components
 - ▶ The world volume

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, rotation
 - ▶ Shape and dimension (G4VSolid)
 - ▶ Material, sensitivity, visualization attributes
 - ▶ Position of daughter volumes
 - ▶ Magnetic field, User limits, Region
 - ▶ Shower parameterization
- ▶ Physical volumes of same type can share a logical volume.
- ▶ The pointers to solid and material must **NOT** be null
- ▶ It is not meant to act as a base class

Visualization attributes

- ▶ Each logical volume can have an associated `G4VisAttributes` object
 - ▶ Visibility, visibility of daughter volumes
 - ▶ Color, line style, line width
 - ▶ Force flag to wire-frame or solid-style mode
- ▶ For parameterized volumes, attributes can be dynamically assigned to the logical volume
 - ▶ indexed by the copy number
- ▶ Lifetime of visualization attributes must be at least as long as the objects they are assigned to

Physical volume

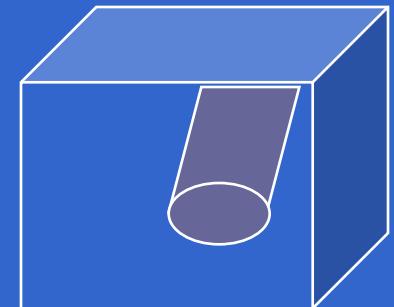
- ▶ **G4PVPlacement** 1 Placement = One Volume
 - ▶ A volume instance positioned once in its mother volume
- ▶ **G4PVParameterised** 1 Parameterized = Many Volumes
 - ▶ Parameterized by the copy number
 - ▶ Shape, size, material, position and rotation can be parameterized, by implementing a concrete 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.

Physical volume

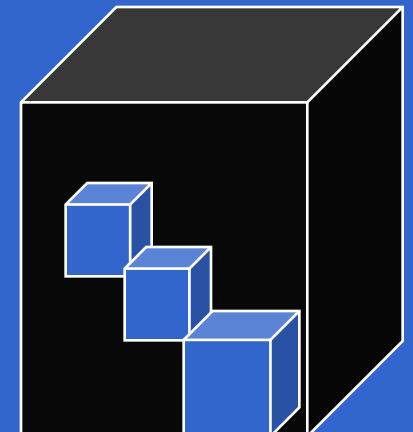
- ▶ **G4PVReplica** 1 Replica = Many Volumes
 - ▶ Mother is filled by daughters of same shape
- ▶ **G4ReflectionFactory** 1 Placement = a set of Volumes
 - ▶ generating placements of a volume and its reflected volume
 - ▶ Useful typically for end-cap calorimeter
- ▶ **G4AssemblyVolume** 1 Placement = a set of Placements
 - ▶ Position a group of volumes

Physical Volumes

- ▶ Placement: it is one positioned volume
- ▶ Repeated: a volume placed many times
 - ▶ can represent any number of volumes
 - ▶ reduces use of memory.
 - ▶ Parameterised
 - ▶ repetition w.r.t. copy number
 - ▶ Replica
 - ▶ simple repetition, similar to G3 divisions
 - ▶ it is not slicing but filling a mother volume with daughters of same shape
- ▶ A mother volume can contain **either**
 - ▶ many placement volumes
 - ▶ **or**, one repeated volume



placement



repeated

G4PVPlacement

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

- ▶ Single volume positioned relatively to the mother volume
 - ▶ In a frame rotated and translated relative to the coordinate system of the mother volume
- ▶ Three additional constructors:
 - ▶ Using **G4Transform3D** instead of rotation matrix and transformation vector to represent the direct rotation and translation of the daughter solid instead of the mother frame
 - ▶ A simple variation: specifying the mother volume as a pointer to its physics volume instead of its logical volume.
 - ▶ The combination of the two variants above

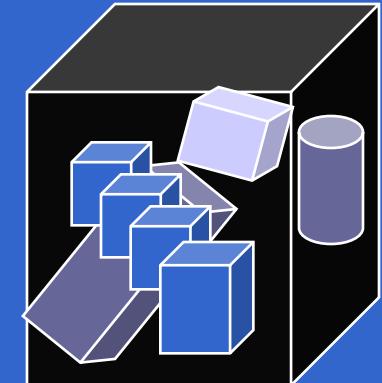
G4PVParameterised

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

- ▶ Replicates the volume **nReplicas** times using the parameterisation **pParam**, within the mother volume **pMother**
- ▶ **pAxis** is a **suggestion** to the navigator along which Cartesian axis replication of parameterized volumes dominates
- ▶ As mentioned previously, G4PVParameterised is a kind of G4VPhysicalVolume.
 - ▶ By one single object, this object represents many volumes as a function of copy number.

Parameterised Physical Volumes

- ▶ User should implement a class derived from **G4VPVParameterisation** abstract base class and define followings **as a function of copy number**
 - ▶ the size of the solid (dimensions)
 - ▶ where it is positioned (transformation, rotation)
- ▶ Optional:
 - ▶ the type of the solid
 - ▶ the material
- ▶ 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



G4PVParameterized : example

```
G4VSolid* solidChamber =
    new G4Box("chamber", 100*cm, 100*cm, 10*cm);

G4LogicalVolume* logicChamber =
    new G4LogicalVolume
    (solidChamber, ChamberMater, "Chamber", 0, 0, 0);

G4VPVParameterisation* chamberParam =
    new ChamberParameterisation();

G4VPhysicalVolume* physChamber =
    new G4PVParameterised("Chamber", logicChamber,
    logicMother, kZAxis, NbOfChambers, chamberParam);
```

G4VPVParameterisation : example

```
class ChamberParameterisation
: public G4VPVParameterisation
{
public:
    ChamberParameterisation();
    virtual ~ChamberParameterisation();
    virtual void ComputeTransformation
        (const G4int copyNo, G4VPhysicalVolume* physVol)
        const;
    virtual void ComputeDimensions
        (G4Box& trackerLayer, const G4int copyNo,
        const G4VPhysicalVolume* physVol) const;
...
```

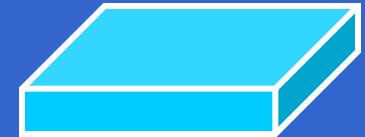
G4VPVParameterisation : example

```
void ChamberParameterisation::ComputeTransformation
(const G4int copyNo, G4VPhysicalVolume* physVol) const
{
    G4double Xposition = ... // w.r.t. copyNo
    G4ThreeVector origin(Xposition,Yposition,Zposition);
    physVol->SetTranslation(origin);
    physVol->SetRotation(0);
}
```

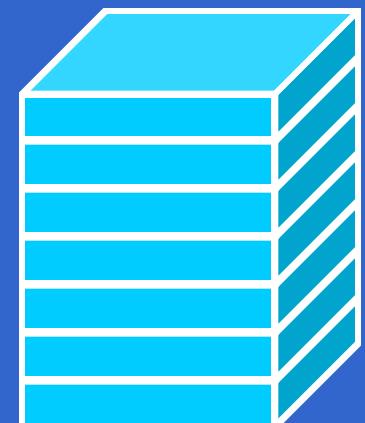
```
void ChamberParameterisation::ComputeDimensions
(G4Box& trackerChamber, const G4int copyNo,
 const G4VPhysicalVolume* physVol) const
{
    G4double XhalfLength = ... // w.r.t. copyNo
    trackerChamber.SetXHalfLength(XhalfLength);
    trackerChamber.SetYHalfLength(YhalfLength);
    trackerChamber.SetZHalfLength(ZhalfLength);
}
```

Replicated Physical Volumes

- ▶ The mother volume is **completely filled** with replicas, all of which are the **same size and shape**.
- ▶ As mentioned previously, **G4PVReplica** is a kind of **G4VPhysicalVolume**.
 - ▶ By one single object, this object represents many daughter volumes as a function of copy number.
- ▶ 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

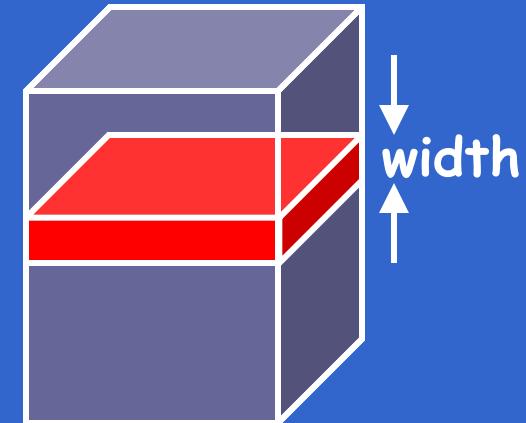
G4PVReplica

```
G4PVReplica(const G4String &pName,  
            G4LogicalVolume *pLogical,  
            G4LogicalVolume *pMother,  
            const EAxis pAxis,  
            const G4int nReplicas,  
            const G4double width,  
            const G4double offset=0);
```

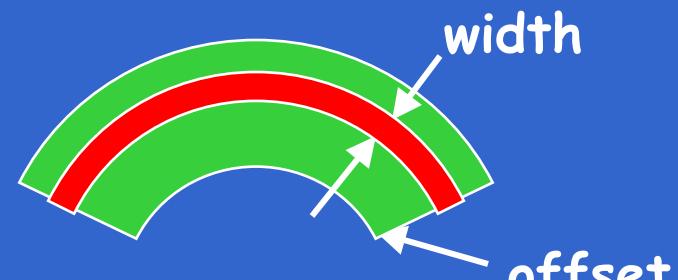
- ▶ `offset` may be used only for tube/cone segment
- ▶ Features and restrictions:
 - ▶ Replicas can be placed inside other replicas
 - ▶ Normal placement volumes can be placed inside replicas, assuming no intersection/overlaps with the mother volume or with other replicas
 - ▶ No volume can be placed inside a **radial** replication
 - ▶ Parameterised volumes **cannot** be placed inside a replica

Replica - axis, width, offset

- ▶ Cartesian axes - **kXaxis**, **kYaxis**, **kZaxis**
 - ▶ offset shall not be used
 - ▶ Center of n-th daughter is given as
$$-width * (nReplicas-1) * 0.5 + n * width$$



- ▶ Radial axis - **kRaxis**
 - ▶ Center of n-th daughter is given as
$$width * (n+0.5) + offset$$



- ▶ Phi axis - **kPhi**
 - ▶ Center of n-th daughter is given as
$$width * (n+0.5) + offset$$



G4PVReplica : example

```
G4double tube_dPhi = 2.* M_PI * rad;
G4VSolid* tube =
    new G4Tubs("tube",20*cm,50*cm,30*cm,0.,tube_dPhi);
G4LogicalVolume * tube_log =
    new G4LogicalVolume(tube, Ar, "tubeL", 0, 0, 0);
G4VPhysicalVolume* tube_phys =
    new G4PVPlacement(0,G4ThreeVector(-200.*cm,0.,0.),
                      "tubeP", tube_log, world_phys, false, 0);
G4double divided_tube_dPhi = tube_dPhi/6.;
G4VSolid* div_tube =
    new G4Tubs("div_tube", 20*cm, 50*cm, 30*cm,
               -divided_tube_dPhi/2., divided_tube_dPhi);
G4LogicalVolume* div_tube_log =
    new G4LogicalVolume(div_tube,Ar,"div_tubeL",0,0,0);
G4VPhysicalVolume* div_tube_phys =
    new G4PVReplica("div_tube_phys", div_tube_log,
                    tube_log, kPhi, 6, divided_tube_dPhi);
```

Computing volumes and masses

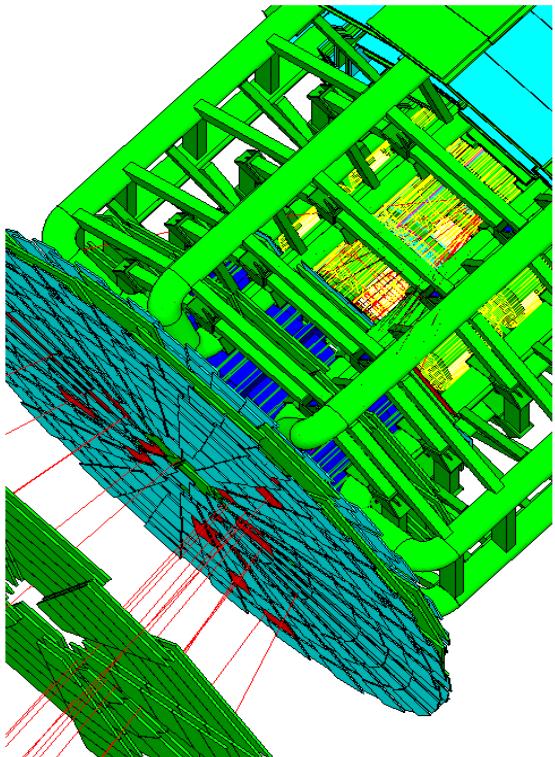
- ▶ Geometrical volume of a generic solid or boolean composition can be computed from the **solid**:

```
G4double GetCubicVolume();
```

- ▶ Exact volume is determinatively calculated for most of CSG solids, while estimation based on Monte Carlo integration is given for other solids.
- ▶ Overall mass of a geometry setup (subdetector) can be computed from the **logical volume**:

```
G4double GetMass(G4bool forced=false,  
                  G4bool propagate=true, G4Material* pMaterial=0);
```

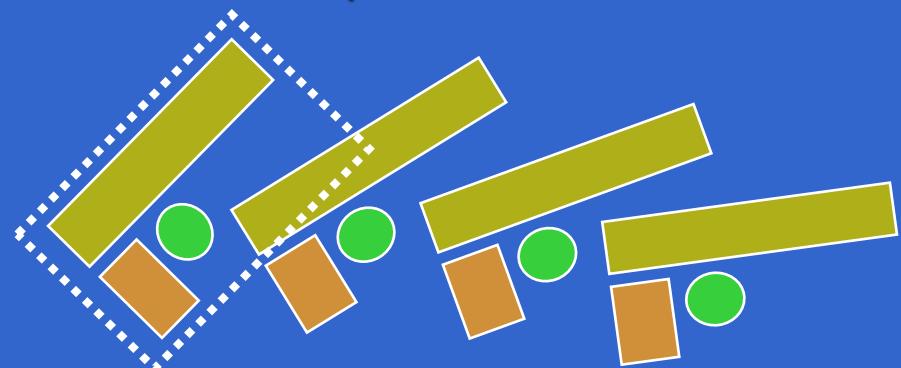
- ▶ The computation may require a considerable amount of time, depending on the complexity of the geometry.
- ▶ The return value is cached and reused until *forced*=true.
- ▶ Daughter volumes will be neglected for *propagate*=false.



Defining a geometry
advanced features

Grouping volumes

- ▶ To represent a regular pattern of positioned volumes, composing a more or less complex structure
 - ▶ structures which are hard to describe with simple replicas or parameterised volumes
 - ▶ structures which may consist of different shapes
 - ▶ Too densely positioned to utilize a mother volume
- ▶ Assembly volume
 - ▶ acts as an *envelope* for its daughter volumes
 - ▶ its role is over once its logical volume has been placed
 - ▶ daughter physical volumes become independent copies in the final structure
- ▶ Participating daughter logical volumes are treated as triplets
 - ▶ logical volume
 - ▶ translation w.r.t. envelop
 - ▶ rotation w.r.t. envelop



G4AssemblyVolume

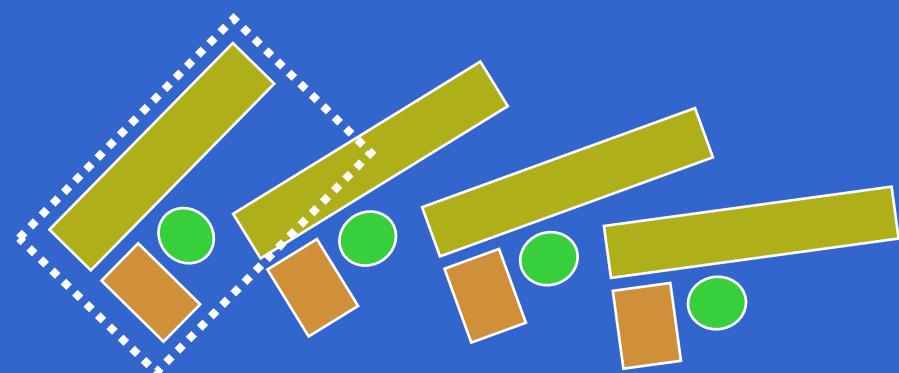
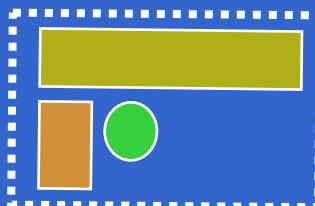
G4AssemblyVolume::AddPlacedVolume

```
( G4LogicalVolume* volume,  
    G4ThreeVector& translation,  
    G4RotationMatrix* rotation );
```

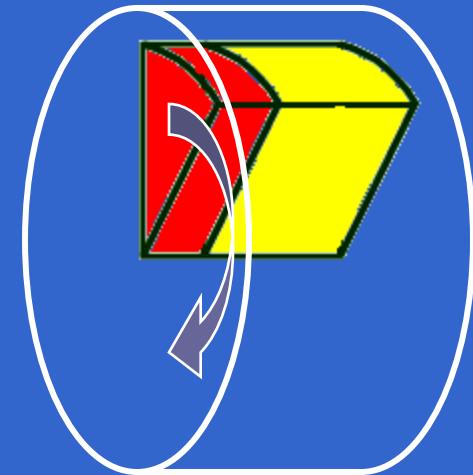
- ▶ Helper class to combine daughter logical volumes in arbitrary way
 - ▶ Imprints of the assembly volume are made inside a mother logical volume through **G4AssemblyVolume::MakeImprint(...)**
 - ▶ Each physical volume name is generated automatically
 - ▶ Format: **av_www_impr_xxx_YYY_ZZZ**
 - ▶ **www** – assembly volume instance number
 - ▶ **xxx** – assembly volume imprint number
 - ▶ **YYY** – name of the placed logical volume in the assembly
 - ▶ **zzz** – index of the associated logical volume
 - ▶ Generated physical volumes (and related transformations) are automatically managed (creation and destruction)

G4AssemblyVolume : example

```
G4AssemblyVolume* assembly = new G4AssemblyVolume();  
G4RotationMatrix Ra;  
G4ThreeVector Ta;  
Ta.setX(...); Ta.setY(...); Ta.setZ(...);  
assembly->AddPlacedVolume( plateLV, Ta, Ra );  
... // repeat placement for each daughter  
  
for( unsigned int i = 0; i < layers; i++ ) {  
    G4RotationMatrix Rm(...);  
    G4ThreeVector Tm(...);  
    assembly->MakeImprint( worldLV, Tm, Rm );  
}
```



Reflecting solids



- ▶ Let's take an example of a pair of endcap calorimeters.
 - ▶ They are mirror symmetric to each other.
 - ▶ Such geometry cannot be made by parallel transformation or 180 degree rotation
- ▶ **G4ReflectedSolid** (derived from G4VSolid)
 - ▶ Utility class representing a solid shifted from its original reference frame to a new **mirror symmetric** one
 - ▶ The reflection (G4Reflect[X/Y/Z]3D) is applied as a decomposition into rotation and translation
- ▶ **G4ReflectionFactory**
 - ▶ Singleton object using G4ReflectedSolid for generating placements of reflected volumes
- ▶ Reflections are currently limited to simple CSG solids
 - ▶ will be extended soon to all solids

Reflecting hierarchies of volumes - 1

```
G4PhysicalVolumesPair G4ReflectionFactory::Place  
(const G4Transform3D& transform3D, // the transformation  
const G4String& name, // the name  
G4LogicalVolume* LV, // the logical volume  
G4LogicalVolume* motherLV, // the mother volume  
G4bool noBool, // currently unused  
G4int copyNo) // optional copy number
```

- ▶ Used for normal placements:
 - i. Performs the transformation decomposition
 - ii. Generates a new reflected solid and logical volume
 - Retrieves it from a map if the reflected object is already created
 - iii. Transforms any daughter and places them in the given mother
 - iv. Returns a pair of physical volumes, the second being a placement in the reflected mother
- ▶ **G4PhysicalVolumesPair** is
std::map<G4VPhysicalVolume*, G4VPhysicalVolume*>

Reflecting hierarchies of volumes - 2

G4PhysicalVolumesPair G4ReflectionFactory::Replicate

```
(const G4String& name,           // the actual name  
G4LogicalVolume* LV,            // the logical volume  
G4LogicalVolume* motherLV,      // the mother volume  
Eaxis                  axis    // axis of replication  
G4int                  replicaNo // number of replicas  
G4int                  width,    // width of single replica  
G4int                  offset=0) // optional mother offset
```

- Creates replicas in the given mother volume
- Returns a pair of physical volumes, the second being a replica in the reflected mother

GGE (Graphical Geometry Editor)

- › Implemented in JAVA, GGE is a graphical geometry editor compliant to Geant4.

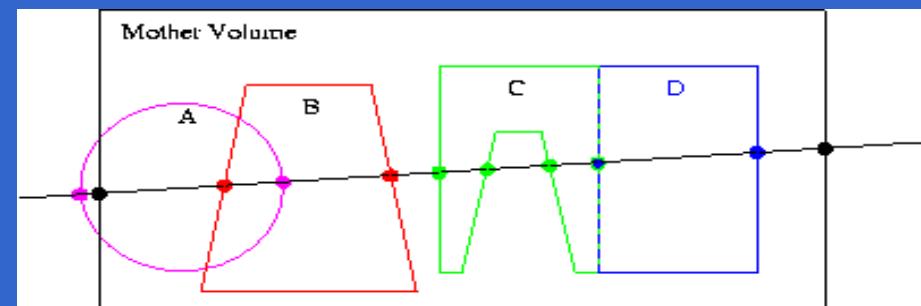
It allows to:

- › Describe a detector geometry including:
 - › materials, solids, logical volumes, placements
- › Graphically visualize the detector geometry using a Geant4 supported visualization system
- › Store persistently the detector description
- › Generate the C++ code according to the Geant4 specifications
- › GGE can be downloaded from Web as a separate tool:
 - <http://erpc1.naruto-u.ac.jp/~geant4/>

Geometry checking tools

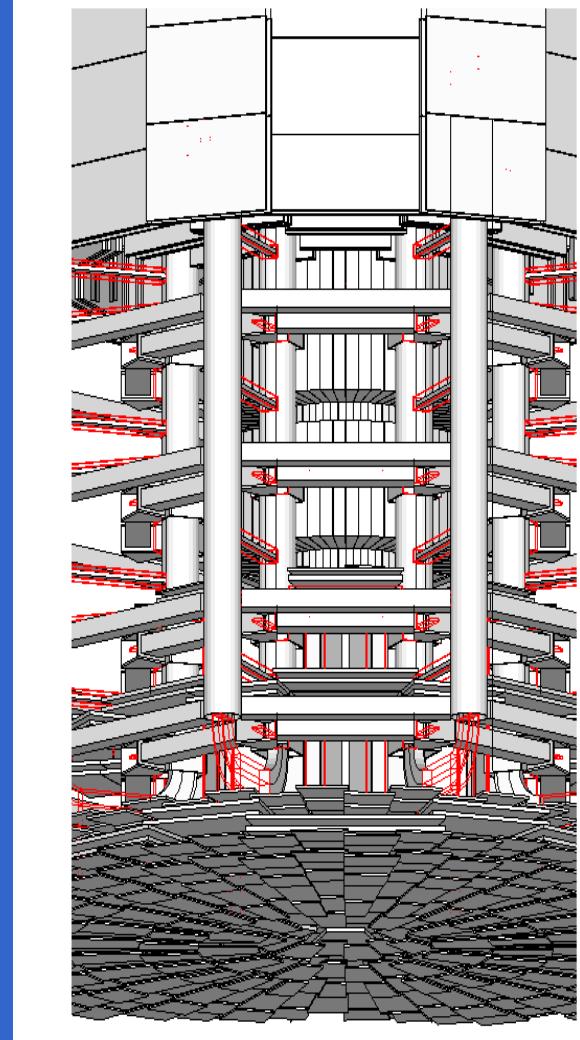
Debugging geometries

- ▶ An *overlapping* volume is a contained volume which actually *protrudes* from its mother volume
 - ▶ Volumes are also often positioned in a same volume with the intent of not provoking intersections between themselves. When volumes in a common mother actually *intersect themselves* are defined as overlapping
- ▶ Geant4 *does not allow* for malformed geometries
- ▶ The problem of detecting overlaps between volumes is bounded by the complexity of the solid models description
- ▶ Utilities are provided for detecting wrong positioning
 - ▶ Graphical tools (DAVID, OLAP)
 - ▶ Kernel run-time commands



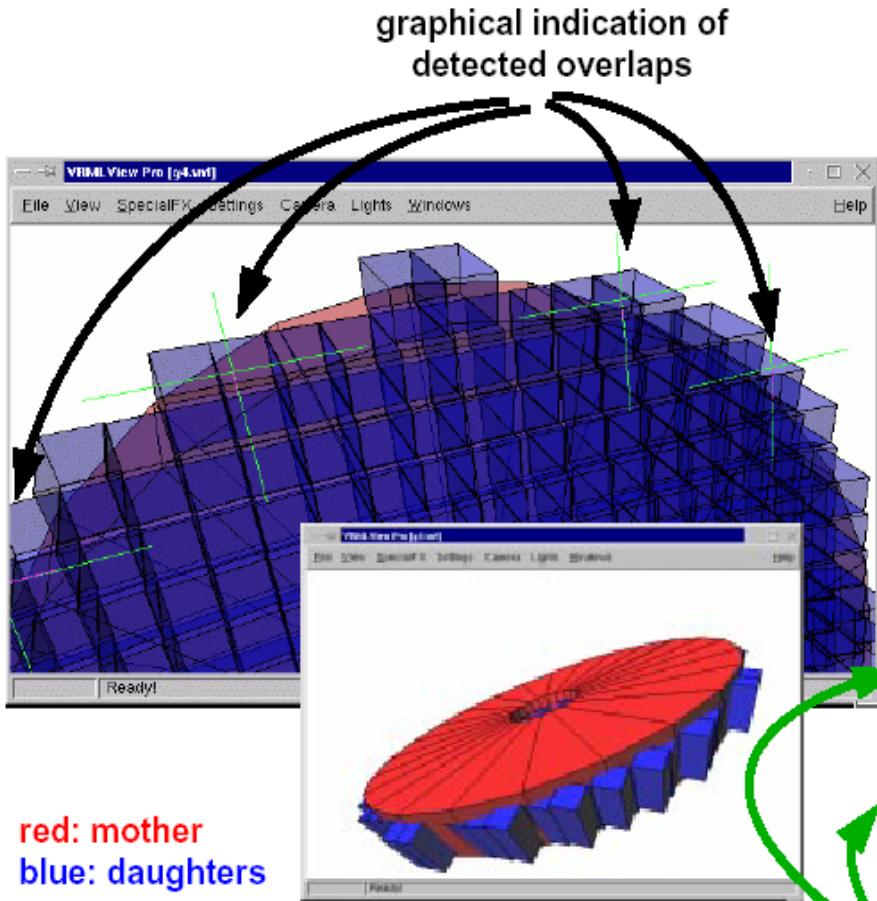
Debugging tools: DAVID

- DAVID is a graphical debugging tool for detecting potential intersections of volumes
- Accuracy of the graphical representation can be tuned to the exact geometrical description.
 - physical-volume surfaces are automatically decomposed into 3D polygons
 - intersections of the generated polygons are parsed.
 - If a polygon intersects with another one, the physical volumes associated to these polygons are highlighted in color (**red** is the default).
- DAVID can be downloaded from the Web as external tool for Geant4
 - <http://arkoop2.kek.jp/~tanaka/DAWN/>
[About_DAVID.html](#)



Debugging tools: OLAP

- Stand-alone batch application
 - Provided as extended example
 - Can be combined with a graphical environment and GUI



daughters are protruding their mother

Geant4 Macro:

```
/vis/scene/create
/vis/sceneHandler/create VRML2FILE
/vis/viewer/create
/olap/goto ECalEnd
/olap/grid 7 7 7
/olap/trigger
/vis/viewer/update
```

Output:

```
delta=59.3416
vol 1: point=(560.513,1503.21,-141.4)
vol 2: point=(560.513,1443.86,-141.4)
A -> B:
[0]: ins=[2] PVName=[NewWorld:0] Type=[N] ...
[1]: ins=[0] PVName=[ECalEndcap:0] Type=[N] ...
[2]: ins=[1] PVName=[ECalEndcap07:38] Type=[N]
B -> A:
[0]: ins=[2] PVName=[NewWorld:0] Type=[N] ...
```

NavigationHistories of points of overlap
(including: info about translation, rotation, solid specs)

Debugging run-time commands

- ▶ Built-in run-time commands to activate verification tests for the user geometry are defined
 - `geometry/test/run` or `geometry/test/grid_test`
- ▶ to start verification of geometry for overlapping regions based on a standard grid setup, limited to the first depth level
 - `geometry/test/recursive_test`
- ▶ applies the grid test to all depth levels (may require lots of CPU time!)
 - `geometry/test/cylinder_test`
- ▶ shoots lines according to a cylindrical pattern
 - `geometry/test/line_test`
- ▶ to shoot a line along a specified direction and position
 - `geometry/test/position`
- ▶ to specify position for the `line_test`
 - `geometry/test/direction`
- ▶ to specify direction for the `line_test`

Debugging run-time commands

- ▶ Example layout:

GeomTest: no daughter volume extending outside mother detected.

GeomTest Error: Overlapping daughter volumes

The volumes Tracker[0] and Overlap[0],
both daughters of volume World[0],
appear to overlap at the following points in global coordinates: (list truncated)

length (cm)	----- start position (cm) -----	----- end position (cm) -----
240	-240	-145.5
		-145.5
	0	-145.5
		-145.5

Which in the mother coordinate system are:

length (cm)	----- start position (cm) -----	----- end position (cm) -----
.	.	.

Which in the coordinate system of Tracker[0] are:

length (cm)	----- start position (cm) -----	----- end position (cm) -----
.	.	.

Which in the coordinate system of Overlap[0] are:

length (cm)	----- start position (cm) -----	----- end position (cm) -----
.	.	.

Visualizing detector geometry tree

- Built-in commands defined to display the hierarchical geometry tree
 - As simple ASCII text structure
 - Graphical through GUI (combined with GAG)
 - As XML exportable format
- Implemented in the visualization module
 - As an additional graphics driver
- G3 DTREE capabilities provided and more

