Geant 4

Detector Description

Gabriele.Cosmo@cern.ch

http://cern.ch/geant4

Detector Description

Part I *The Basics*

Part II *Logical and physical volumes*

Part III *Solids, touchables*

Part IV *Visualization attributes & Optimization technique*

Part V *Advanced features*

PART I

Detector Description: the Basics

Materials

- *The System of units & constants*
- *Definition of elements*
- *Materials and mixtures*
- *Some examples ...*

Unit system

- Geant4 has no default unit. To give a number, unit must be "multiplied" to the number.
  - for example :

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

  - If no unit is specified, the *internal* G4 unit will be used, but <u>this is discouraged</u> !
  - Almost all commonly used units are available.
  - The user can define new units.
  - Refer to CLHEP: `SystemOfUnits.h`
- Divide a variable by a unit you want to get.

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

System of Units

- System of units are defined in CLHEP, based on:
  - millimetre (**`mm`**), nanosecond (**`ns`**), Mega eV (**`MeV`**), positron charge (**`eplus`**) degree Kelvin (**`kelvin`**), the amount of substance (**`mole`**), luminous intensity (**`candela`**), radian (**`radian`**), steradian (**`steradian`**)
- All other units are computed from the basic ones.
- In output, Geant4 can choose the most appropriate unit to use. Just specify the *category* for the data (`Length`, `Time`, `Energy`, etc…):

```
G4cout << G4BestUnit(StepSize, “Length”);
```

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

Defining new units

- New units can be defined directly as constants, or (suggested way) via `G4UnitDefinition`.
  - `G4UnitDefinition ( name, symbol, category, value )`
- Example (mass thickness):
  - ```
    G4UnitDefinition ("grammpercm2", "g/cm2",
                      "MassThickness", g/cm2);
    ```
  - The new category "MassThickness" will be registered in the kernel in **G4UnitsTable**
- To print the list of units:
  - From the code
    ```
    G4UnitDefinition::PrintUnitsTable();
    ```
  - At run-time, as UI command:
    ```
    Idle> /units/list
    ```
    ```

Definition of Materials

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

Isotopes, Elements and Materials

- **`G4Isotope`** and **`G4Element`** describe the properties of the *atoms*:
  - Atomic number, number of nucleons, mass of a mole, shell energies
  - Cross-sections per atoms, etc...
- **`G4Material`** describes the *macroscopic* properties of the matter:
  - temperature, pressure, state, density
  - Radiation length, absorption length, etc...

Elements & Isotopes

- Isotopes can be assembled into elements

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

- ... building elements as follows:

```
G4Element (const G4String& name,
           const G4String& symbol, // element symbol
                 G4int     nIso ); // # of isotopes
G4Element::AddIsotope(G4Isotope* iso,  // isotope
                      G4double relAbund); // fraction of atoms
                                          // per volume
```

Material of one element

- Single element material

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

- Prefer low-density material to vacuum

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);
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);
Air->AddElement(elN, 70.0*perCent);
Air->AddElement(elO, 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);
```

Example: gas

- It may be necessary to specify temperature and pressure
  - (dE/dx computation affected)

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

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

Example: vacuum

- Absolute vacuum does not exist. It is a gas at very low density !
  - Cannot define materials composed of multiple elements through Z or A, or with $\rho = 0$.

```
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);
```

Describing a detector

- *Detector geometry modeling*
- *The basic concepts: solids & volumes*

Describe your detector

- Derive your own concrete class from `G4VUserDetectorConstruction` abstract base class.
- Implementing the method `Construct()`:
  - Modularize it according to each detector component or sub-detector:
    - Construct all necessary materials
    - Define shapes/solids required to describe the geometry
    - Construct and place volumes of your detector geometry
    - Define sensitive detectors and identify detector volumes which to associate them
    - Associate magnetic field to detector regions
    - Define visualization attributes for the detector elements

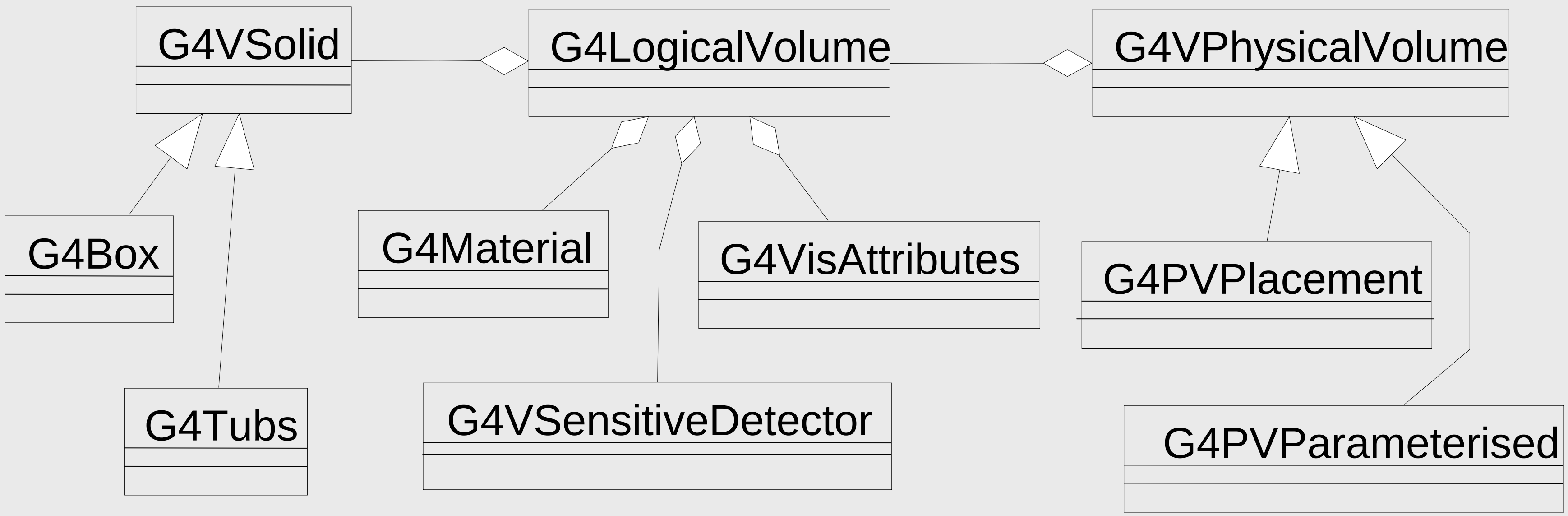
Creating a Detector Volume

- Start with its Shape & Size
  - Box 3x5x7 cm, sphere R=8m
- Add properties:
  - material, B/E field,
  - make it sensitive
- Place it in another volume
  - in one place
  - repeatedly using a function

- *Solid*
- *Logical-Volume*
- *Physical-Volume*

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
  - <u>The world volume</u>

PART II

Detector Description:

Logical and Physical Volumes

G4LogicalVolume

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

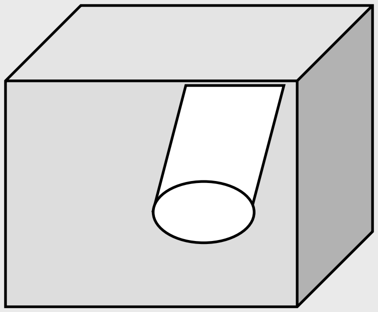
- Contains all information of volume except position:
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Position of daughter volumes
  - Magnetic field, User limits
  - Shower parameterisation
- Physical volumes of same type can share a logical volume.
- The pointers to solid and material must be NOT null
- Once created it is automatically entered in the LV store
- It is not meant to act as a base class

G4VPhysicalVolume

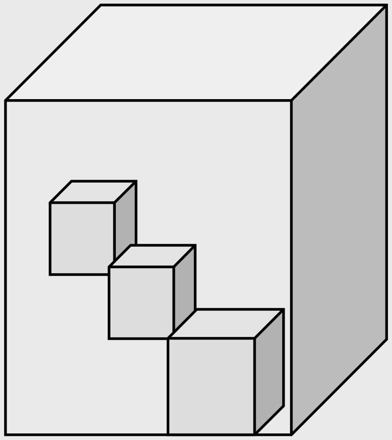
- G4PVPlacement 1 Placement = One Volume
  - A volume instance positioned once in a mother volume
- G4PVParameterised 1 Parameterised = Many Volumes
  - Parameterised by the copy number
    - Shape, size, material, position and rotation can be parameterised, by implementing a concrete class of `G4VPVParameterisation`.
  - Reduction of memory consumption
    - Currently: parameterisation can be used only for volumes that either a) have no further daughters <u>or</u> b) are identical in size & shape.
- G4PVReplica 1 Replica = Many Volumes
  - Slicing a volume into smaller pieces (if it has a symmetry)

Physical Volumes

- Placement: it is one positioned volume
- Repeated: a volume placed many times
  - can represent any number of volumes
  - reduces use of memory.
  - Replica
    - simple repetition, similar to G3 divisions
  - Parameterised
- A **mother** volume can contain **either**
  - **many placement** volumes **OR**
  - **one repeated** volume



placement



repeated

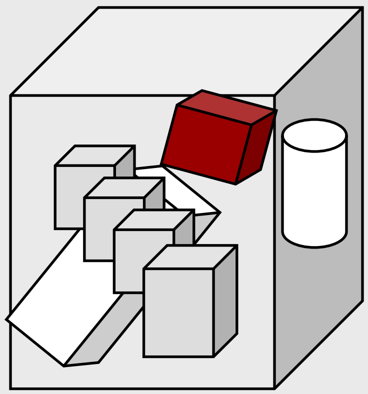
G4PVPlacement

```
G4PVPlacement(G4RotationMatrix* pRot,
              const G4ThreeVector& tlate,
              G4LogicalVolume* pCurrentLogical,
              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:
  - A simple variation: specifying the mother volume as a pointer to its physical volume instead of its logical volume.
  - Using `G4Transform3D` to represent the direct rotation and translation of the solid instead of the frame
  - The combination of the two variants above

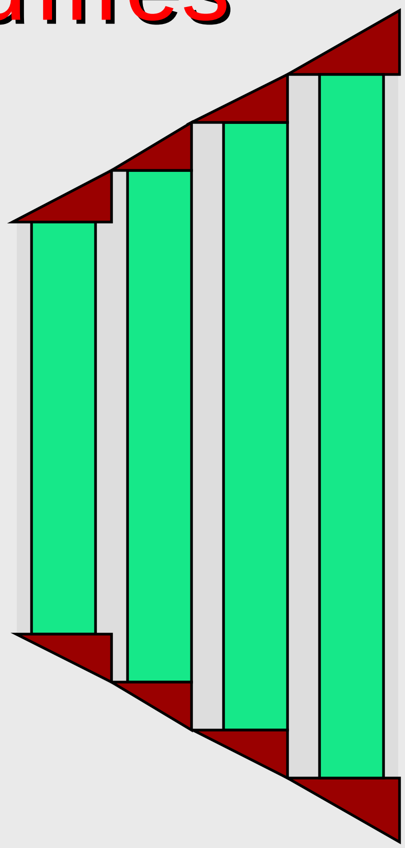
Parameterised Physical Volumes

- User written functions define:
  - the size of the solid (dimensions)
    - Function `ComputeDimensions(…)`
  - where it is positioned (transformation)
    - Function `ComputeTransformations(…)`
- Optional:
  - the type of the solid
    - Function `ComputeSolid(…)`
  - the material
    - Function `ComputeMaterial(…)`
- Limitations:
  - Applies to simple CSG solids only
  - Daughter volumes allowed only for special cases
- Very powerful
  - Consider parameterised volumes as “leaf” volumes



Uses of Parameterised Volumes

- Complex detectors
  - with large repetition of volumes
    - regular or irregular
- Medical applications
  - the material in animal tissue is measured
    - cubes with varying material



G4PVParameterised

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

- Replicates the volume `nReplicas` times using the parameterisation `pParam`, within the mother volume
- The positioning of the replicas is dominant along the specified Cartesian axis
  - If `kUndefined` is specified as axis, 3D voxelisation for optimisation of the geometry is adopted
- Represents many touchable detector elements differing in their positioning and dimensions. Both are calculated by means of a `G4VPVParameterisation` object
- Alternative constructor using pointer to physical volume for the mother

Parameterisation
example - 1

```
G4VSolid* solidChamber = new G4Box("chamber", 100*cm, 100*cm, 10*cm);
G4LogicalVolume* logicChamber =
   new G4LogicalVolume(solidChamber, ChamberMater, "Chamber", 0, 0, 0);
G4double firstPosition = -trackerSize + 0.5*ChamberWidth;
G4double firstLength = fTrackerLength/10;
G4double lastLength  = fTrackerLength;
G4VPVParameterisation* chamberParam =
   new ChamberParameterisation( NbOfChambers, firstPosition,
                                ChamberSpacing, ChamberWidth,
                                firstLength, lastLength);
G4VPhysicalVolume* physChamber =
   new G4PVParameterised( "Chamber", logicChamber, logicTracker,
                          kZAxis, NbOfChambers, chamberParam);
```

Use **kUndefined** for activating 3D voxelisation for optimisation

Parameterisation

example - 2

```
class ChamberParameterisation : public G4VPVParameterisation
{
  public:
    ChamberParameterisation( G4int NoChambers, G4double startZ,
                             G4double spacing, G4double widthChamber,
                             G4double lenInitial, G4double lenFinal );
   ~ChamberParameterisation();
   void ComputeTransformation (const G4int copyNo,
                               G4VPhysicalVolume* physVol) const;
   void ComputeDimensions (G4Box& trackerLayer, const G4int copyNo,
                           const G4VPhysicalVolume* physVol) const;
}
```

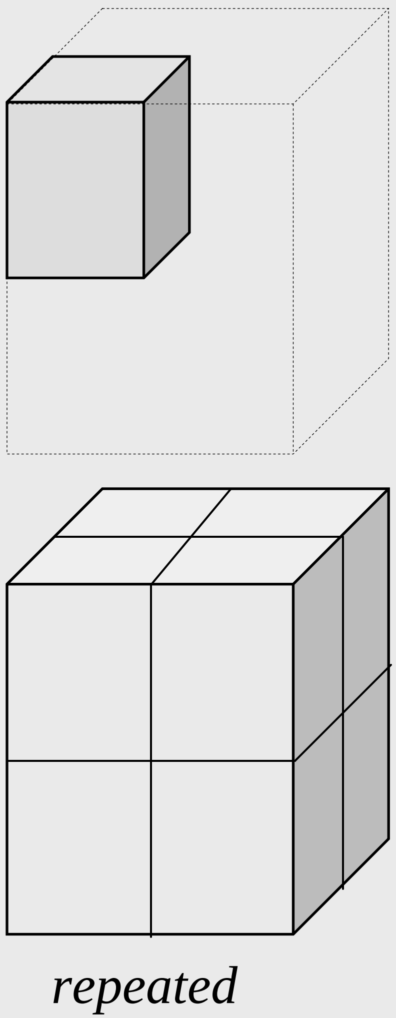
Parameterisation

example - 3

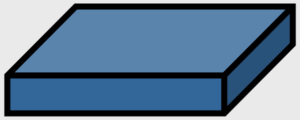
```
void ChamberParameterisation::ComputeTransformation
(const G4int copyNo, G4VPhysicalVolume* physVol) const
{
  G4double Zposition= fStartZ + (copyNo+1) * fSpacing;
  G4ThreeVector origin(0, 0, Zposition);
  physVol->SetTranslation(origin);
  physVol->SetRotation(0);
}
void ChamberParameterisation::ComputeDimensions
(G4Box& trackerChamber, const G4int copyNo,
 const G4VPhysicalVolume* physVol) const
{
  G4double  halfLength= fHalfLengthFirst + copyNo * fHalfLengthIncr;
  trackerChamber.SetXHalfLength(halfLength);
  trackerChamber.SetYHalfLength(halfLength);
  trackerChamber.SetZHalfLength(fHalfWidth);
}
```

Replicated Physical Volumes

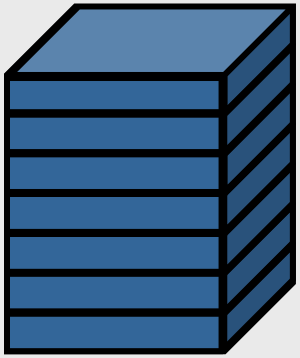
- The mother volume is sliced into replicas, all of the same size and dimensions.
- Represents many touchable detector elements differing only in their positioning.
- Replication may occur along:
  - Cartesian axes (X, Y, Z) – slices are considered perpendicular to the axis of replication
    - Coordinate 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



G4PVReplica



a daughter volume
to be replicated



mother volume

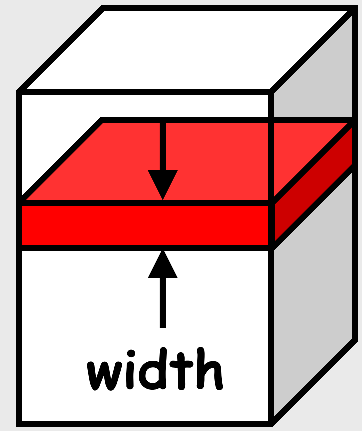
```
G4PVReplica(const G4String& pName,
            G4LogicalVolume* pCurrentLogical,
            G4LogicalVolume* pMotherLogical,
            const EAxis pAxis,
            const G4int nReplicas,
            const G4double width,
            const G4double offset=0);
```

- Alternative constructor: using pointer to physical volume for the mother
- An `offset` can only be associated to a mother offset along the axis of replication
- 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

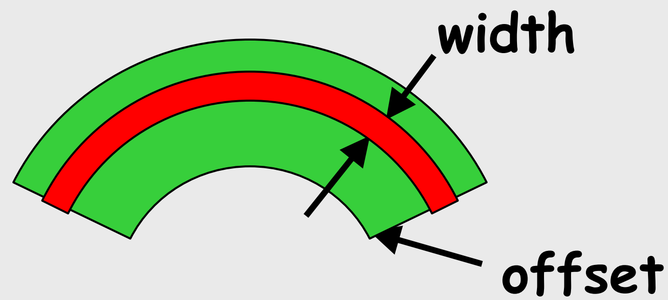
- Cartesian axes - **kXaxis, kYaxis, kZaxis**
  - offset shall not be used
  - Center of n-th daughter is given as

    -width*(nReplicas-1)*0.5+n*width



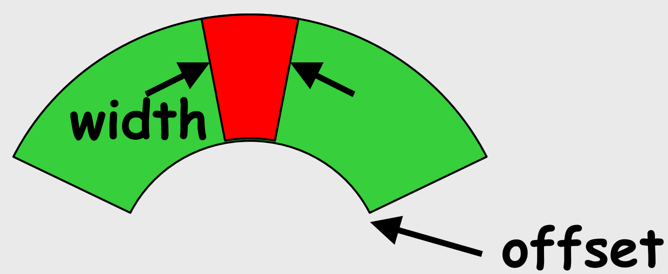
- Radial axis - **kRaxis**
  - Center of n-th daughter is given as

    width*(n+0.5)+offset



- Phi axis - **kPhi**
  - Center of n-th daughter is given as

    width*(n+0.5)+offset



Replication
example

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

Divided Physical Volumes

- Implemented as “special” kind of parameterised volumes
  - Applies to CSG-like solids only (box, tubs, cons, para, trd, polycone, polyhedra)
  - Divides a volume in identical copies along one of its axis (copies are not strictly identical)
    - e.g. - a tube divided along its radial axis
    - Offsets can be specified
- The possible axes of division vary according to the supported solid type
- Represents many touchable detector elements differing only in their positioning
- **`G4PVDivision`** is the class defining the division
  - The parameterisation is calculated automatically using the values provided in input

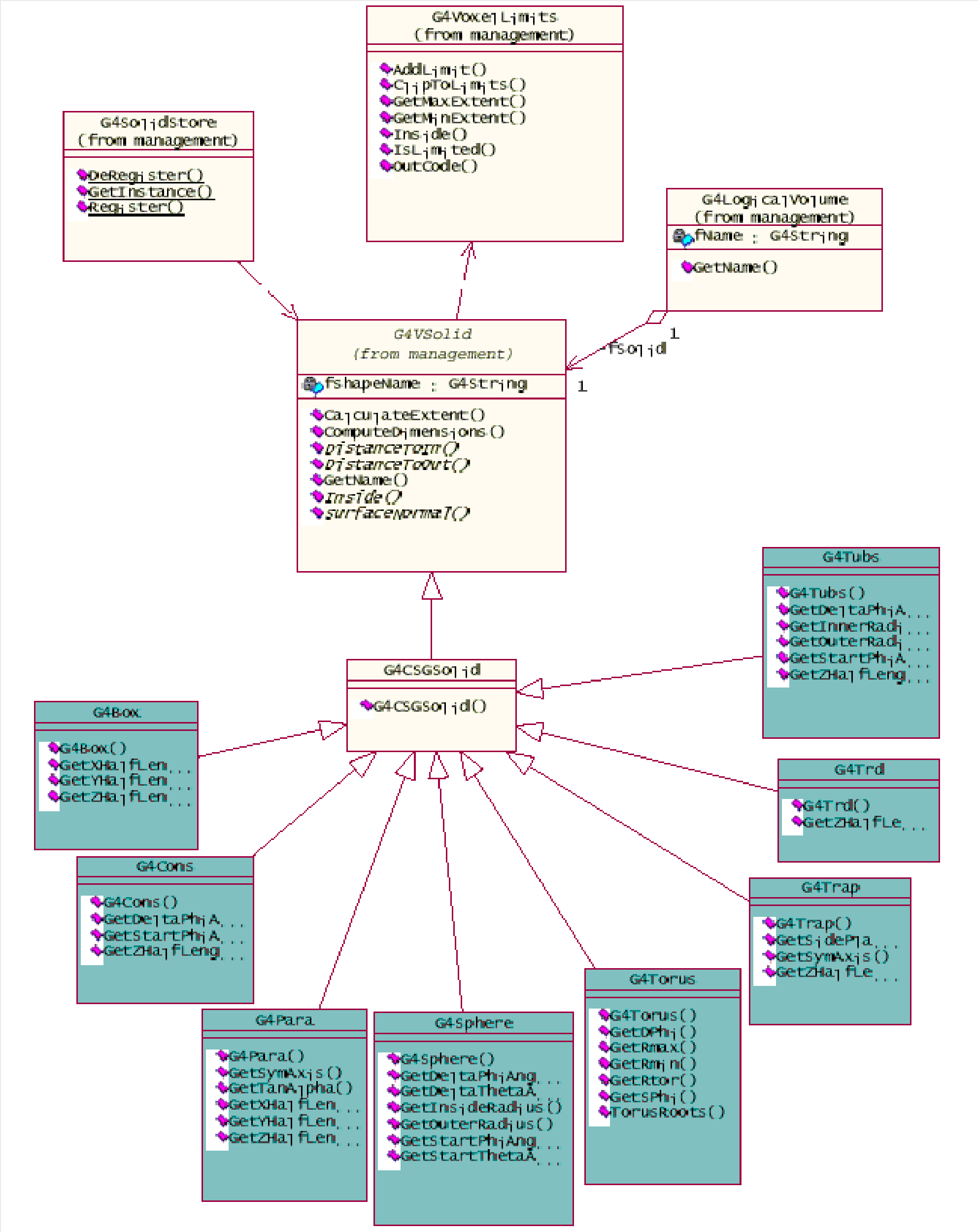
PART III

Detector Description:

Solids & Touchables

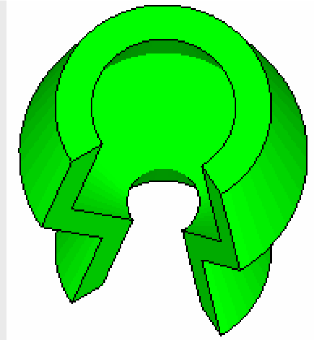
G4VSolid

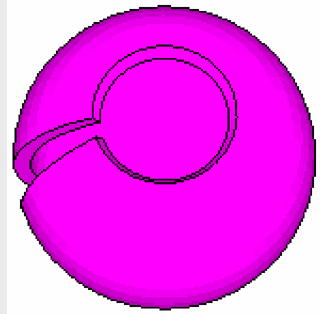
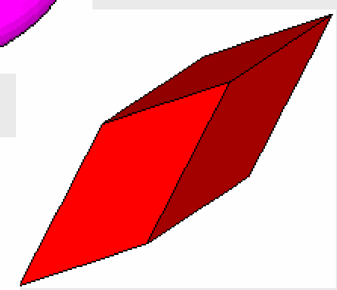
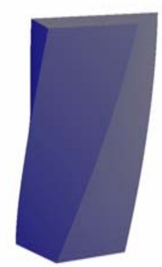
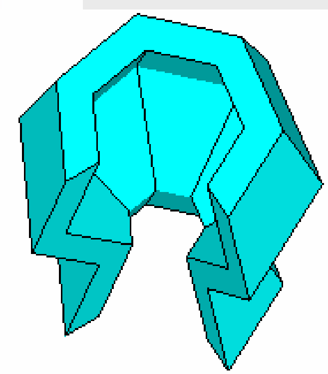
- Abstract class. All solids in Geant4 derive from it
  - Defines but does not implement all functions required to:
    - compute distances to/from the shape
    - check whether a point is inside the shape
    - compute the extent of the shape
    - compute the surface normal to the shape at a given point
- Once constructed, each solid is automatically registered in a specific solid store



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, ...
    - G4TwistedTubs, G4TwistedTrap, ...
  - BREP (Boundary REPresented) solids
    - G4BREPSolidPolycone, G4BSplineSurface, ...
    - Any order surface
  - Boolean solids
    - G4UnionSolid, G4SubtractionSolid, ...



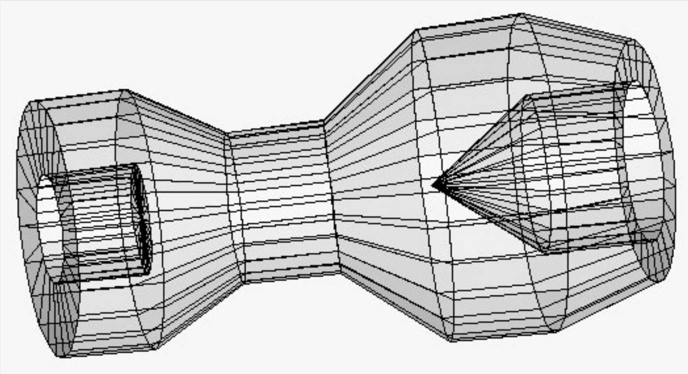
CSG: G4Tubs, G4Cons

```
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
```

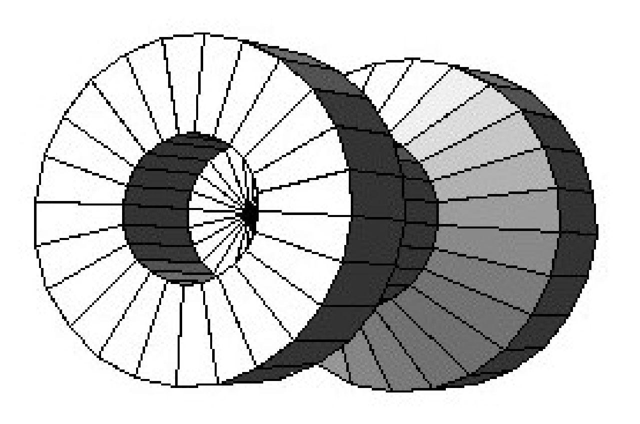
Specific CSG Solids: G4Polycone

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

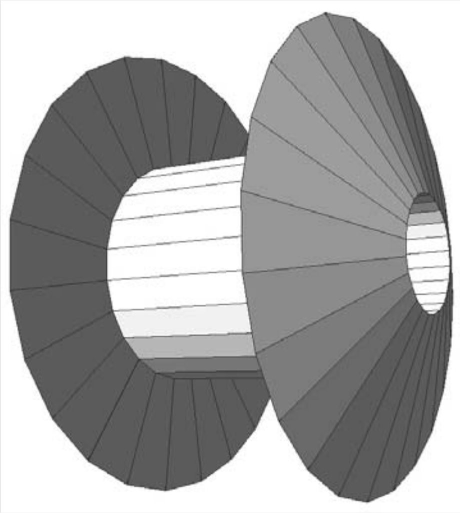


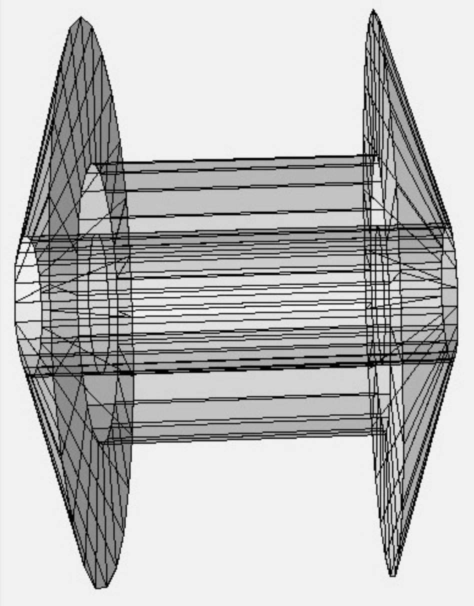
- `numRZ` - numbers of corners in the `r,z` space
- `r, z` - coordinates of corners

- Additional constructor using planes



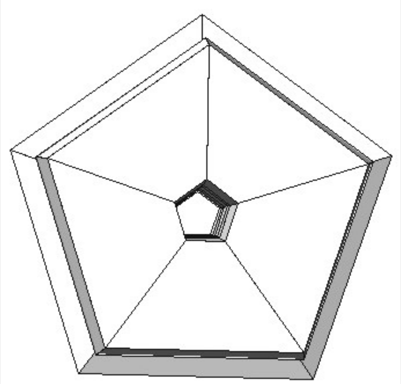
BREP Solids

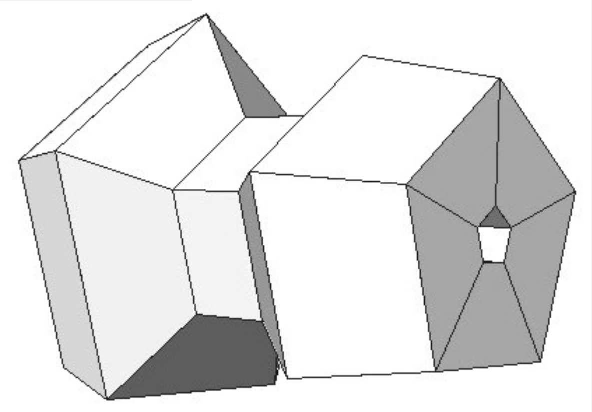




- *BREP = Boundary REPresented Solid*
- Listing all its surfaces specifies a solid
  - e.g. 6 squares for a cube
- Surfaces can be
  - planar, 2nd or higher order
    - elementary BREPS
  - Splines, B-Splines, *NURBS (Non-Uniform B-Splines)*
    - advanced BREPS
- Few elementary BREPS pre-defined
  -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

BREPS:
G4BREPSolidPolyhedra

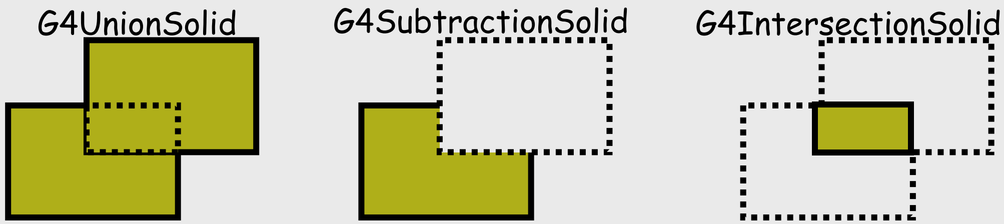




```
G4BREPSolidPolyhedra(const G4String& pName,
                           G4double phiStart,
                           G4double phiTotal,
                           G4int sides,
                           G4int nZplanes,
                           G4double zStart,
                     const G4double zval[],
                     const G4double rmin[],
                     const G4double rmax[]);
```

- `sides` - numbers of sides of each polygon in the `x-y` plane
- `nZplanes` - numbers of planes perpendicular to the `z` axis
- `zval[]` - `z` coordinates of each plane
- `rmin[], rmax[]` - Radii of inner and outer polygon at each plane

Boolean Solids



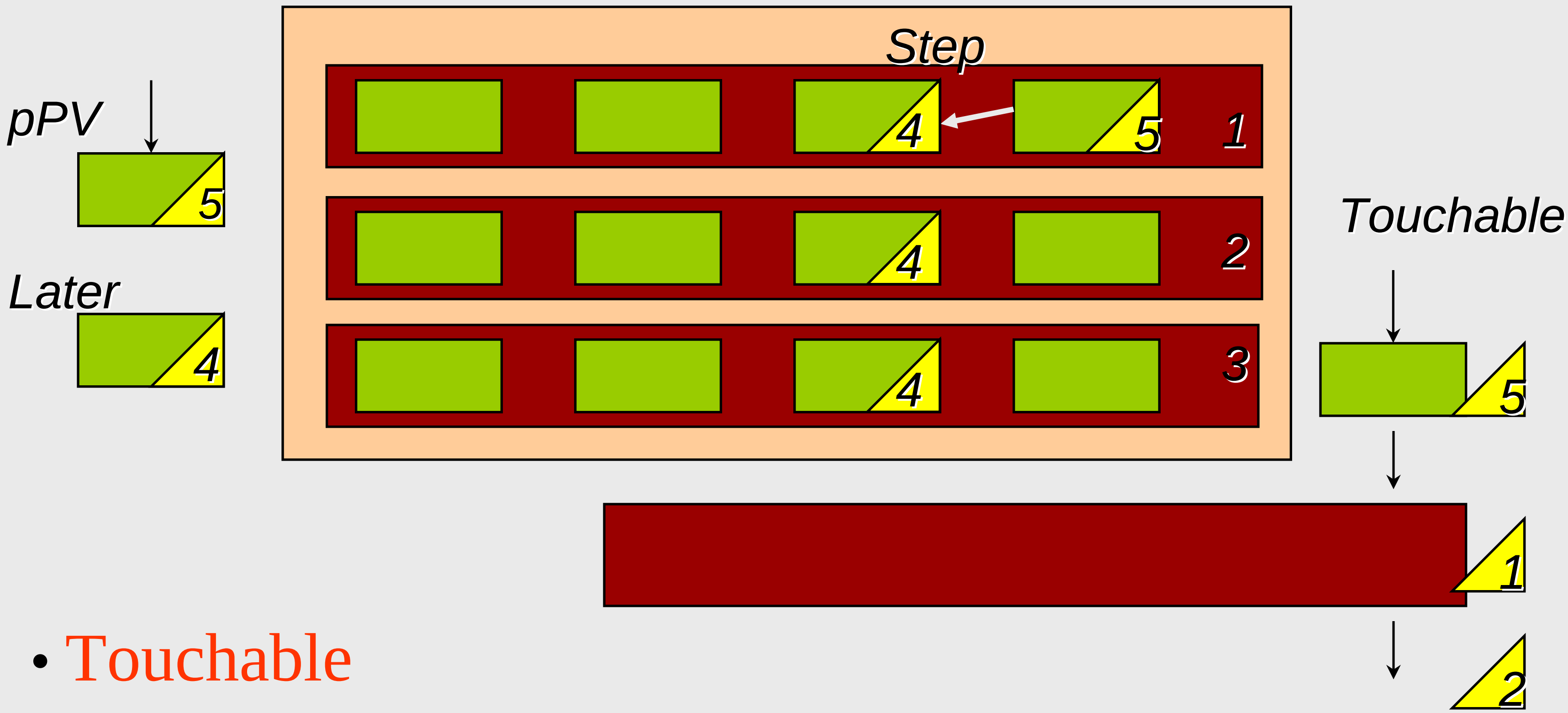
- Solids can be combined using boolean operations:
  - `G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid`
  - Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2nd solid
    - 2nd solid is positioned relative to the coordinate system of the 1st solid
- Example:

```
G4Box box(“Box", 20, 30, 40);
G4Tubs cylinder(“Cylinder”, 0, 50, 50, 0, 2*M_PI);  // r:     0 -> 50
                                                     // z:   -50 -> 50
                                                     // phi:   0 ->  2 pi
G4UnionSolid union("Box+Cylinder", &box, &cylinder);
G4IntersectionSolid intersect("Box Intersect Cylinder", &box, &cylinder);
G4SubtractionSolid subtract("Box-Cylinder", &box, &cylinder);
```

- 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

How to identify a volume uniquely?

- Need to identify a volume uniquely
- Is a physical volume pointer enough? NO!



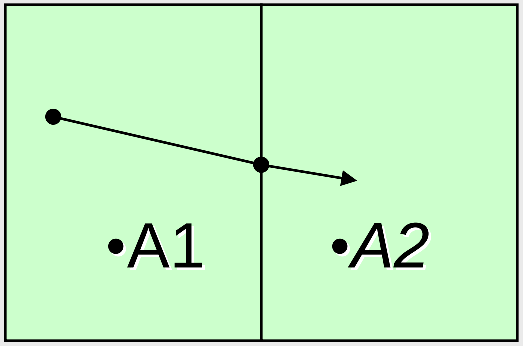
- Touchable

What can a touchable do ?

- All generic touchables can reply to these queries:
  - positioning information (rotation, position)
    - `GetTranslation()`, `GetRotation()`
- Specific types of touchable also know:
  - (solids) - their associated shape: `GetSolid()`
  - (volumes) - their physical volume: `GetVolume()`
  - (volumes) - their replication number: `GetReplicaNumber()`
  - (volumes hierarchy or touchable history):
    - info about its hierarchy of placements: `GetHistoryDepth()`
      - At the top of the history tree is the world volume
    - modify/update touchable: `MoveUpHistory()`, `UpdateYourself()`
      - take additional arguments

Benefits of Touchables in track

- Permanent information stored
  - to avoid implications with a “live” volume tree
- Full geometrical information available
  - to processes
  - to sensitive detectors
  - to hits



Touchable - 1

- G4Step has two G4StepPoint objects as its starting and ending points. All the geometrical information of the particular step should be got from “PreStepPoint”
  - Geometrical information associated with G4Track is basically same as “PostStepPoint”
- Each G4StepPoint object has:
  - position in world coordinate system
  - global and local time
  - material
  - G4TouchableHistory for geometrical information
    - Copy-number, transformations
- *Handles* (or *smart-pointers*) to touchables are intrinsically used. Touchables are reference counted

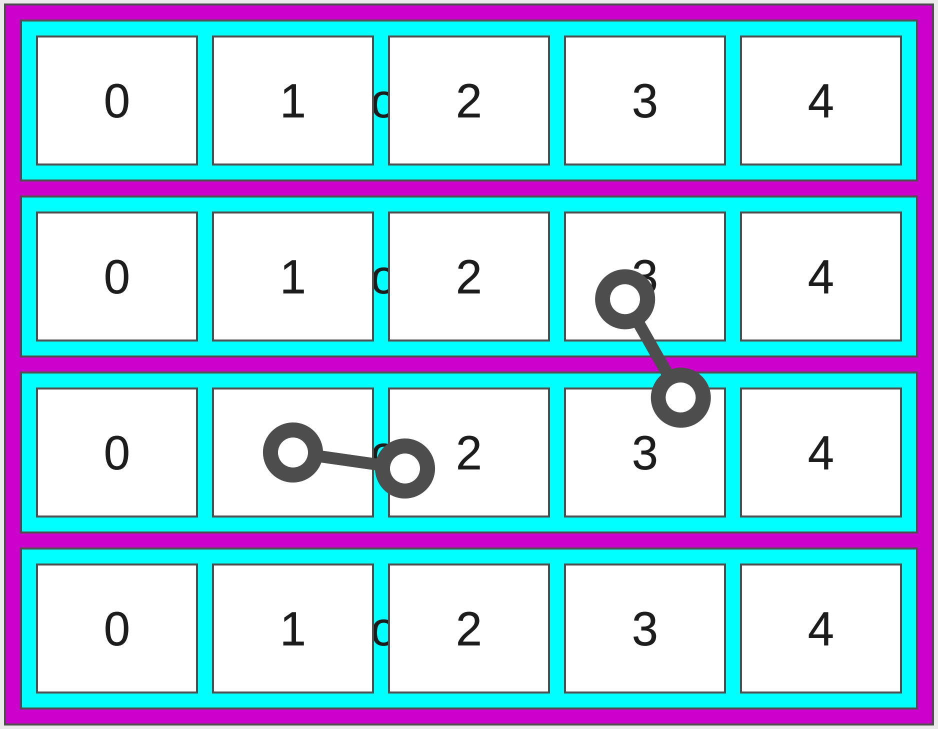
Touchable - 2

- G4TouchableHistory has information of geometrical hierarchy of the point

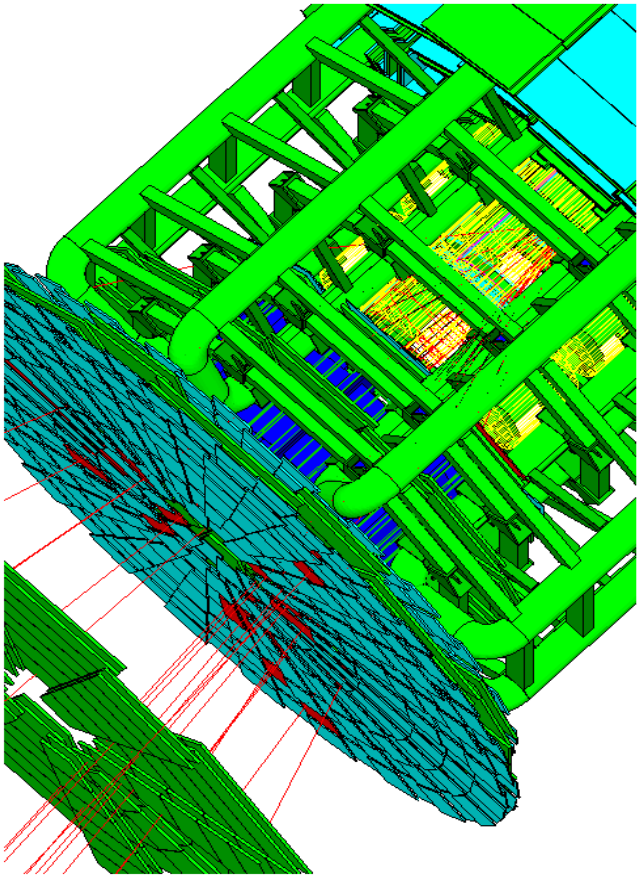
```
G4Step* aStep = ..;
G4StepPoint* preStepPoint = aStep->GetPreStepPoint();
G4TouchableHandle theTouchable =
               preStepPoint->GetTouchableHandle();
G4int copyNo = theTouchable->GetReplicaNumber();
G4int motherCopyNo = theTouchable->GetReplicaNumber(1);
G4ThreeVector worldPos = preStepPoint->GetPosition();
G4ThreeVector localPos = theTouchable->GetHistory()->
    GetTopTransform().TransformPoint(worldPos);
```

Copy numbers

- Suppose a calorimeter is made of 4x5 cells
  - and it is implemented by two levels of replica.
- In reality, there is only one physical volume object for each level. Its position is parameterized by its copy number
- To get the copy number of each level, suppose what happens if a step belongs to two cells



  - Remember geometrical information in G4Track is identical to "PostStepPoint". You cannot get the collect copy number for "PreStepPoint" if you directly access to the physical volume
- Use touchable to get the proper copy number, transform matrix,…



PART IV

Detector Description:

Visualization attributes & Optimization technique

Detector Description

Visualization attributes & Optimization technique

- *Visualization attributes*
- *Optimization technique & tuning*

Visualization of Detector

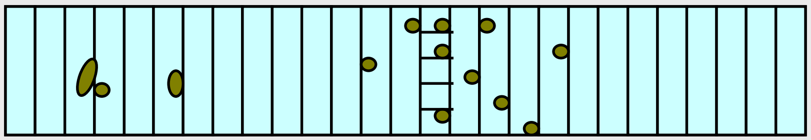
- Each logical volume can have associated a `G4VisAttributes` object
  - Visibility, visibility of daughter volumes
  - Color, line style, line width
  - Force flag to wire-frame or solid-style mode
- For parameterised volumes, attributes can be dynamically assigned to the logical volume
- Lifetime of visualization attributes must be at least as long as the objects they’re assigned to

Visualization of Hits and Trajectories

- Each `G4VHit` concrete class must have an implementation of *Draw()* method.
  - Colored marker
  - Colored solid
  - Change the color of detector element
- `G4Trajectory` class has a *Draw()* method.
  - Blue : positive
  - Green : neutral
  - Red : negative
  - You can implement alternatives by yourself

Smart voxels

- For each mother volume
  - a one-dimensional virtual division is performed
    - the virtual division is along a chosen axis
    - the axis is chosen by using an heuristic
  - Subdivisions (slices) containing same volumes are gathered into one
  - Subdivisions containing many volumes are refined
    - applying a virtual division again using a second Cartesian axis
    - the third axis can be used for a further refinement, in case
- *Smart voxels* are computed at initialisation time
  - When the detector geometry is *closed*
  - Do not require large memory or computing resources
  - At tracking time, searching is done in a hierarchy of virtual divisions



Detector description tuning

- Some geometry topologies may require ‘special’ tuning for ideal and efficient optimisation
  - for example: a dense nucleus of volumes included in very large mother volume
- Granularity of voxelisation can be explicitly set
  - Methods `Set/GetSmartless()` from `G4LogicalVolume`
- Critical regions for optimisation can be detected
  - Helper class `G4SmartVoxelStat` for monitoring time spent in detector geometry optimisation
    - Automatically activated if `/run/verbose` greater than `1`

```
Percent      Memory     Heads    Nodes   Pointers    Total CPU    Volume
-------      ------     -----    -----   --------    ---------    -----------
  91.70          1k         1       50         50         0.00    Calorimeter
   8.30          0k         1        3          4         0.00    Layer
```

Visualising voxel structure

- The computed voxel structure can be visualized with the final detector geometry
  - Helper class **`G4DrawVoxels`**
  - Visualize voxels given a logical volume
    - `G4DrawVoxels::DrawVoxels(const G4LogicalVolume*)`
  - Allows setting of visualization attributes for voxels
    - `G4DrawVoxels::SetVoxelsVisAttributes(…)`
  - useful for debugging purposes
  - Can also be done through a visualization command at run-time:
    - `/vis/scene/add/logicalVolume <logical-volume-name> [<depth>]`

Customising optimisation

- Detector regions may be excluded from optimisation (ex. for debug purposes)
  - Optional argument in constructor of `G4LogicalVolume` or through provided set methods
    - `SetOptimisation`/`IsToOptimise()`
  - Optimisation is turned on by default
- Optimisation for parameterised volumes can be chosen
  - Along one single Cartesian axis
    - Specifying the axis in the constructor for `G4PVParameterised`
  - Using 3D voxelisation along the 3 Cartesian axes
    - Specifying in `kUndefined` in the constructor for `G4PVParameterised`

PART V

Detector Description:

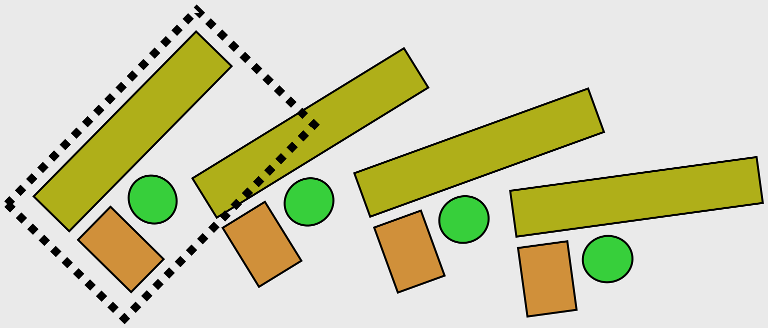
Advanced features

Detector Description

Advanced features

- *Grouping volumes*
- *Reflections of volumes and hierarchies*
- *Detector regions*
- *User defined solids*
- *Debugging tools*

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
- ***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

G4AssemblyVolume

```
G4AssemblyVolume( G4LogicalVolume* volume,
                  G4ThreeVector& translation,
                  G4RotationMatrix* rotation);
```

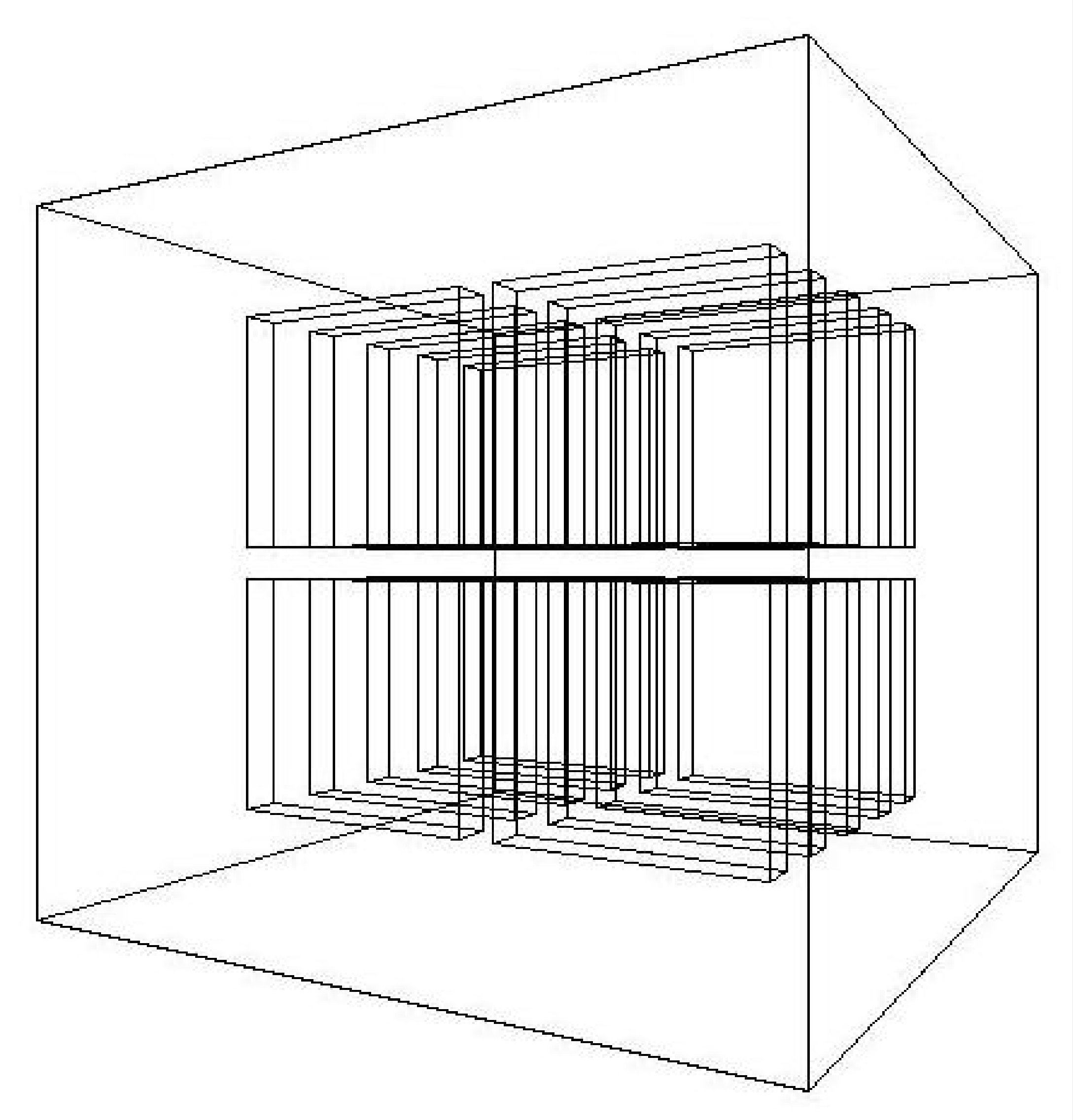
- Helper class to combine logical volumes in arbitrary way
  - Participating logical volumes are treated as triplets
    - logical volume, translation, rotation
  - Imprints of the assembly volume are made inside a mother logical volume through `G4AssemblyVolume::MakeImprint(…)`
  - Each physical volume 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)

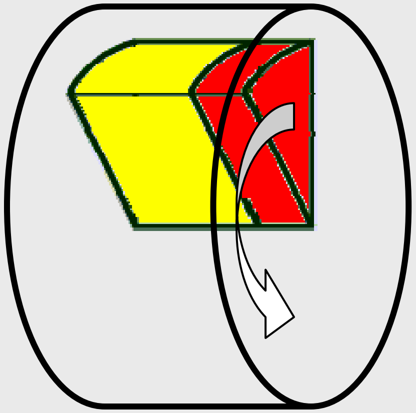
Assembly of volumes:
example -1

```
// Define a plate
   G4VSolid* PlateBox = new G4Box( "PlateBox", plateX/2., plateY/2., plateZ/2. );
   G4LogicalVolume* plateLV = new G4LogicalVolume( PlateBox, Pb, "PlateLV", 0, 0, 0 );
 // Define one layer as one assembly volume
   G4AssemblyVolume* assemblyDetector = new G4AssemblyVolume();
 // Rotation and translation of a plate inside the assembly
   G4RotationMatrix Ra; G4ThreeVector Ta;
 // Rotation of the assembly inside the world
   G4RotationMatrix Rm;
 // Fill the assembly by the plates
   Ta.setX( caloX/4. ); Ta.setY( caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
   Ta.setX( -1*caloX/4. ); Ta.setY( caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
   Ta.setX( -1*caloX/4. ); Ta.setY( -1*caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
   Ta.setX( caloX/4. ); Ta.setY( -1*caloY/4. ); Ta.setZ( 0. );
   assemblyDetector->AddPlacedVolume( plateLV, G4Transform3D(Ra,Ta) );
 // Now instantiate the layers
   for( unsigned int i = 0; i < layers; i++ ) {
     // Translation of the assembly inside the world
     G4ThreeVector Tm( 0,0,i*(caloZ + caloCaloOffset) - firstCaloPos );
     assemblyDetector->MakeImprint( worldLV, G4Transform3D(Rm,Tm) );
   }
```

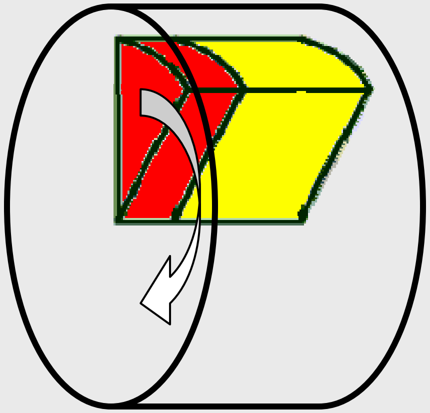
Assembly of volumes:

example -2





Reflecting volumes



- `G4ReflectedSolid`
  - utility class representing a solid shifted from its original reference frame to a new *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
  - Provides tools to detect/return a reflected volume
- Reflections can be applied to CSG and specific solids

Reflecting hierarchies of volumes - 1

`G4ReflectionFactory::Place(…)`

- Used for normal placements:
  1. Performs the transformation decomposition
  2. Generates a new reflected solid and logical volume
     - Retrieves it from a map if the reflected object is already created
  3. Transforms any daughter and places them in the given mother
  4. Returns a pair of physical volumes, the second being a placement in the reflected mother

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

Reflecting hierarchies of volumes - 2

G4ReflectionFactory::Replicate(…)

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

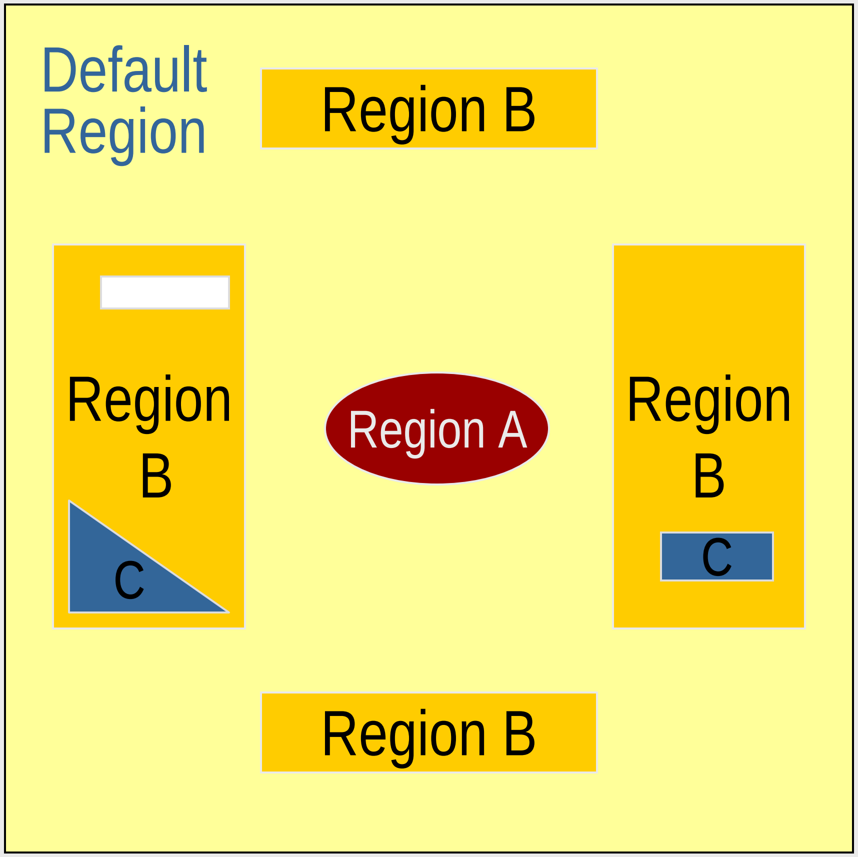
```
G4PhysicalVolumesPair
Replicate(const G4String&  name,      // the actual name
          G4LogicalVolume* LV,        // the logical volume
          G4LogicalVolume* motherLV,  // the mother volume
          Eaxis            axis       // axis of replication
          G4int            replicaNo  // number of replicas
          G4int            width,     // width of single replica
          G4int            offset=0)  // optional mother offset
```

Cuts by Region

- Geant4 has had a unique production threshold (‘cut’) expressed in length (i.e. minimum range of secondary)
  - For all volumes
  - Possibly different for each particle.
- Yet appropriate length scales can vary greatly between different areas of a large detector
  - E.g. a vertex detector (5 μm) and a muon detector (2.5 cm)
  - Having a unique (low) cut can create a performance penalty
- Geant4 allows for several cuts
  - Globally or per particle
  - Enabling the tuning of production thresholds at the level of a sub-detector, i.e. region
  - Cuts are applied only for gamma, electron and positron and only for processes which have infrared divergence

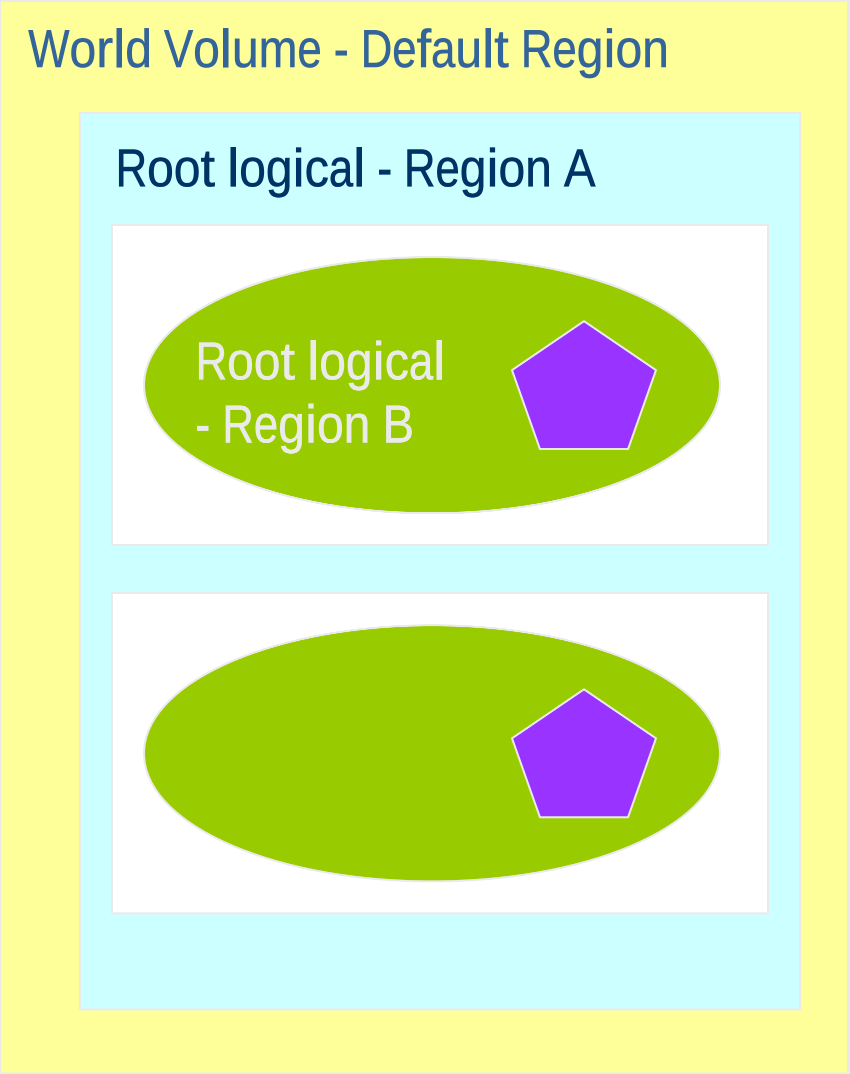
Detector Region

- Concept of region:
  - Set of geometry volumes, typically of a sub-system
    - barrel + end-caps of the calorimeter;
    - “Deep” areas of support structures can be a region.
  - Or any group of volumes
- A set of cuts in range is associated to a region
  - a different range cut for each particle among gamma, e-, e+ is allowed in a region



Region and cut

- Each region has its unique set of cuts.
- World volume is recognized as the default region. The default cuts defined in Physics list are used for it.
  - User is not allowed to define a region to the world volume or a cut to the default region
- A logical volume becomes a root logical volume once it is assigned to a region.
  - All daughter volumes belonging to the root logical volume share the same region (and cut), unless a daughter volume itself becomes to another root
- Important restriction :
  - No logical volume can be shared by more than one regions, regardless of root volume or not

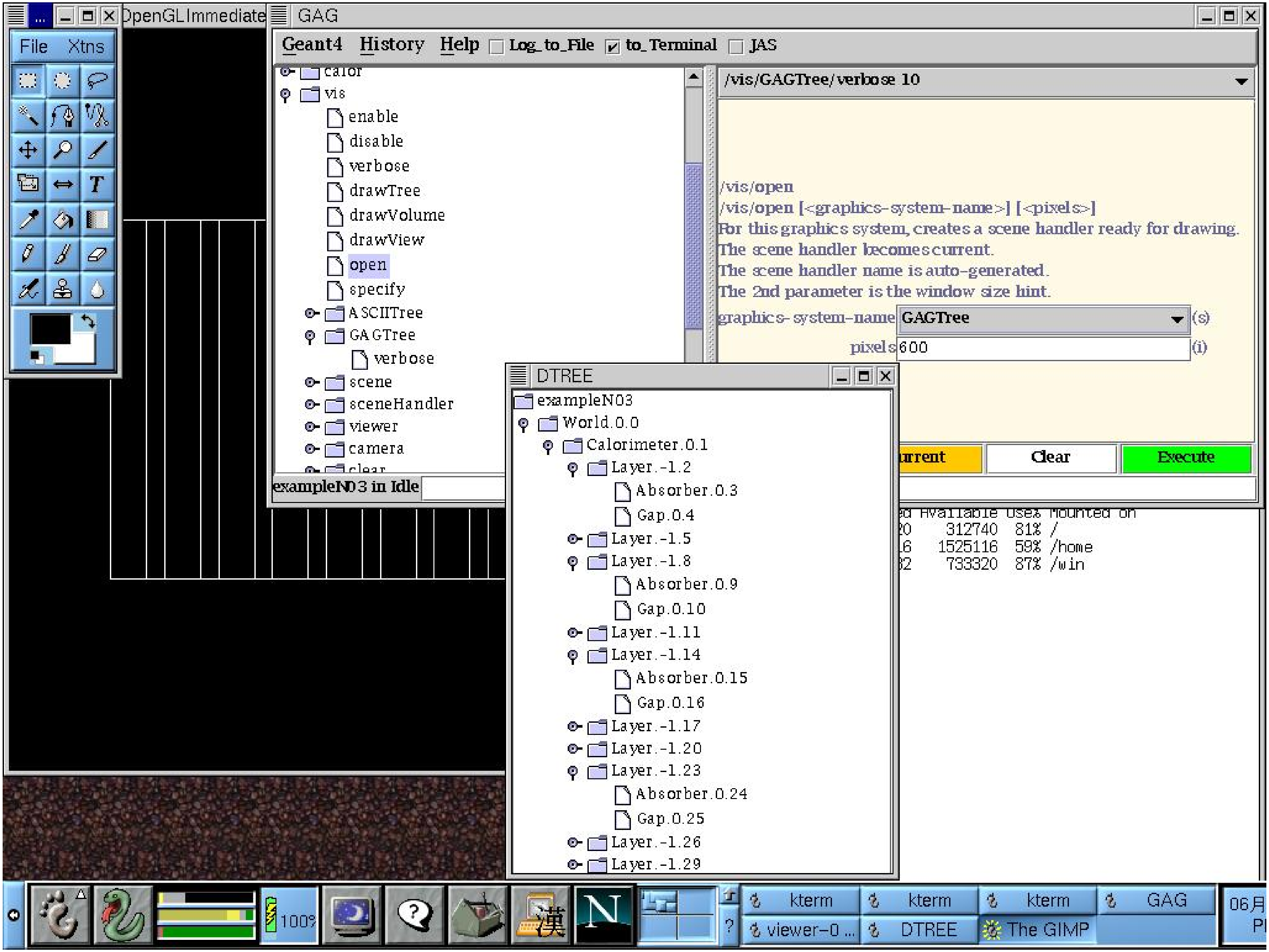


GGE (Graphical Geometry Editor)

- 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, e.g. DAWN
  - 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/

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



OpenGLImmediate
File Xtns
GAG
Geant4 History Help Log_to_File to_Terminal JAS
/vis/GAGTree/verbose 10
calor
vis
enable
disable
verbose
drawTree
drawVolume
drawView
open
specify
ASCIITree
GAGTree
verbose
scene
sceneHandler
viewer
camera
clear
exampleN03 in Idle
/vis/open
/vis/open [<graphics-system-name>] [<pixels>]
For this graphics system, creates a scene handler ready for drawing.
The scene handler becomes current.
The scene handler name is auto-generated.
The 2nd parameter is the window size hint.
graphics-system-name GAGTree (s)
pixels 600 (i)
Clear
Execute
DTREE
exampleN03
World.0.0
Calorimeter.0.1
Layer.-1.2
Absorber.0.3
Gap.0.4
Layer.-1.5
Layer.-1.8
Absorber.0.9
Gap.0.10
Layer.-1.11
Layer.-1.14
Absorber.0.15
Gap.0.16
Layer.-1.17
Layer.-1.20
Layer.-1.23
Absorber.0.24
Gap.0.25
Layer.-1.26
Layer.-1.29
312740 81% /
1525116 59% /home
733320 87% /win
kterm
kterm
kterm
GAG
viewer-0 ...
DTREE
The GIMP
100%

Computing volumes and masses

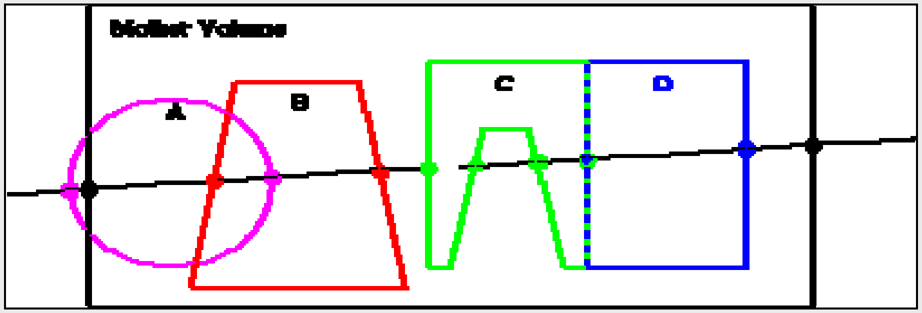
- Geometrical volume of a generic solid or boolean composition can be computed from the **solid**:

```
G4double GetCubicVolume();
```

- Overall mass of a geometry setup (subdetector) can be computed from the **logical volume**:

```
G4double GetMass(G4Bool forced=false,
                 G4Material* parameterisedMaterial=0);
```

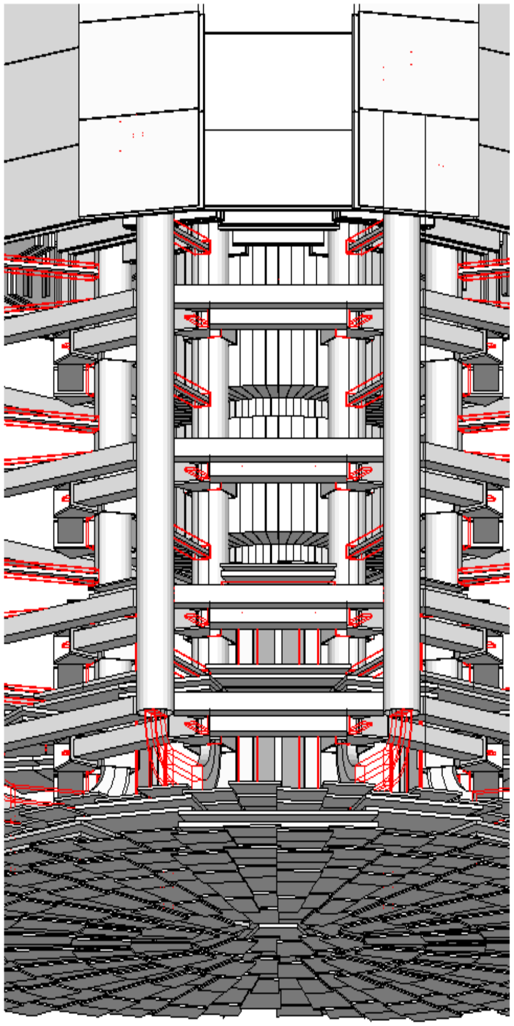
Debugging geometries



- 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
  - 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://geant4.kek.jp/GEANT4/vis/DAWN/About_DAVID.html



Debugging run-time commands

- Built-in run-time commands to activate verification tests for the user geometry. Tests can be applied recursively to all depth levels (may require CPU time!): `[recursion_flag]`

`geometry/test/run [recursion_flag]` or

`geometry/test/grid_test [recursion_flag]`

  - to start verification of geometry for overlapping regions based on a standard grid setup

`geometry/test/cylinder_test [recursion_flag]`

  - shoots lines according to a cylindrical pattern

`geometry/test/line_test [recursion_flag]`

  - to shoot a line along a specified direction and position

`geometry/test/position` and `geometry/test/direction`

  - to specify position & direction for the `line_test`

- Resolution/dimensions of grid/cylinders can be tuned

Debugging run-time commands - 2

- Example layout:

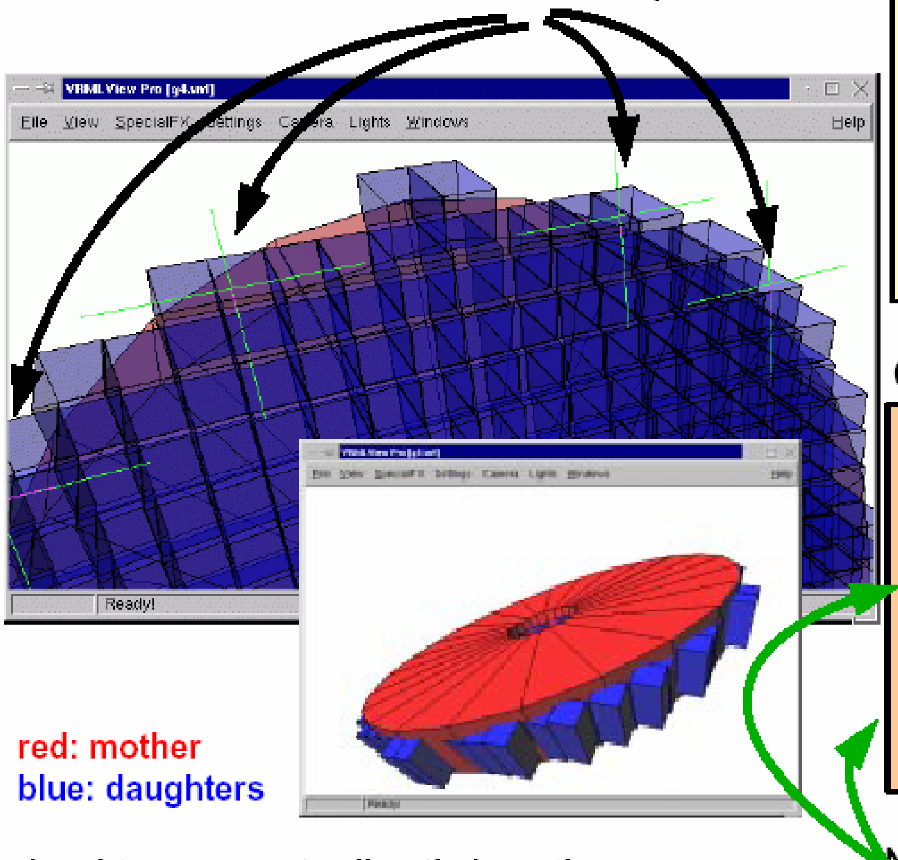
```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list truncated)
  length (cm)    ----- 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) -----
    . . .
```

Debugging tools: OLAP

- Uses tracking of neutral particles to verify boundary crossing in opposite directions
- Stand-alone batch application
  - Provided as extended example
  - Can be combined with a graphical environment and GUI (ex. Qt library)
  - Integrated in the CMS Iguana Framework

Debugging tools: OLAP

graphical indication of
detected overlaps



red: mother
blue: daughters

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)