Geant 4

Detector Description - Advanced

http://geant4.cern.ch

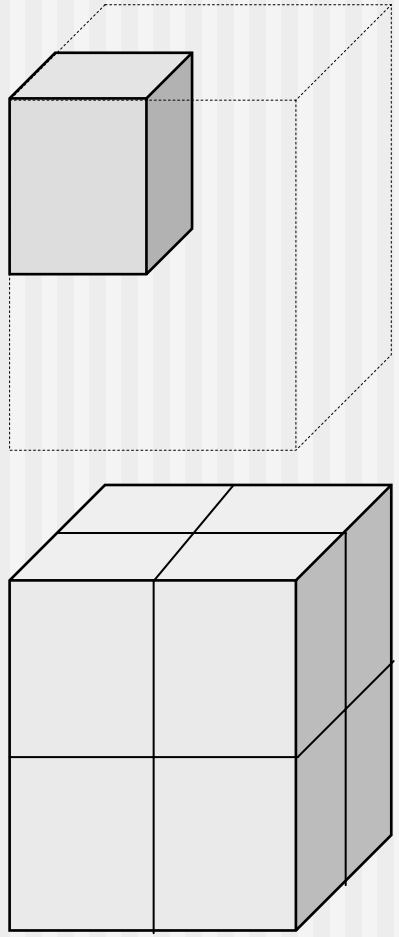
PART I

Detector Description:

Replicas

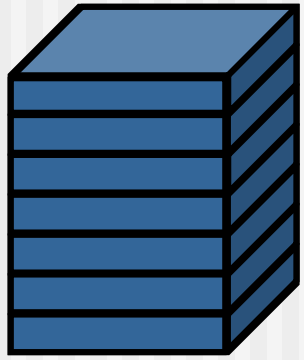
Replicated Physical Volumes

- The mother volume can be sliced into replicas, all of the same size and dimensions.
- Represents many 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);
```

- 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

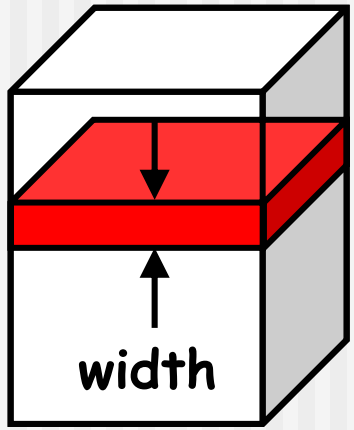
Replica – axis, width, offset

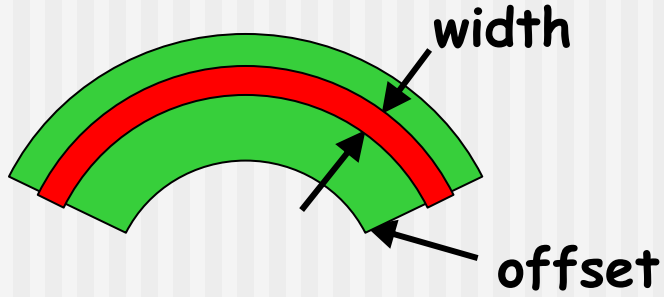
- Cartesian axes - **kXaxis, kYaxis, kZaxis**
  - offset not be used
  - Center of n-th daughter is given as

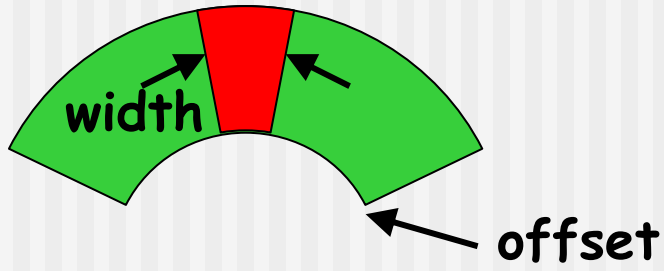
    -width*(nReplicas-1)*0.5+n*width
- Radial axis - **kRaxis**
  - Center of n-th daughter is given as

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

    width*(n+0.5)+offset







Replication
example – 6 sectors of a cylinder

```
G4double tube_dPhi = twopi;
 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 = twopi/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);
```

mother volume

logical volume to be replicated

6 replicas along the kPhi axis, of width 2π / 6

# of replicas

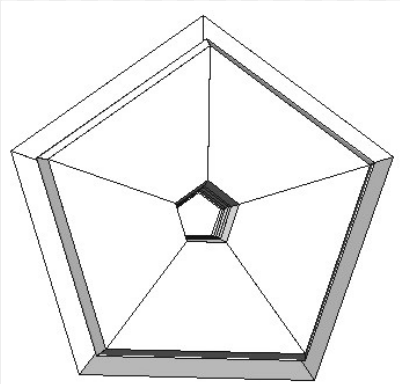
width

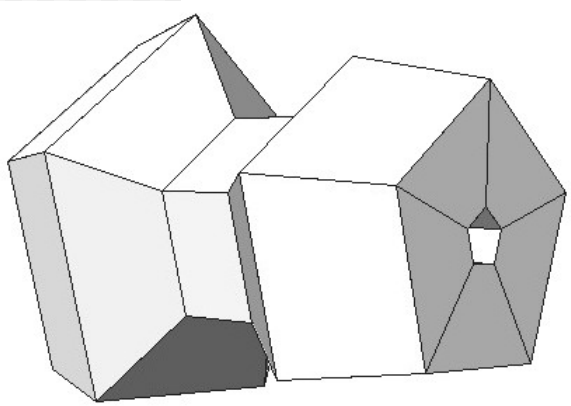
PART II

Detector Description:

Advanced solids, booleans

BREPS:
G4BREPSolidPolyhedra





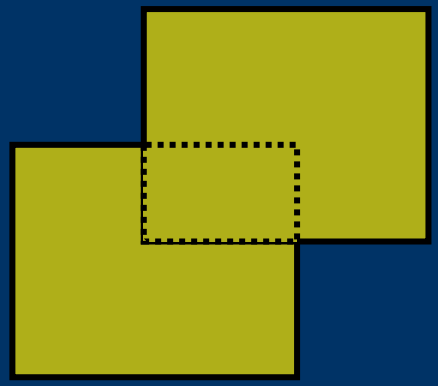
```
G4BREPSolidPolyhedra(const G4String& pName,
                     G4double phiStart,
                     G4double phiTotal,
                     G4int sides,
                     G4int nZplanes,
                     G4double zStart,
               const G4double zval[],
               const G4double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

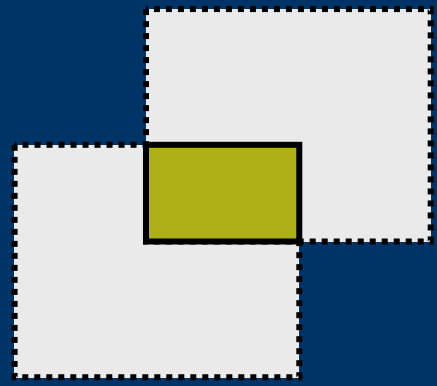
- Solids can be **combined using boolean operations**:
  - **`G4UnionSolid, G4SubtractionSolid, G4IntersectionSolid`**
  - Requires: 2 solids, 1 boolean operation, and an (optional) *transformation* for the 2nd solid
  - 2nd solid is positioned relative to the coordinate system of the 1st solid
  - Result of boolean operation becomes a solid. Thus the third solid can be combined to the resulting solid of first operation.
- Solids to be combined can be either CSG or other Boolean solids.
- Note: tracking cost for the navigation in a complex Boolean solid is proportional to the number of constituent CSG solids

G4UnionSolid

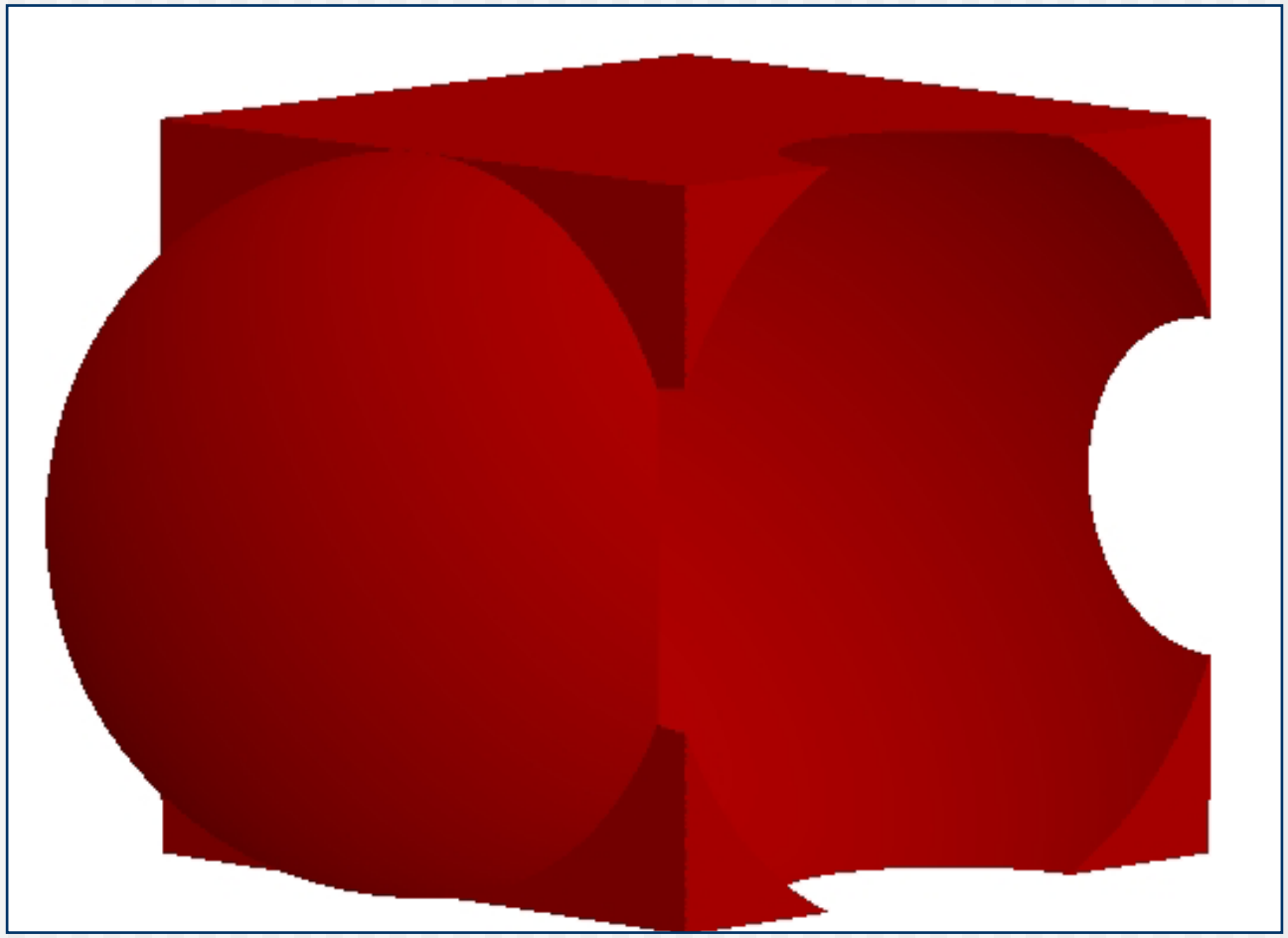


G4SubtractionSolid

G4IntersectionSolid



Boolean solid



Boolean solids, example

```
G4VSolid* box = new G4Box("Box",50*cm,60*cm,40*cm);
G4VSolid* cylinder
 = new G4Tubs("Cylinder",0.,50.*cm,50.*cm,0.,2*M_PI*rad);
G4VSolid* union
 = new G4UnionSolid("Box+Cylinder", box, cylinder);
G4VSolid* subtract
 = new G4SubtractionSolid("Box-Cylinder", box, cylinder,
       0, G4ThreeVector(30.*cm,0.,0.));
G4RotationMatrix* rm = new G4RotationMatrix();
rm->RotateX(30.*deg);
G4VSolid* intersect
   = new G4IntersectionSolid("Box&&Cylinder",
       box, cylinder, rm, G4ThreeVector(0.,0.,0.));
```

translation

rotation

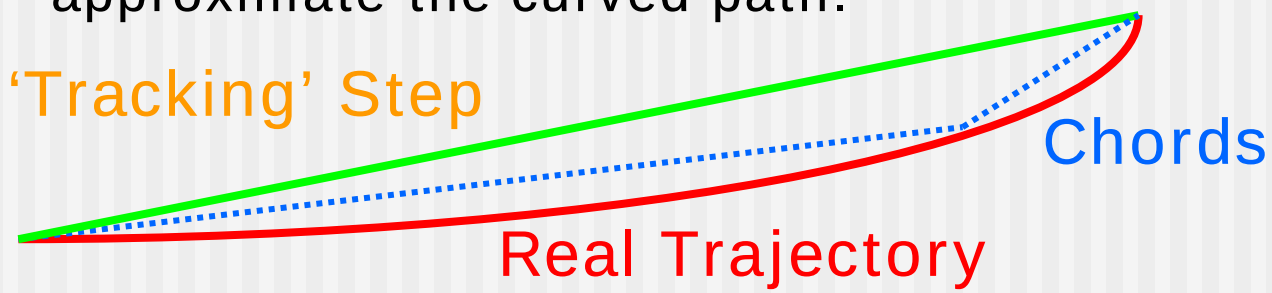
- The origin and the coordinates of the combined solid are the **same as those of the first solid**.

PART III

Detector Description:
Magnetic fields

Field integration

- To propagate a particle inside a field → solve the equation of motion of the particle in the field (e.g. using the Runge-Kutta method for the integration of the equations of motion).
- In specific cases other solvers can also be used:
  - In a uniform field, using the analytical solution.
  - In a smooth but varying field, with RK+helix.
- Using the method to calculate the track's motion in a field, Geant4 breaks up this curved path into linear chord segments.
  - We determine the chord segments so that they closely approximate the curved path.

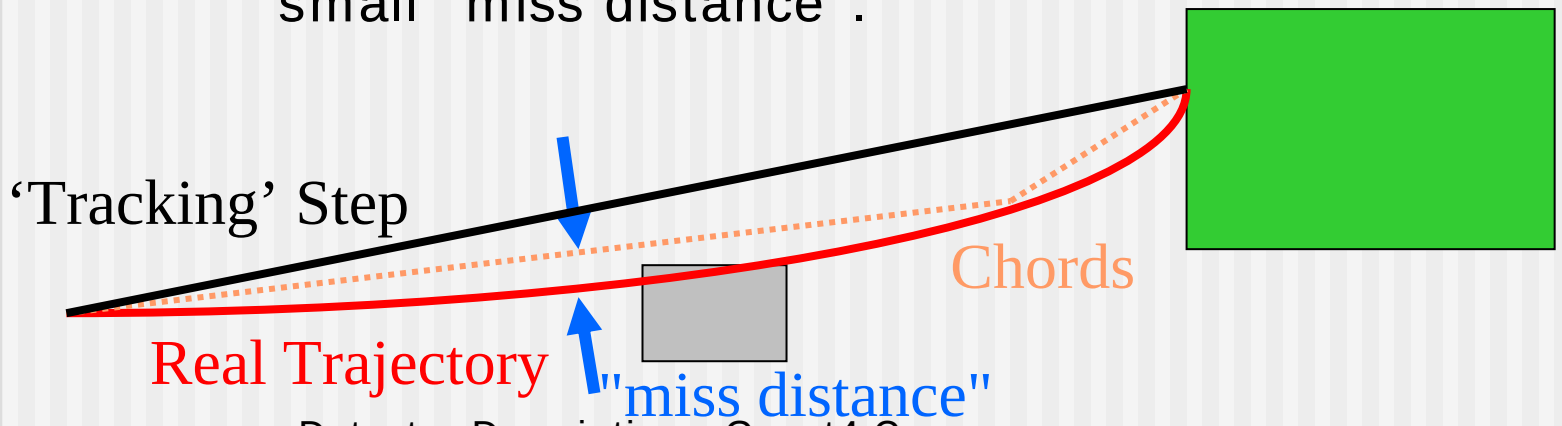


Tracking in field

- Use the chords to interrogate the `G4Navigator`, to see whether the track has crossed a volume boundary.
- One physics/tracking step can create several chords.
  - In some cases, one step consists of several helix turns.
- User 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.

    It is quite **expensive in CPU** performance to set too small "miss distance".



Magnetic field (1)

Create your Magnetic field class

- **Uniform field** :

  Use an object of the **G4UniformMagField** class

```
G4MagneticField* magField =
      new G4UniformMagField(G4ThreeVector(1.*Tesla,0.,0.);
```

- **Non-uniform field** :

  Create your own concrete class derived from **G4MagneticField** and implement **GetFieldValue()**

```
void MyField::GetFieldValue(const double Point[4],
double *field) const
```

Points[0..2] are position in global coordinate system, Point[3] is time, field[0..2] are returning magnetic field

Magnetic field (2)

- Tell Geant4 to use your field:
  - Retrieve the global Field Manager from G4TransportationManager

```
G4FieldManager* globalFieldMgr =
G4TransportationManager::GetTransportationManager()
    ->GetFieldManager();
```

  - **Register the field** to the FieldManager,

```
globalFieldMgr->SetDetectorField(magField);
```

  - and create a Chord Finder

```
globalFieldMgr->CreateChordFinder(magField);
```

- /example/novice/N04 is a good starting point

PART IV

Detector Description:

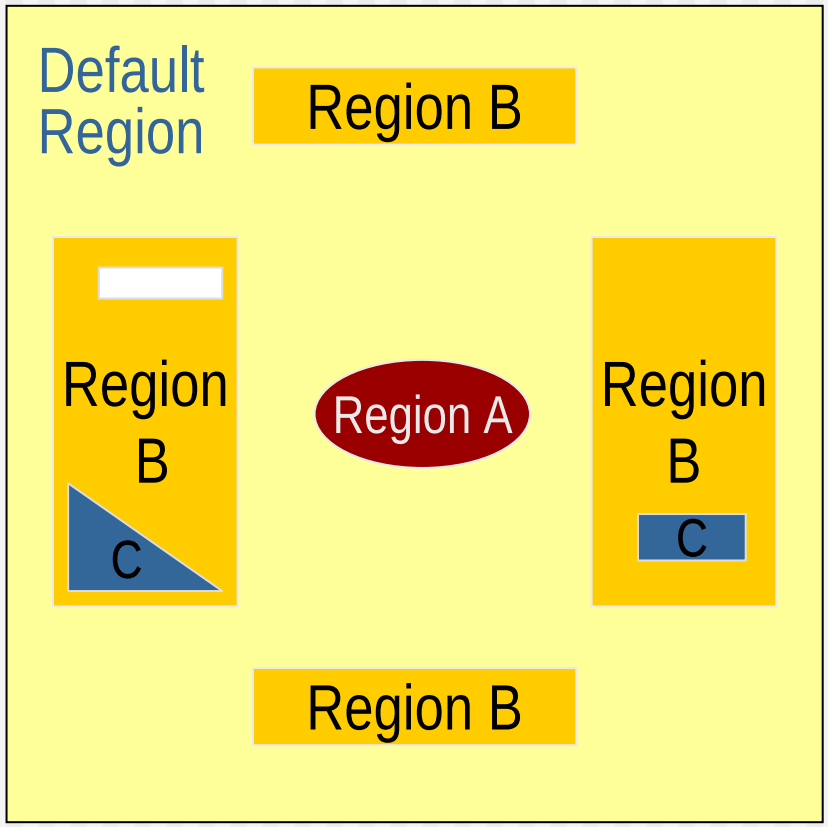
Regions and cut-per-region

Cuts by Region

- Geant4 has had a **unique production threshold** ('cut') expressed in length (i.e. minimum range of secondary)
  - For *all volumes,* but possibly different for each particle (e^+, e^-, γ-rays).
- **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**
- Now 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 γ, electron and positron and only for processes which have infrared divergence (e.g. soft δ-rays or soft bremsstrahlung photons)

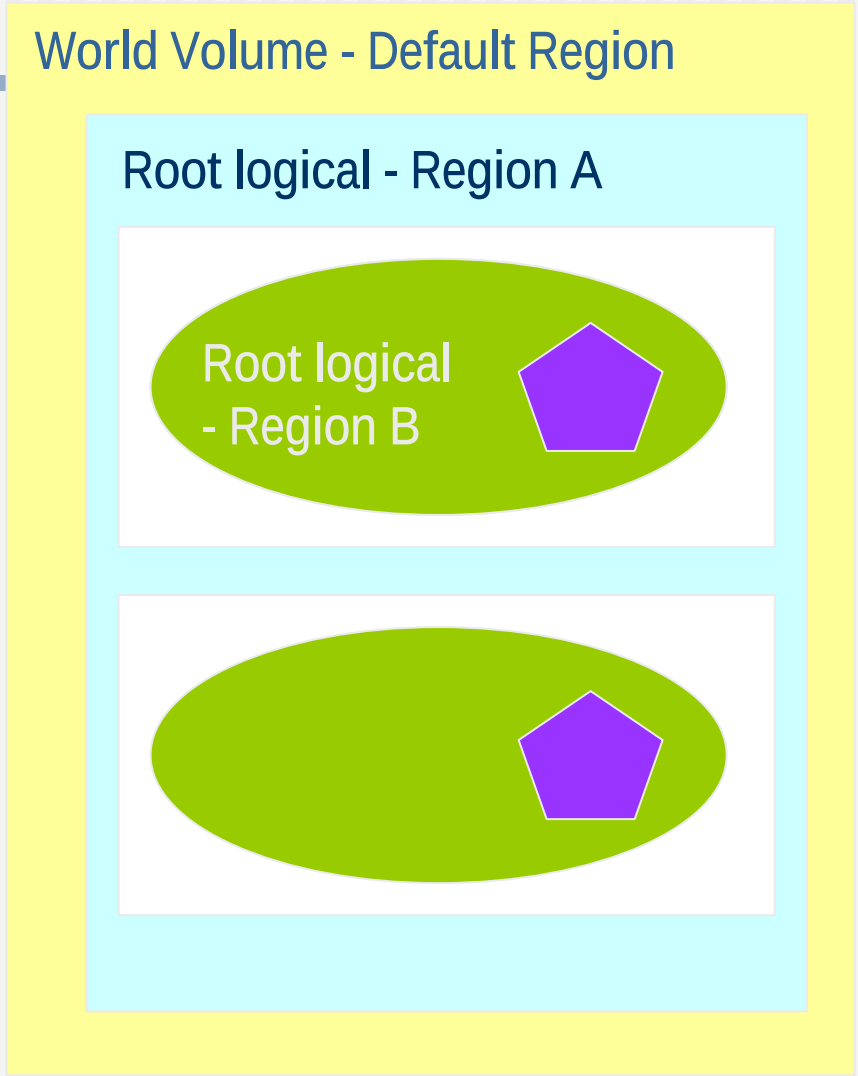
Detector Region

- 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 γ, 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.
- 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



Region and cut – how to use

```
G4Region* emCalorimeter = new G4Region("EM-Calorimeter");
emCalorimeter->AddRootLogicalVolume(emCalorimeterLV);
[...]

G4String regName = “EM-calorimeter";
G4Region* region = G4RegionStore::GetInstance()->GetRegion(regName);

cuts = new G4ProductionCuts;
cuts->SetProductionCut(0.01*mm,G4ProductionCuts::GetIndex("gamma"));
cuts->SetProductionCut(0.1*mm,G4ProductionCuts::GetIndex("e-"));
cuts->SetProductionCut(0.1*mm,G4ProductionCuts::GetIndex("e+"));

region->SetProductionCuts(cuts);
```

create a region

attach a logical volume to the region

retrieve a region

create production cuts

attach cuts to the region

PART V

Detector Description:
Additional tools and debugging

Computing volumes and masses

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

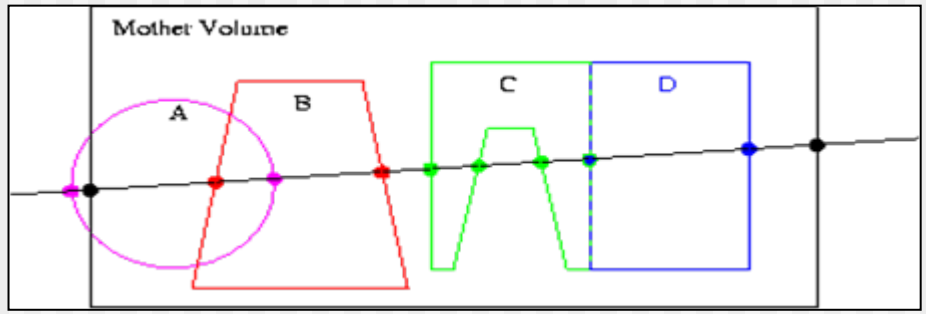
```
G4double GetCubicVolume();
```

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

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

This may take a long computing time, for complex geometries

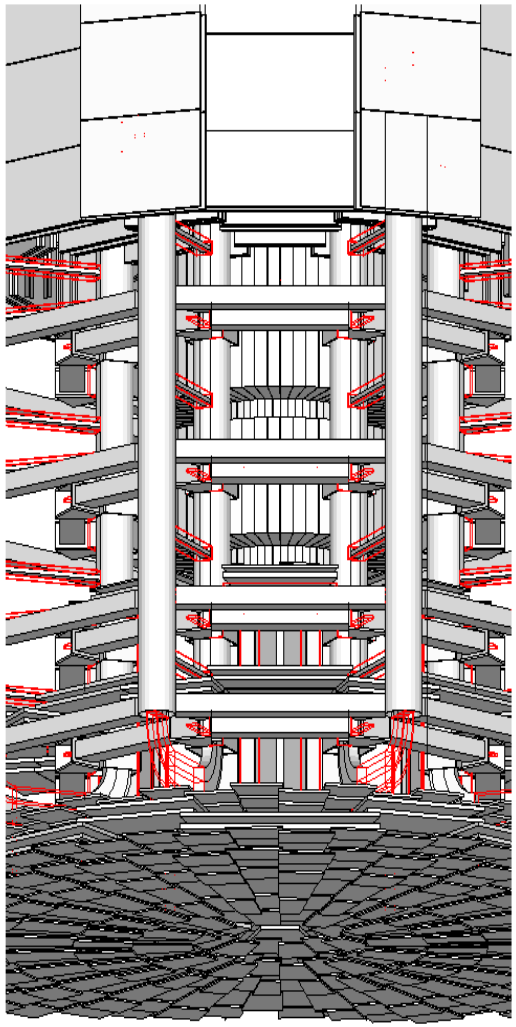
Debugging geometries



- 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: a point in the space cannot be univocally assigned to a physical volume
  - The problem of detecting overlaps between volumes is bounded by the complexity of the solid models description
  - You may have warnings or loops in the simulation
- Utilities are provided for detecting wrong positioning
  - Graphical tools
  - Kernel commands and built-in utilities

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]`

`Idle> geometry/test/run [recursion_flag]` or

`Idle> geometry/test/grid_test [recursion_flag]`

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

`Idle> geometry/test/line_test [recursion_flag]`

to shoot a line along a specified direction and position

`Idle>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 commands

- An other tool to check for overlaps is the **`CheckOverlap()`** method of `G4VPhysicalVolume`

Verifies if the placed volume is overlapping with **existing** daughters or with the mother volume. Returns true if the volume is overlapping

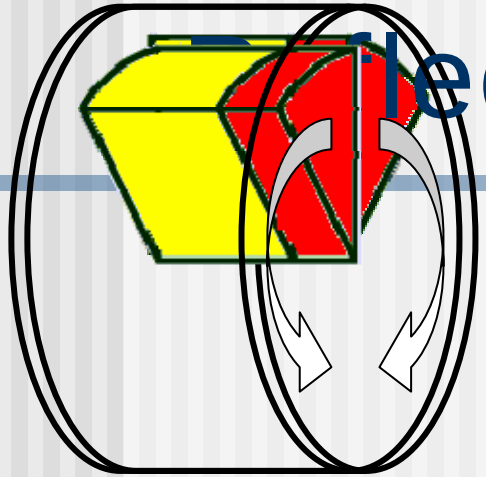
```
myPhysicalVolume->CheckOverlap();
```

It is convenient to run this tool when the geometry is complete (all volumes defined), retrieving all the Physical Volumes from the `G4PhysicalVolumeStore()`

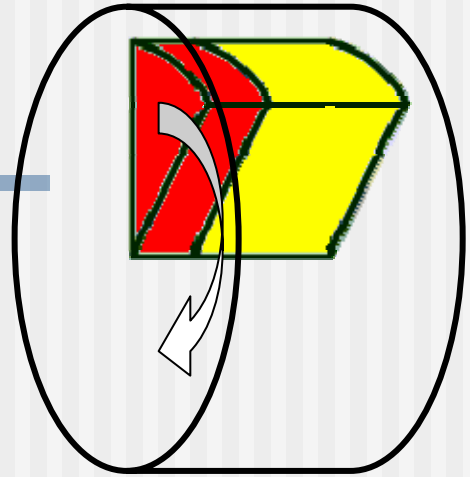
PART VI

Backup slides

Reflecting solids



- Let's take an example of a pair of mirror symmetric volumes.
- Such geometry cannot be made by parallel transformation or 180 degree rotation.



- G4ReflectedSolid (derived from G4VSolid)
  - Utility class representing a solid shifted from its original reference frame to a new mirror symmetric one
  - The reflection (G4Reflect[X/Y/Z]3D) is applied as a decomposition into rotation and translation
- G4ReflectionFactory
  - Singleton object using G4ReflectedSolid for generating placements of reflected volumes
- Reflections are currently limited to simple CSG solids.

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
```