Geant 4

Detector Description

Authors: *J.Apostolakis, G.Cosmo*

Detector Description

Part I *The Basics*

Part II *Logical and physical volumes*

Part III *Solids & touchables*

Part IV *Electromagnetic field*

Part V *Visualization attributes & Optimization technique*

Part VI *Advanced features*

PART 1

Detector Description: the Basics

Concepts for Detector Description

The following concepts will be described:

- Unit system
- Material
- Detector Geometry
- Sensitive Volumes
- Hits

Unit system

- Geant4 has no default unit. To create a physical quantity, a number must be “multiplied” by the unit.
  - 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 this is discouraged, as the default could change!
  - Almost all commonly used units are available.
  - The user can define new units.
  - Refer to CLHEP: `SystemOfUnits.h`
- To print, divide a variable by the unit you want:

```
G4cout << dE / MeV << “ (MeV)” << G4endl;
```

HEP 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`, .. :

```
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,    // number of atoms
                 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);
```

- Note that 'total' vacuum is not allowed
  - Must use low-density gas

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);
```

PART Ib

Creating a Detector or Setup

Describe your detector

- Derive your own concrete class from `G4VUserDetectorConstruction` abstract base class.
- Key: Implementing the method `Construct()`:
  - You can structure it to do all that is necessary:
    - Construct all necessary materials
    - Define the shapes / solids that you need
    - Construct and place the volumes of your detector / setup
    - Define sensitive detectors and identify detector volumes which to associate them
    - Associate a field to detector regions
    - Define visualization attributes for the detector elements
  - Or you can structure it in pieces, one for each detector component or sub-detector:

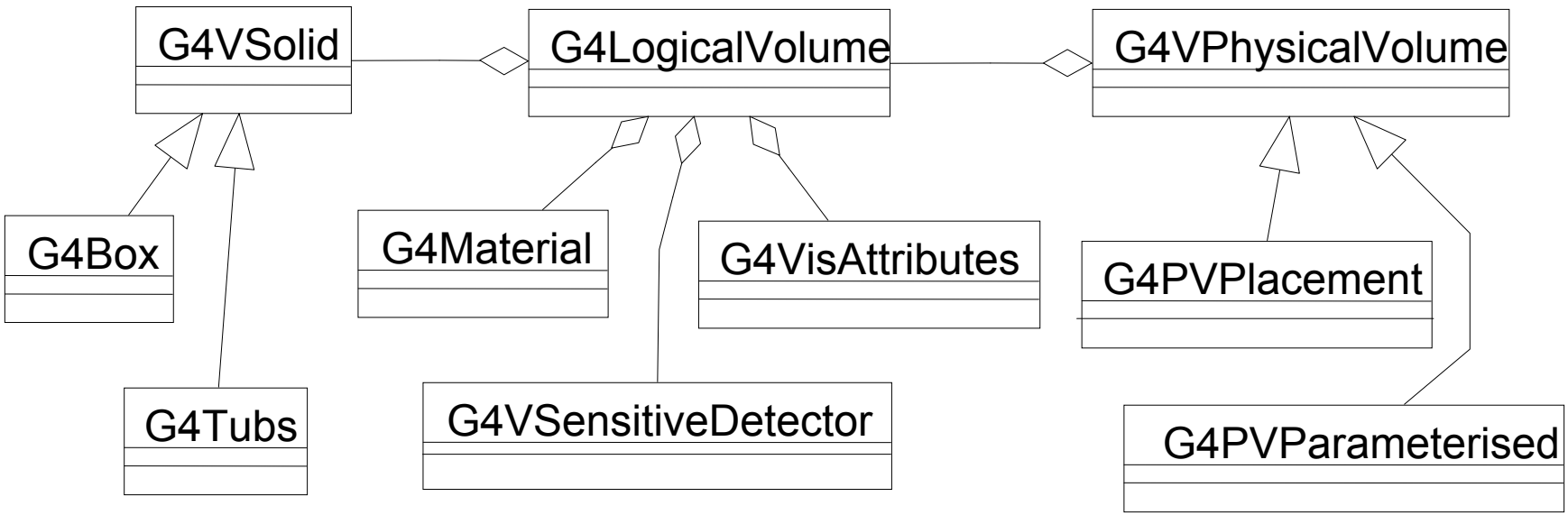
Creating a Detector Volume

- 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");
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

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);
```

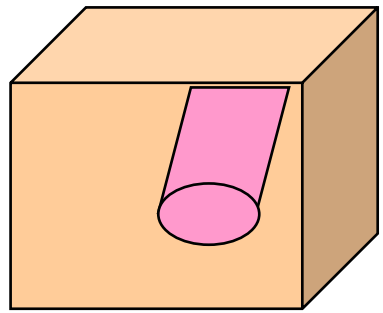
- Contains all information of volume except position:
  - Shape and dimension (G4VSolid)
  - Material, sensitivity, visualization attributes
  - Magnetic field, User limits
  - Identity of daughter volumes
  - Shower parameterisation
- Physical volumes of the same type can share a logical volume.
- The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

G4VPhysicalVolume

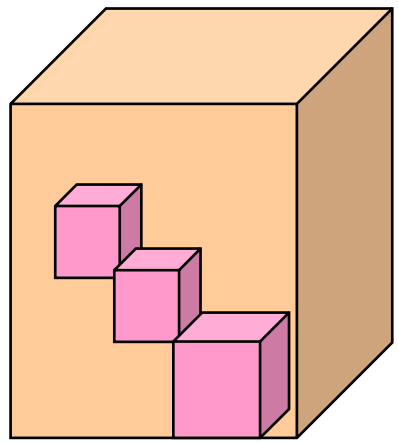
- **`G4PVPlacement`** 1 Placement = One Volume
  - A volume instance positioned once in a mother volume
- **`G4PVReplica`** 1 Replica = Many Volumes
  - Slicing a volume into smaller pieces (if it has a symmetry)
- **`G4PVParameterised`** 1 Parameterised = Many Volumes
  - Parameterised by the copy number
    - 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 or b) are identical in size & shape.

Physical Volumes

- Placement: it is one positioned volume
  - Is it the most common and well suited for most usage.
- Repeated: it is a volume placed many times
  - A replica or parameterised can represent any number of volumes
  - It reduces the amount of memory used.
  - So it is best suited where a large number of similar volumes must be created.
- NOTE: A **mother** volume can contain **either**
  - **many placement** volumes **OR**
  - **one repeated** volume



placement



repeated

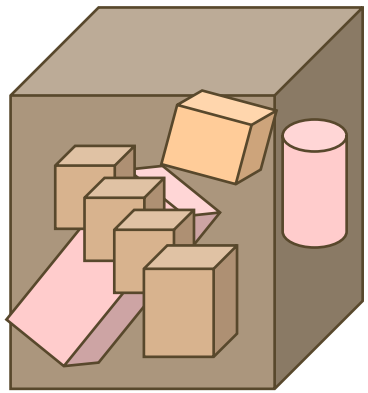
G4PVPlacement

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

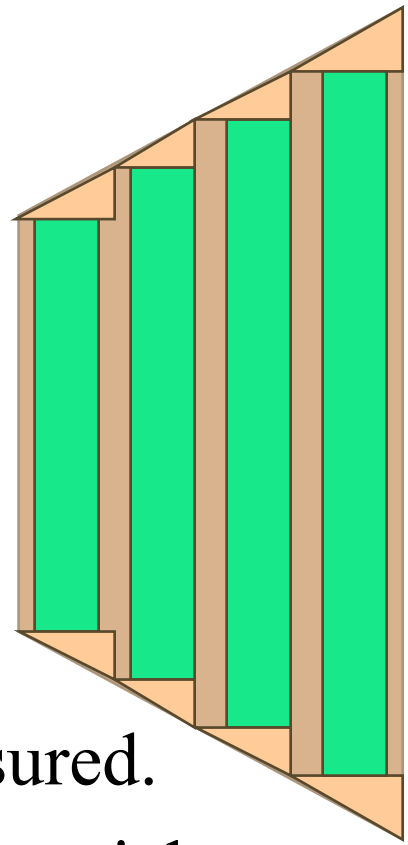
- A single volume positioned relatively to the mother volume
  - In a frame translated and rotated relative to the coordinate system of the mother volume
- Three additional constructors:
  - Using `G4Transform3D` to represent the direct rotation and translation of the solid instead of the frame
  - A simple variation: specifying the mother volume as a pointer to its physical volume instead of its logical volume (likely to become obsolescent).
  - The combination of the two variants above (ditto)

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 Parameterized Volumes



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

    G4 geometry: 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 `pMotherLogical`
- 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, logTracker,
                          kZAxis, NbOfChambers, chamberParam);
```

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

Parameterisation
example - 2

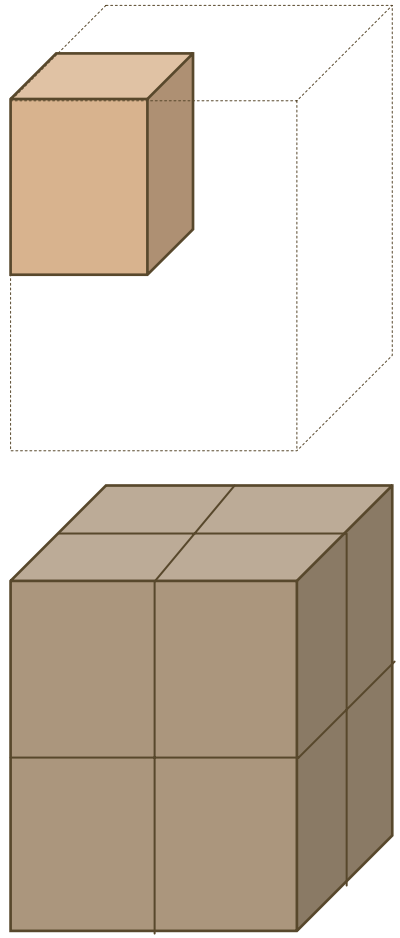
```
class ChamberParameterisation : public G4VPVParameterisation
{
  public:
    ChamberParameterisation( G4int NoChambers, G4double startZ,
                             G4double spacing, G4double widthChamber,
                             G4double lenInitial, G4double lenFinal );
   ~ChamberParameterisation();
   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 (except R).
- 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



repeated

G4PVReplica

```
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 phi-replication, where the mother solid has an offset along this axis
- 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

Replication
example

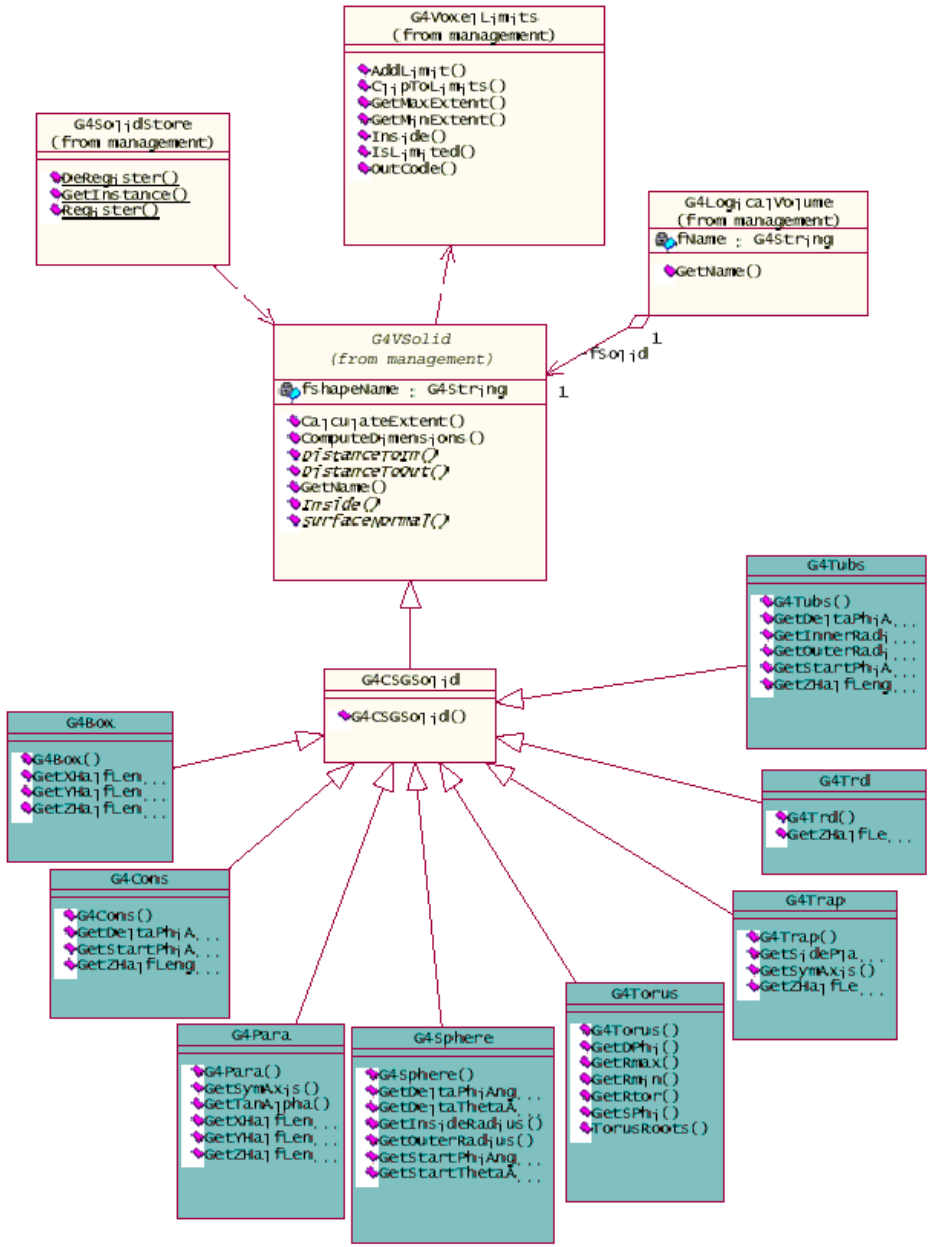
```
G4double tube_dPhi = 2.* M_PI;
 G4VSolid* tube =
   new G4Tubs("tube", 20*cm, 50*cm, 30*cm, 0., tube_dPhi*rad);
 G4LogicalVolume * tube_log =
   new G4LogicalVolume(tube, Ar, "tubeL", 0, 0, 0);
 G4VPhysicalVolume* tube_phys =
   new G4PVPlacement(0,G4ThreeVector(-200.*cm, 0., 0.*cm),
                     "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);
```

PART III

Detector Description:
Solids & Touchables

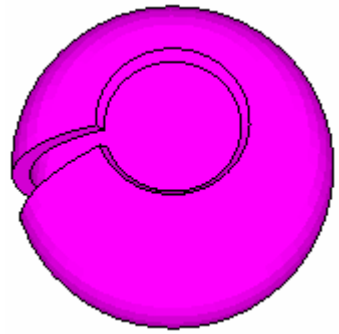
G4VSolid

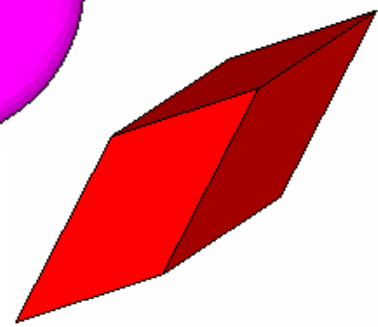
- Abstract class. All solids in Geant4 derive from it
  - Defines but does not implement all functions required to:
    - compute distances to/from the shape
    - check whether a point is inside the shape
    - compute the extent of the shape
    - compute the surface normal to the shape at a given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

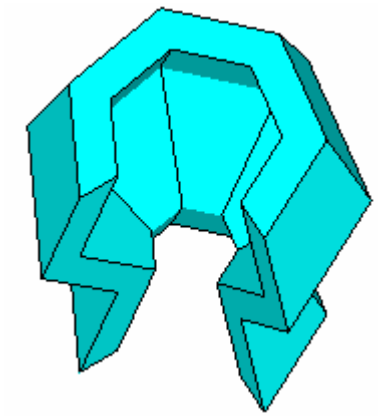


Solids

- 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, …
  - BREP (Boundary REPresented) solids
    - G4BREPSolidPolycone, G4BSplineSurface, …
    - Any order surface
  - Boolean solids
    - G4UnionSolid, G4SubtractionSolid, …
  - STEP interface
    - to import BREP solid models from CAD systems - STEP compliant solid modeler





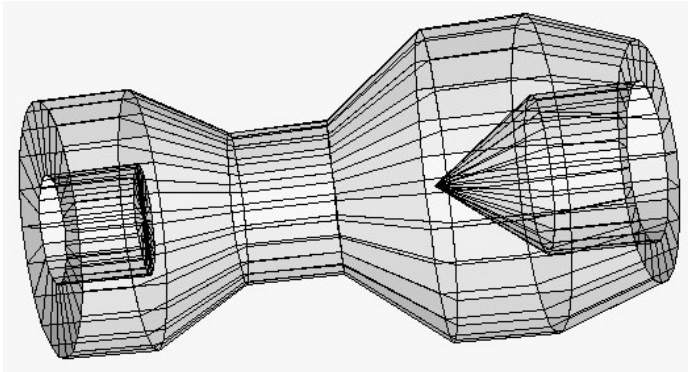


CSG: G4Tubs, G4Cons

```
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
```

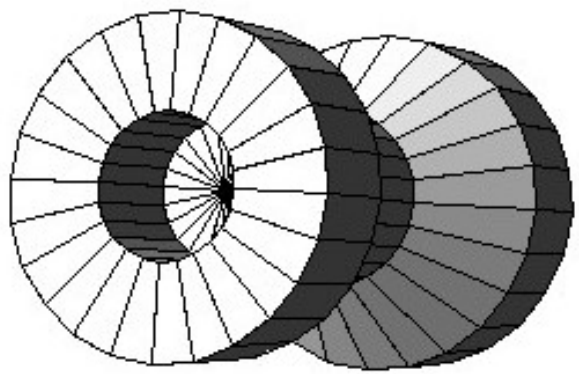
Specific CSG Solids: G4Polycone

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

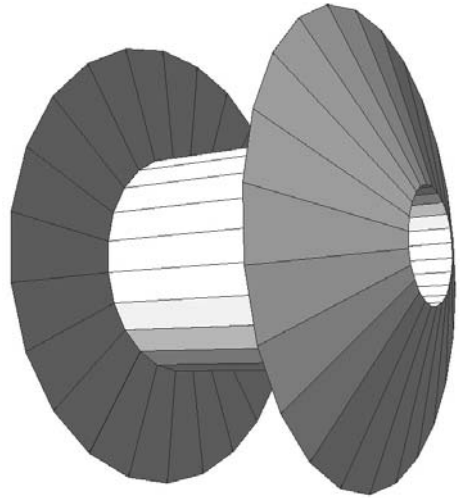


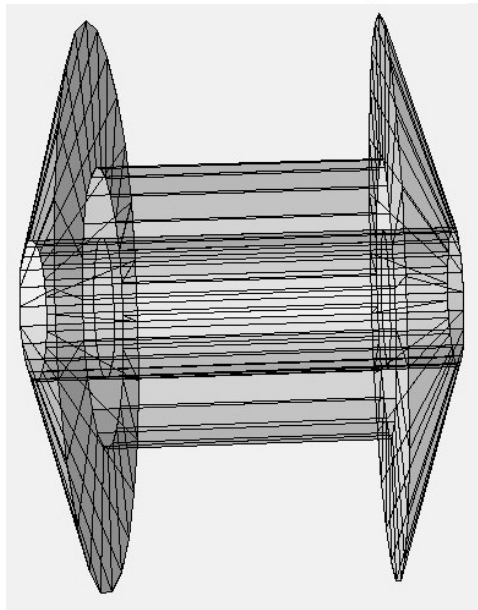
- `numRZ` - numbers of corners in the `r,z` space
- `r, z` - coordinates of corners

- Additional constructor using planes



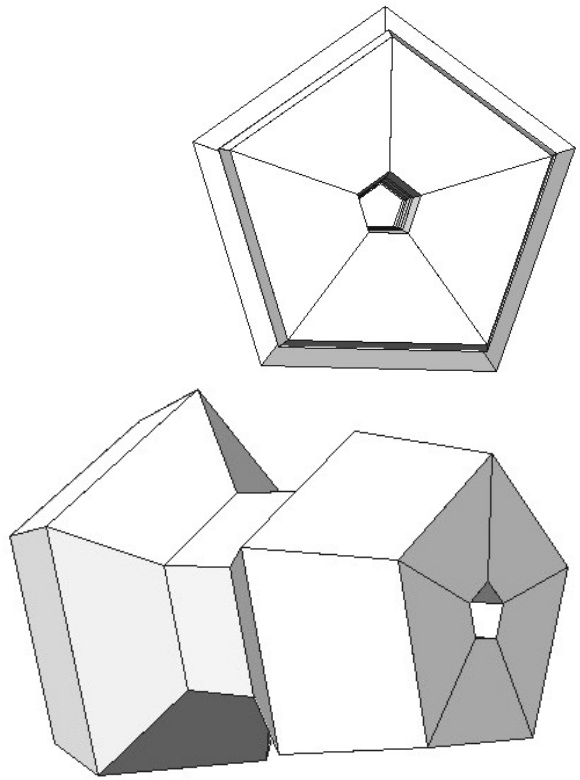
BREP Solids





- *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

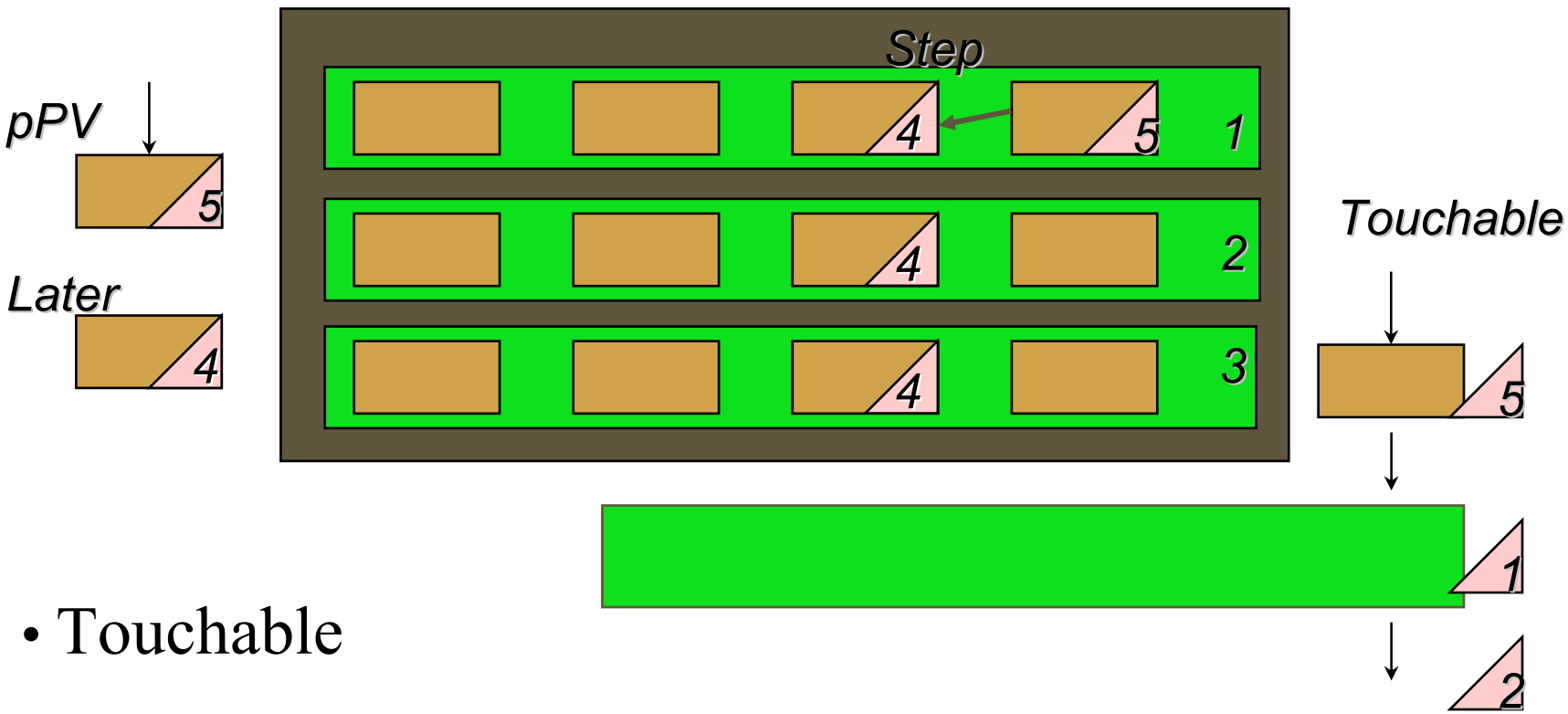
- Solids can be combined using boolean operations:
  - `G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid`
  - Requires: 2 solids, 1 boolean operation, and an (optional) transformation for the 2nd solid
    - 2nd solid is positioned relative to the coordinate system of the 1st solid
- Example:

```
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!



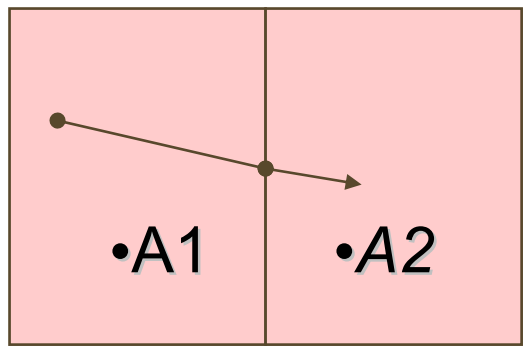
- Touchable

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
  - unlike “live” volume tree
    - which the Navigator creates
- Full geometrical information available
  - to processes
  - to sensitive detectors
  - to user actions



Touchable - 1

- G4Step has two G4StepPoint objects as its starting and ending points. All the volume information of the particular step should be taken 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
- Since release 4.0, *handles* (or *smart-pointers*) to touchables are used.
  - So touchables are reference counted
  - A touchable lives as long as another object keeps a handle to it.

Touchable - 2

- G4TouchableHistory has the full information of the geometrical hierarchy of the current volume

```
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);
```

PART IV

Electromagnetic Fields

Detector sensitivity

- A logical volume becomes sensitive if it has a pointer to a concrete class derived from `G4VSensitiveDetector`.
- A sensitive detector either
  - constructs one or more hit objects or
  - accumulates values to existing hits

  using information given in a `G4Step` object.

NOTE: you must get the volume information from the “PreStepPoint”.

Sensitive detector and Hit

- Each “Logical Volume” can have a pointer to a sensitive detector.
- Hit is a snapshot of the physical interaction of a track or an accumulation of interactions of tracks in the sensitive region of your detector.
- A sensitive detector creates hit(s) using the information given in `G4Step` object. The user has to provide his/her own implementation of the detector response.
- Hit objects, which still are the user’s class objects, are collected in a `G4Event` object at the end of an event.
  - The UserSteppingAction class should NOT do this.

Hit class

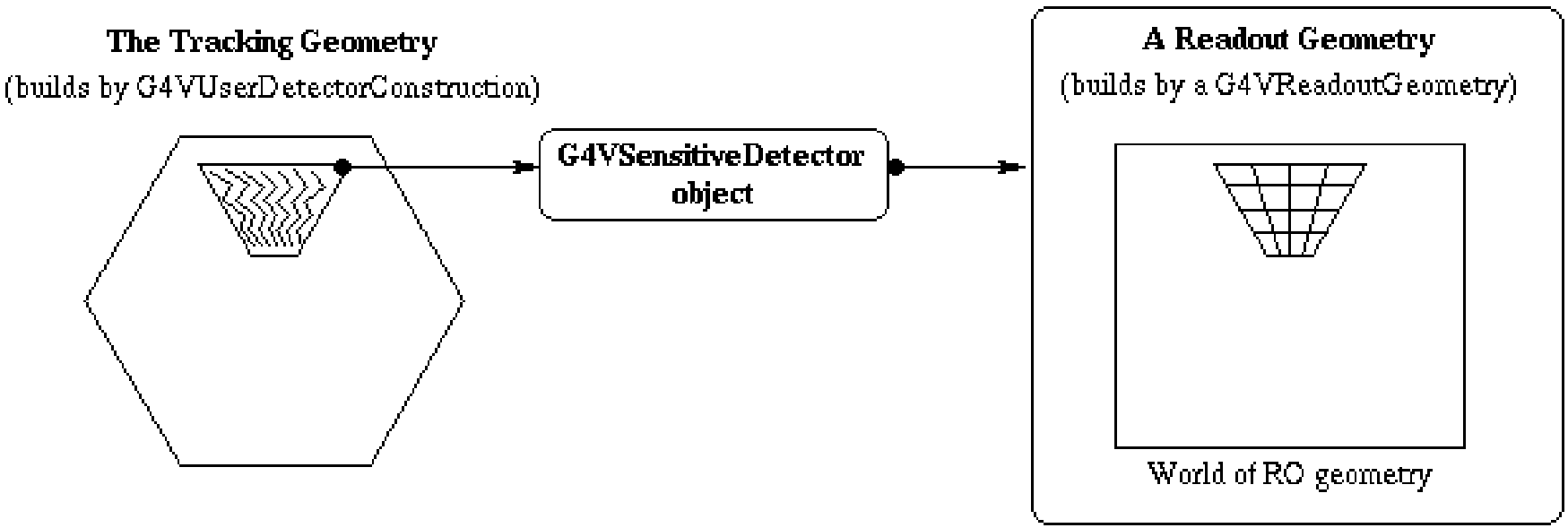
- Hit is a user-defined class derived from `G4VHit`.
- You can store various types information by implementing your own concrete Hit class.
- For example:
  - Position and time of the step
  - Momentum and energy of the track
  - Energy deposition of the step
  - Geometrical information
  - or any combination of above

Hit class

- Hit objects of a concrete hit class must be stored in a dedicated collection which is instantiated from `G4THitsCollection` template class.
- The collection will be associated to a `G4Event` object via `G4HCofThisEvent`.
- Hits collections are accessible
  - through `G4Event` at the end of event,
  - through `G4SDManager` during processing an event.
    --> Used for Event filtering.

Readout geometry

- Readout geometry is a virtual and artificial geometry which can be defined in parallel to the real detector geometry.
- A readout geometry is optional.
- Each one is associated to a sensitive detector.



Digitization

- Digit represents a detector output (e.g. ADC/TDC count, trigger signal).
- Digit is created with one or more hits and/or other digits by a concrete implementation derived from `G4VDigitizerModule`.
- In contradiction to the Hit which is generated at tracking time automatically, the digitize() method of each `G4VDigitizerModule` must be explicitly invoked by the user's code (e.g. EventAction).

Defining a sensitive detector

- Basic strategy

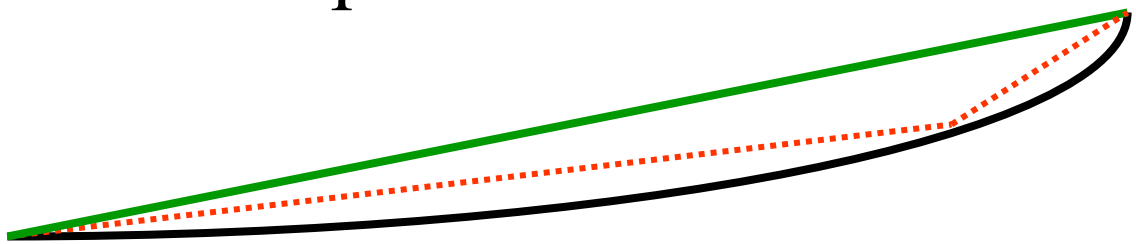
```
G4LogicalVolume* myLogCalor = ……;
G4VSensitiveDetector* pSensitivePart =
   new MyCalorimeterSD(“/mydet/calorimeter”);
G4SDManager* SDMan = G4SDManager::GetSDMpointer();
SDMan->AddNewDetector(pSensitivePart);
myLogCalor->SetSensitiveDetector(pSensitivePart);
```

Magnetic field

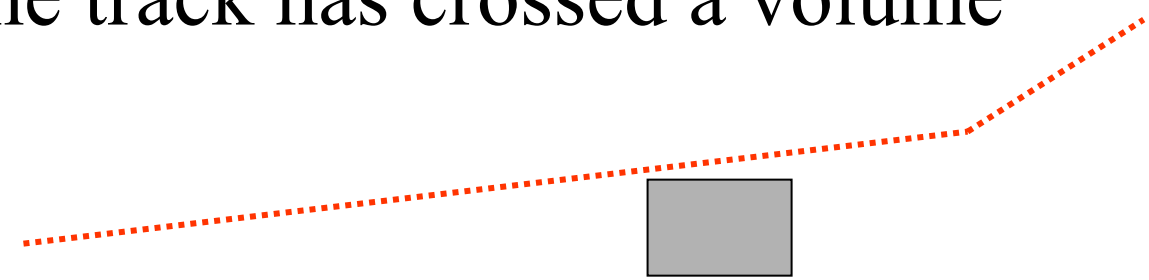
- In order to propagate a particle inside a field (e.g. magnetic, electric or both), we integrate the equation of motion of the particle in the field.
- In general this is best done using a Runge-Kutta method for the integration of ordinary differential equations. Several Runge-Kutta methods are available.
- In specific cases other solvers can also be used:
  - In a uniform field, as the analytical solution is known.
  - In a nearly uniform field, where we perturb it.

Magnetic field

- Once a method is chosen that allows G4 to calculate the track's motion in a field, we break up this curved path into linear chord segments.

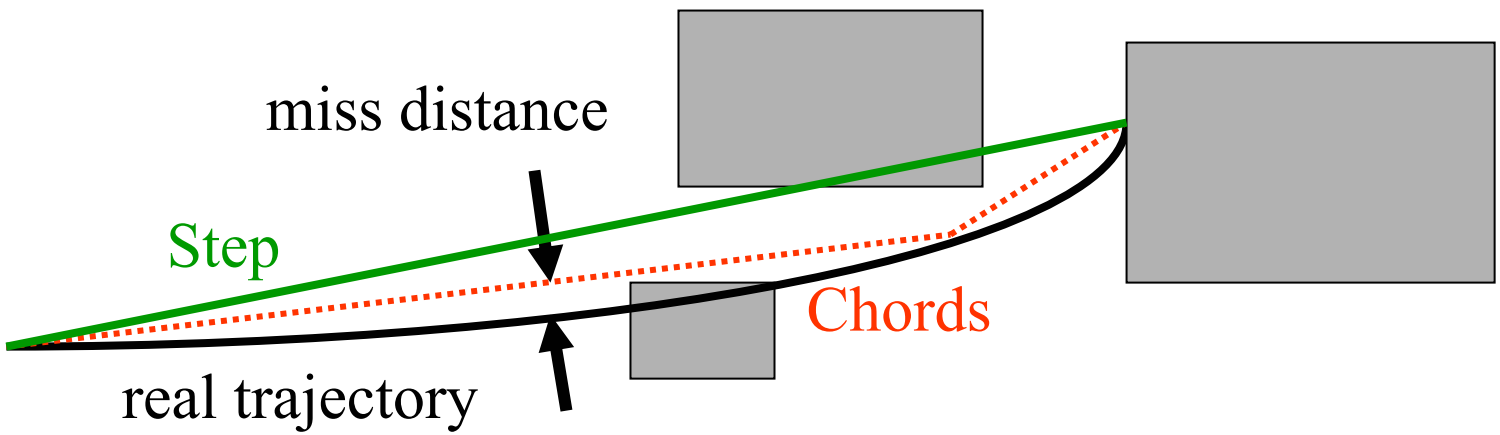


- We determine the chord segments so that they closely approximate the curved path.
- We use the chords to interrogate the Navigator, to see whether the track has crossed a volume boundary.



Magnetic field

- You can set the accuracy of the volume intersection,
  - by setting a parameter called the “miss distance”
    - it is a measure of the error in whether the approximate track intersects a volume.
    - Default “miss distance” is 3 mm.
- One step can consist of more than one chords.
  - In some cases, one step consists of several turns.



Magnetic field

- Magnetic field class
  - Uniform field :

    `G4UniformMagField` class object
  - Non-uniform field :

    Concrete class derived from `G4MagneticField`
- Set it to `G4FieldManager` and create a Chord Finder.

```
G4FieldManager* fieldMgr =
  G4TransportationManager::GetTransportationManager()
        ->GetFieldManager();
fieldMgr->SetDetectorField(magField);
fieldMgr->CreateChordFinder(magField);
```

PART V

Detector Description:

Visualization attributes

& optimization technique

Visualization of Detector

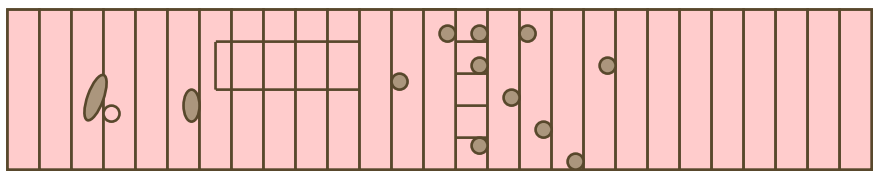
- Each logical volume can have associated a `G4VisAttributes` object
  - Visibility, visibility of daughter volumes
  - Color, line style, line width
  - Force flag to wire-frame or solid-style mode
- For parameterised volumes, attributes can be dynamically assigned to the logical volume
- Lifetime of 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

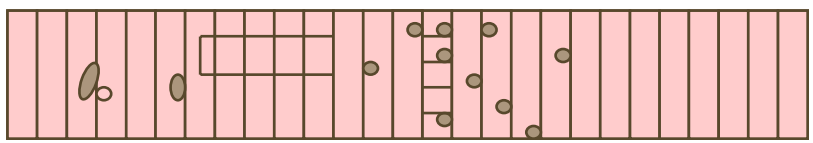
Volume Intersection Optimisation

- Encountering volumes is very costly
  - for simple physics it can take 80% of CPU time
  - Must try to avoid intersection calculations
- '*Smart voxels*' optimise intersections
  - Much less need to tune geometry
  - Can handle 'flat' CAD geometries



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
```

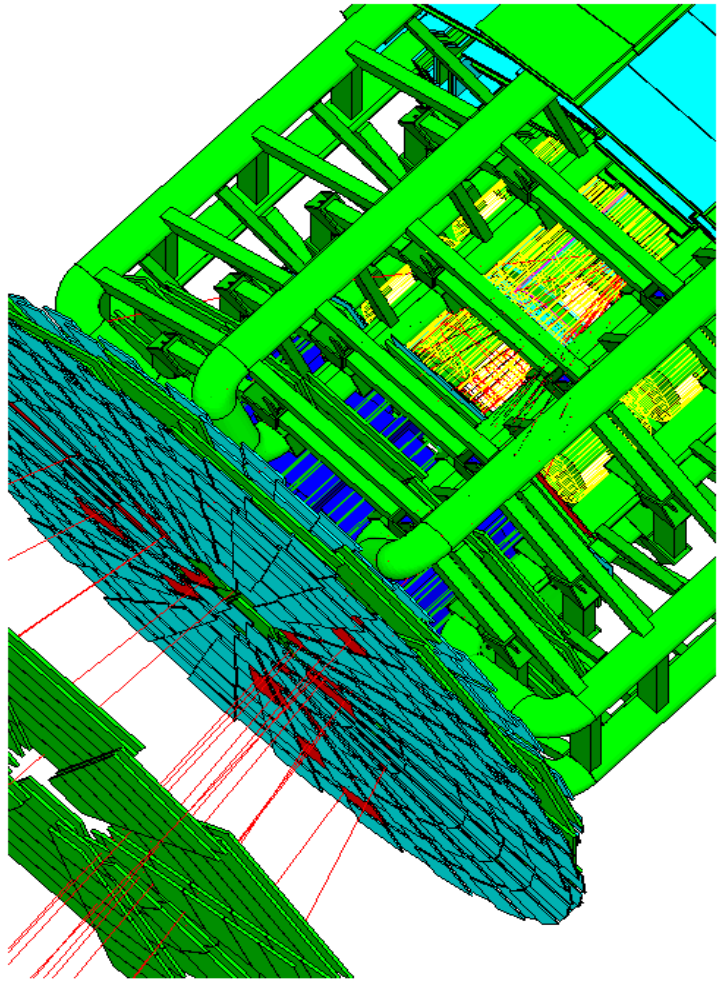
Visualising voxel structure

- 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 VI



Detector Description:

Advanced features

Detector Description
Advanced features

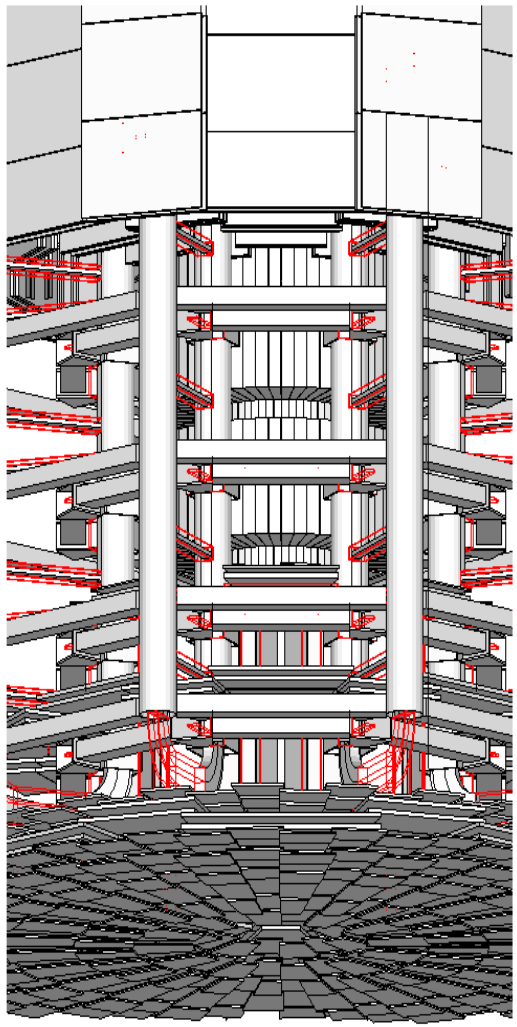
- *Debugging tools*
- *Grouping volumes*
- *Reflections of volumes and hierarchies*
- *User defined solids*
- *Interface to CAD systems*

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 influenced 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
  - It intersects volumes directly, using their graphical representations.
- 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 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 - 2

- Example layout:

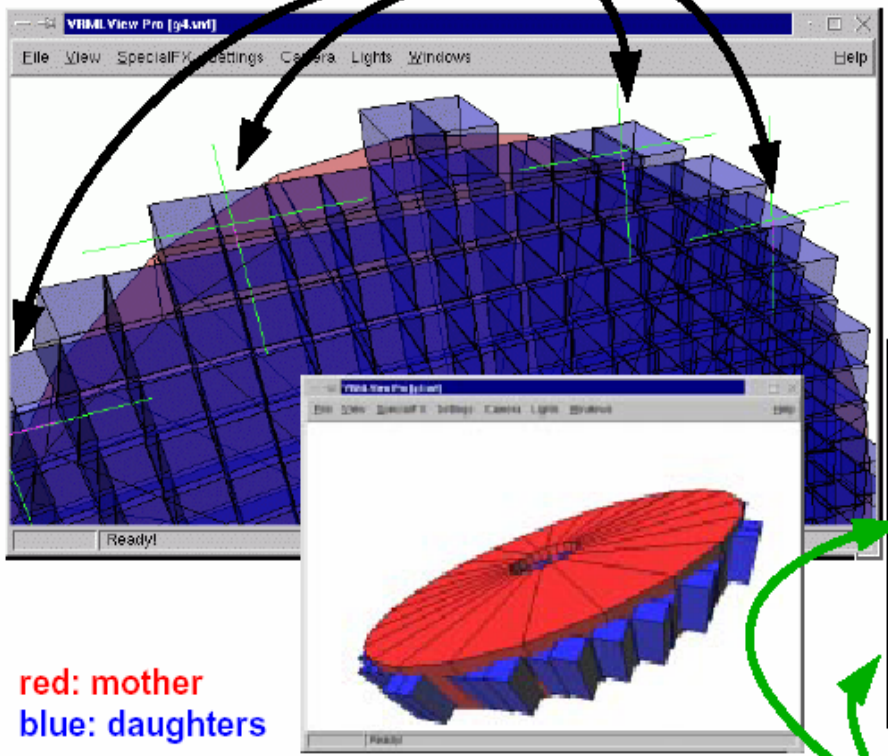
```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list truncated)
  length (cm)    -----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

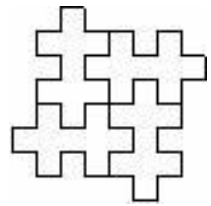
Output:

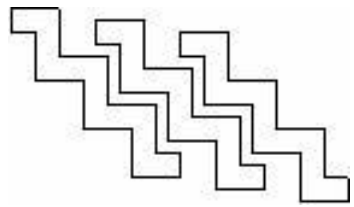
```
delta=59.3416
vol 1: point=(560.513,1503.21,-141.4)
vol 2: point=(560.513,1443.86,-141.4)
A -> B:
[0]:  ins=[2]  PVName=[NewWorld:0] Type=[N] ...
[1]:  ins=[0]  PVName=[ECalEndcap:0] Type=[N] ..
[2]:  ins=[1]  PVName=[ECalEndcapO7:38] Type=[N]

B -> A:
[0]:  ins=[2]  PVName=[NewWorld:0] Type=[N] ...
```

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

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

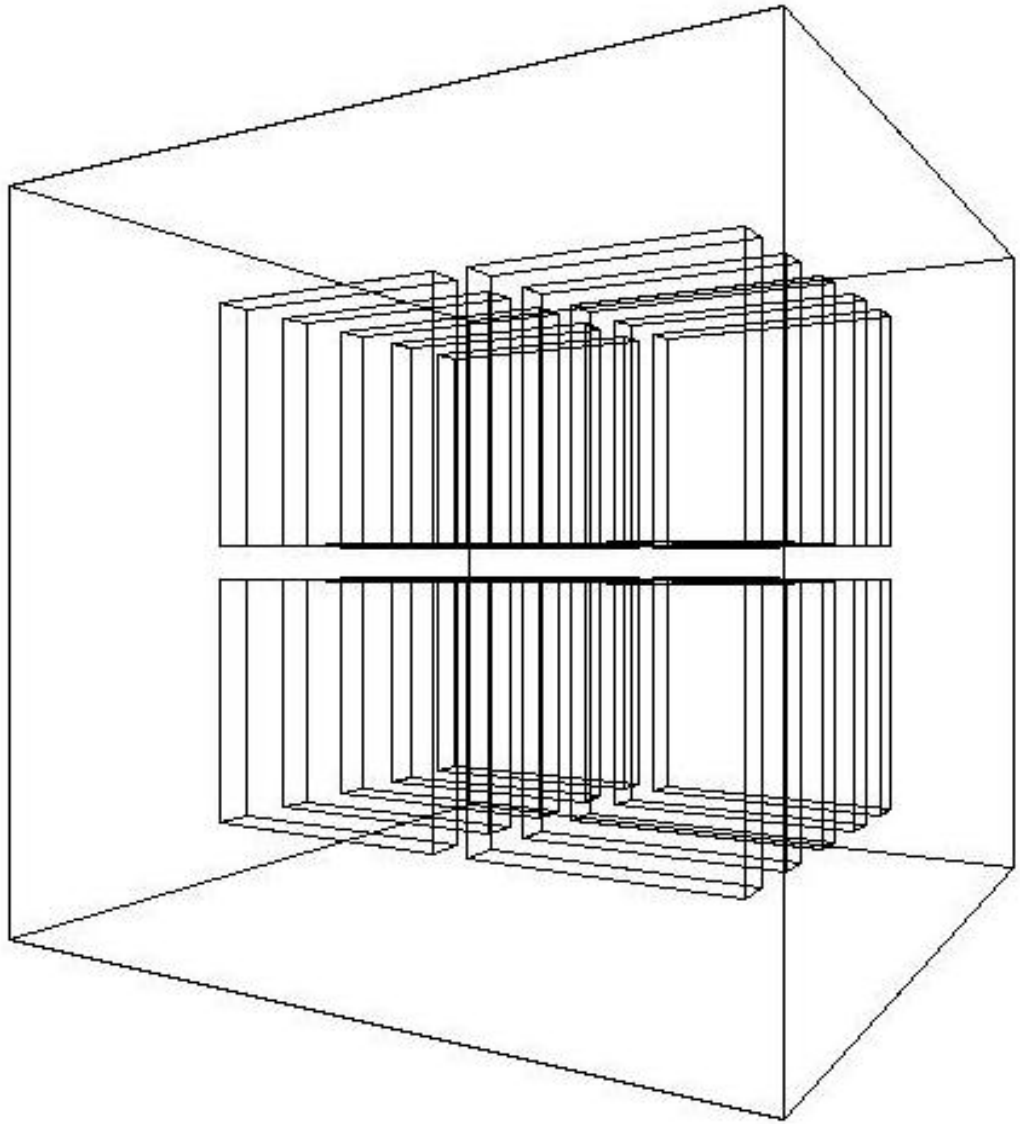
```
G4AssemblyVolume( G4LogicalVolume* volume,
                  G4ThreeVector& translation,
                  G4RotationMatrix* rotation);
```

- 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 solids

- `G4ReflectedSolid`
  - utility class representing a solid shifted from its original reference frame to a new *reflected* 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 can be applied to CSG and specific solids

Reflecting hierarchies of volumes - 1

`G4ReflectionFactory::Place(…)`

- 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
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
```

User defined solids

- All solids should derive from `G4VSolid` and implement its abstract interface
  - will guarantee the solid is treated as any other solid predefined in the kernel
- Basic functionalities required for a solid
  - Compute distances to/from the shape
  - Detect if a point is inside the shape
  - Compute the surface normal to the shape at a given point
  - Compute the extent of the shape
  - Provide few visualization/graphics utilities

What a solid should reply to…- 1

```
EInside Inside(const G4ThreeVector& p) const;
```

- *Should return, considering a predefined tolerance:*
  - `kOutside` - *if the point at offset* `p` *is outside the shapes boundaries*
  - `kSurface` - *if the point is close less than* `Tolerance/2` *from the surface*
  - `kInside` - *if the point is inside the shape boundaries*

```
G4ThreeVector SurfaceNormal(const G4ThreeVector& p) const;
```

- *Should return the outwards pointing unit normal of the shape for the surface closest to the point at offset* `p`.

```
G4double DistanceToIn(const G4ThreeVector& p,
                      const G4ThreeVector& v) const;
```

- *Should return the distance along the normalized vector* `v` *to the shape from the point at offset* `p`. *If there is no intersection, returns* `kInfinity`. *The first intersection resulting from 'leaving' a surface/volume is discarded. Hence, it is tolerant of points on the surface of the shape*

What a solid should reply to…- 2

```
G4double DistanceToIn(const G4ThreeVector& p) const;
```

- *Calculates the distance to the nearest surface of a shape from an outside point* `p`. *The distance can be an underestimate*

```
G4double DistanceToOut(const G4ThreeVector& p,
                       const G4ThreeVector& v,
                       const G4bool  calcNorm=false,
                             G4bool* validNorm=0,
                             G4ThreeVector* n=0) const;
```

- *Returns the distance along the normalised vector* `v` *to the shape, from a point at an offset* `p` *inside or on the surface of the shape. Intersections with surfaces, when the point is less than* `Tolerance/2` *from a surface must be ignored. If* `calcNorm` *is* `true`, *then it must also set* `validNorm` *to either:*
  - `True` - *if the solid lies entirely behind or on the exiting surface. Then it must set* `n` *to the outwards normal vector (the Magnitude of the vector is not defined)*
  - `False` - *if the solid does not lie entirely behind or on the exiting surface*

```
G4double DistanceToOut(const G4ThreeVector& p) const;
```

- *Calculates the distance to the nearest surface of a shape from an inside point* `p`. *The distance can be an underestimate*

Solid: more functions…

```
G4bool CalculateExtent(const EAxis pAxis,
                       const G4VoxelLimits& pVoxelLimit,
                       const G4AffineTransform& pTransform,
                             G4double& pMin, G4double& pMax) const;
```

- *Calculates the minimum and maximum extent of the solid, when under the specified transform, and within the specified limits. If the solid is not intersected by the region, return* `false`*, else return* `true`

Member functions for the purpose of visualization:

```
void DescribeYourselfTo (G4VGraphicsScene& scene) const;
```

- *"double dispatch" function which identifies the solid to the graphics scene*

```
G4VisExtent GetExtent () const;
```

- *Provides extent (bounding box) as possible hint to the graphics view*

Interface to CAD systems

- Models imported from CAD systems can describe the solid geometry of detectors made by large number of elements with the greatest accuracy and detail
  - A solid model contains the purely geometrical data representing the solids and their position in a given reference frame
- Solid descriptions of detector models can be imported from CAD systems
  - e.g. Euclid & Pro/Engineer
    - using STEP AP203 compliant protocol
- Tracking in BREP solids created through CAD systems is supported

How to import CAD geometries

- Detector geometry description should be modularized
  - By sub-detector and sub-detector components
  - Each component in a separate STEP file
- `G4AssemblyCreator` and `G4Assembly` classes from the *STEPinterface* module should be used to read a STEP file generated by a CAD system and create the assembled geometry in Geant4
  - Geometry is generated and described through BREP shapes
  - Geometry modules for each component are assembled in the user code

Importing STEP models: example -1

```
G4AssemblyCreator MyAC("tracker.stp");
  // Associate a creator to a given STEP file.
MyAC.ReadStepFile();
  // Reads the STEP file.
STEPentity* ent=0;
  // No predefined STEP entity in this example.
  // A dummy pointer is used.
MyAC.CreateG4Geometry(*ent);
  // Generates GEANT4 geometry objects.

void *pl =  MyAC.GetCreatedObject();
  // Retrieve vector of placed entities.
G4Assembly* assembly = new G4Assembly();
  // An assembly is an aggregation of placed entities.
assembly->SetPlacedVector(*(G4PlacedVector*)pl);
  // Initialise the assembly.
```

Importing STEP models:
example - 2

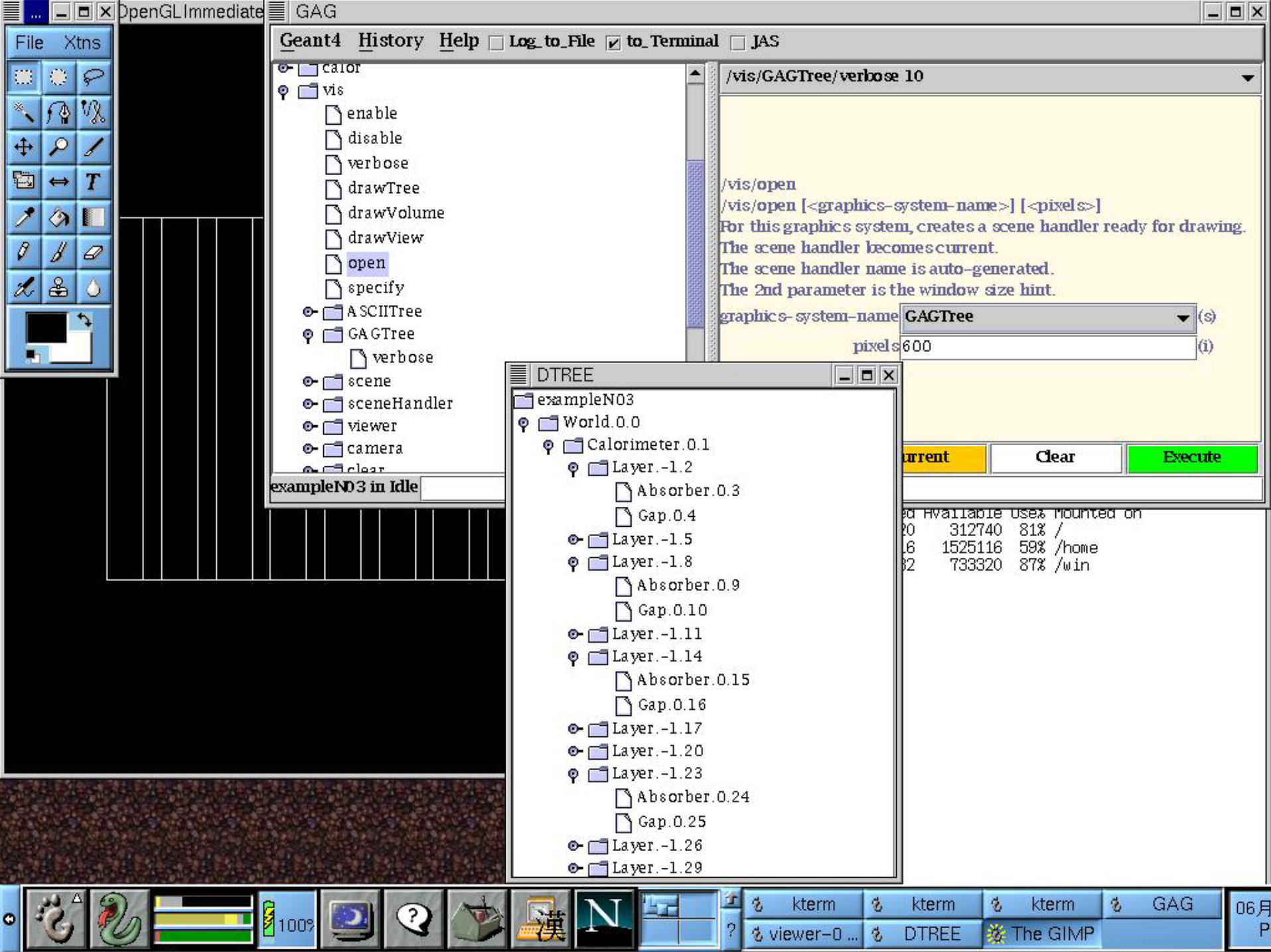
```
G4int solids = assembly->GetNumberOfSolids();
   // Get the total number of solids among all entities.
for(G4int c=0; c<solids; c++)
   // Generate logical volumes and placements for each solid.
{
   ps = assembly->GetPlacedSolid(c);
   G4LogicalVolume* lv =
      new G4LogicalVolume(ps->GetSolid(), Lead, "STEPlog");
   G4RotationMatrix* hr = ps->GetRotation();
   G4ThreeVector* tr = ps->GetTranslation();
   G4VPhysicalVolume* pv =
      new G4PVPlacement(hr, *tr, ps->GetSolid()->GetName(),
                        lv, experimentalHall_phys, false, c);
}
```

GGE (Graphical Geometry Editor)

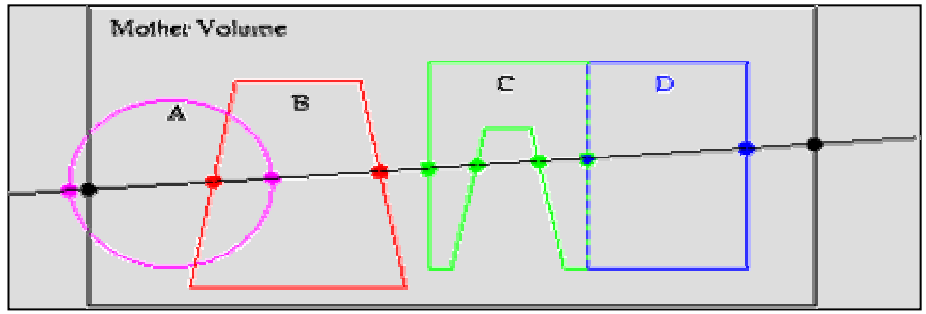
- Implemented in JAVA, GGE is a graphical geometry editor compliant to Geant4. It allows to:
  - Describe a detector geometry including:
    - materials, solids, logical volumes, placements
  - Graphically visualize the detector geometry using a Geant4 supported visualization system, e.g. DAWN
  - Store 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
GAG
Geant4 History Help Log_to_File to_Terminal JAS
/vis/GAGTree/verbose 10
/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
exampleN03 in Idle
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
kterm
kterm
kterm
GAG
viewer-0 ...
DTREE
The GIMP

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