Detector Description:

Geant 4

Advanced Features

http://cern.ch/geant4

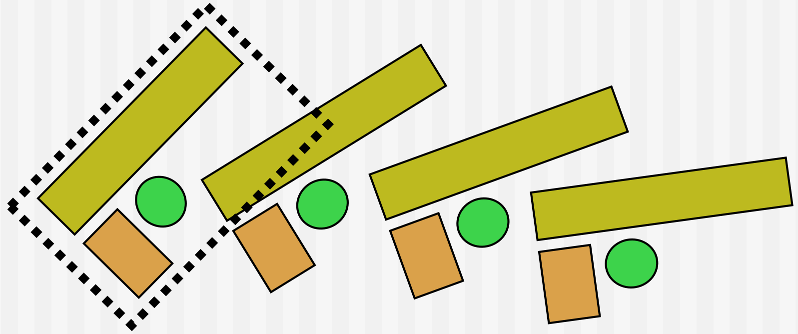
PART V

Detector Description

Advanced features

- *Grouping volumes*
- *Reflections of volumes and hierarchies*
- *Nested parameterisations*
- *Navigation in regular structures*
- *Detector regions*
- *User defined solids*

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

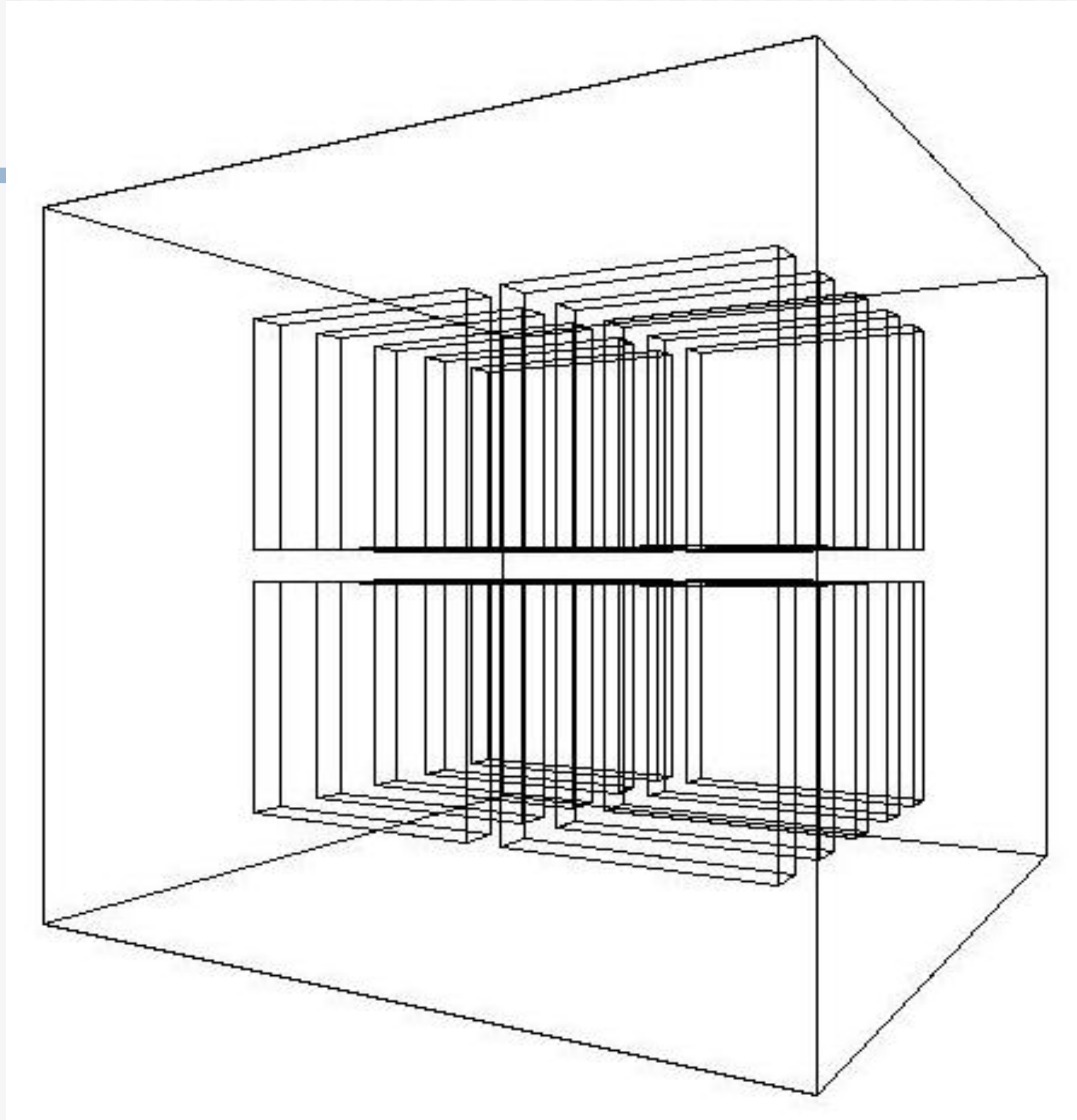
```
G4AssemblyVolume( G4LogicalVolume* volume,
                  G4ThreeVector& translation,
                  G4RotationMatrix* rotation);
```

- 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)

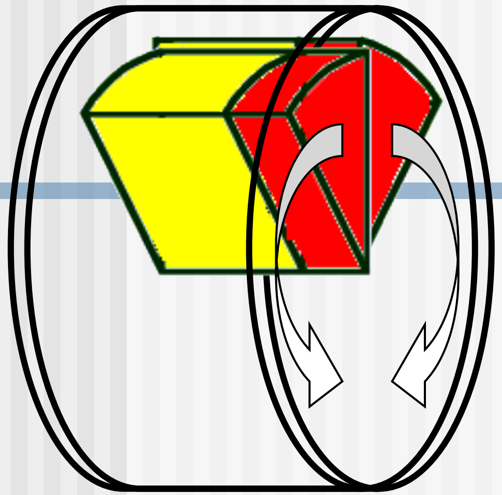
Assembly of volumes:
example -1

```
// 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) );
   }
```

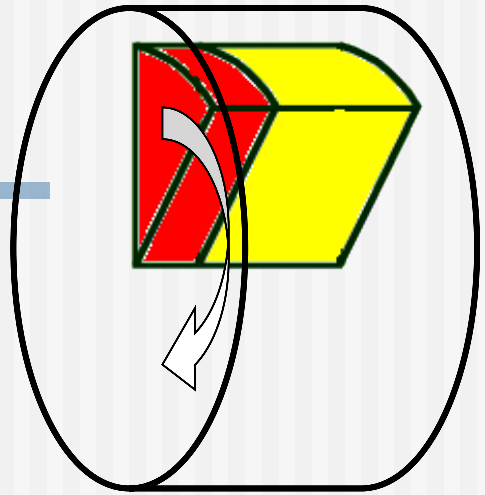
Assembly of volumes:
example -2



Reflecting volumes



- 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**
  - 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:
  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

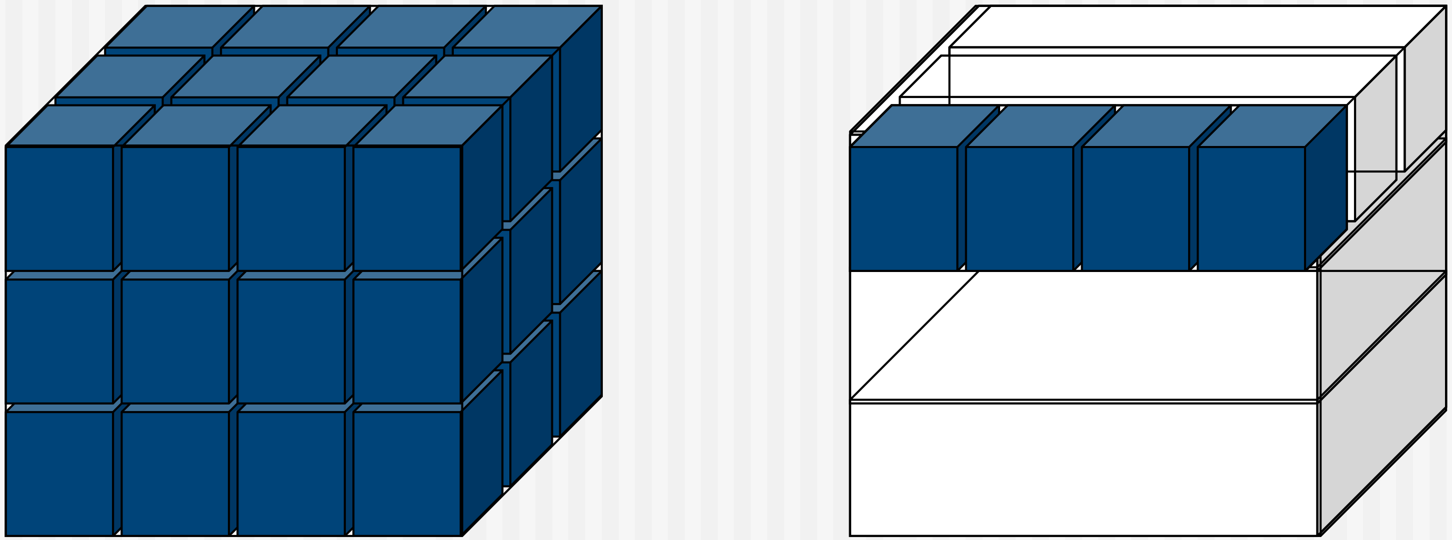
```
G4PhysicalVolumesPair
Replicate(const G4String&  name,       // the actual name
          G4LogicalVolume* LV,         // the logical volume
          G4LogicalVolume* motherLV,   // the mother volume
          Eaxis            axis        // axis of replication
          G4int            replicaNo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

PART V

Nested Parameterisations

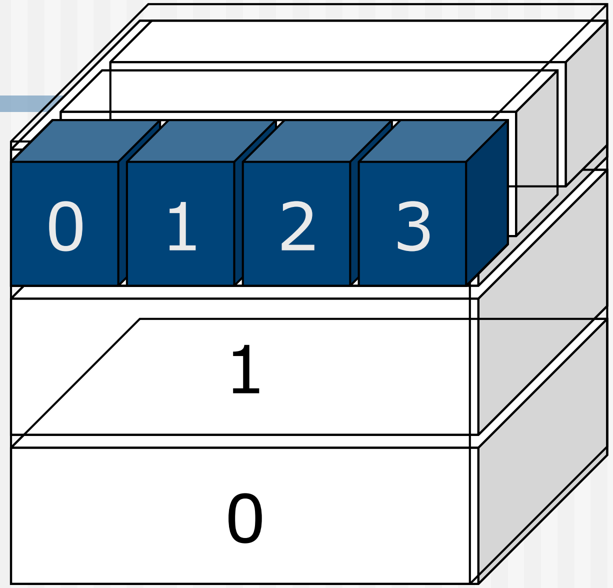
3D Parameterised Volumes

- Typical use-case: geometry with three-dimensional repetition of same shape and size of volumes without gap between them
  - Materials of such volumes are changing according to position
    - E.g. voxels made by CT Scan data (DICOM)
- Solution: instead of direct 3D parameterised volume…
  - *Use replicas for 1st and 2nd axes sequentially, and then use one-dimensional parameterisation along the 3rd axis !*



- Less memory for geometry optimisation and faster navigation for many voxels

G4VNestedParameterisation



- Given that the geometry is defined as two sequential replicas and then one-dimensional parameterisation…
  - Material of a voxel must be parameterised not only by the copy number of the voxel, but also by the copy numbers of its ancestors
  - Material is indexed by three indices
- **G4VNestedParameterisation** is a special parameterisation class derived from G4VPVParameterisation base class.
  - **ComputeMaterial()** method of **G4VNestedParameterisation** has a touchable object of the parent physical volume, in addition to the copy number of the voxel.
    - Index of first axis = `theTouchable->GetCopyNumber(1);`
    - Index of second axis = `theTouchable->GetCopyNumber(0);`
    - Index of third axis = copy number

Using Nested Parameterisations

- **`G4VNestedParameterisation`** is a specialised kind of the generic **`G4VPVParameterization`** abstract class.
  - It can be used as argument to **`G4PVParameterised`** for defining a parameterised volume
- Nested parameterisation of "placement" type for volumes is not supported
  - All levels used as indices for the materials must be of kind *repeated volume* (either parameterised or replica)
    - There cannot be a level of "placement" volumes in between

Using Nested Parameterisations - 2

- **G4VNestedParameterisation** class has three pure virtual methods which must be implemented
  - in addition to the **ComputeTransformation()** method, which is mandatory for all sub-classes of **G4VPVParameterization**

```
virtual G4Material* ComputeMaterial(G4VPhysicalVolume *currentVol,
                    const G4int repNo, const G4VTouchable *pTouchable=0)=0;
```

  - Returns a material pointer w.r.t. copy numbers of itself and ancestors
  - Typically, returns a default material if the parent touchable pointer is zero

```
virtual G4int GetNumberOfMaterials() const=0;
```

  - Returns the total number of materials which may appear as the return value for the `ComputeMaterial()` method.

```
virtual G4Material* GetMaterial(G4int idx) const=0;
```

  - Returns `idx`-$_{th}$ material.
  - `idx` is <u>not</u> a copy number. `idx = [0, nMaterial-1]`

PART V

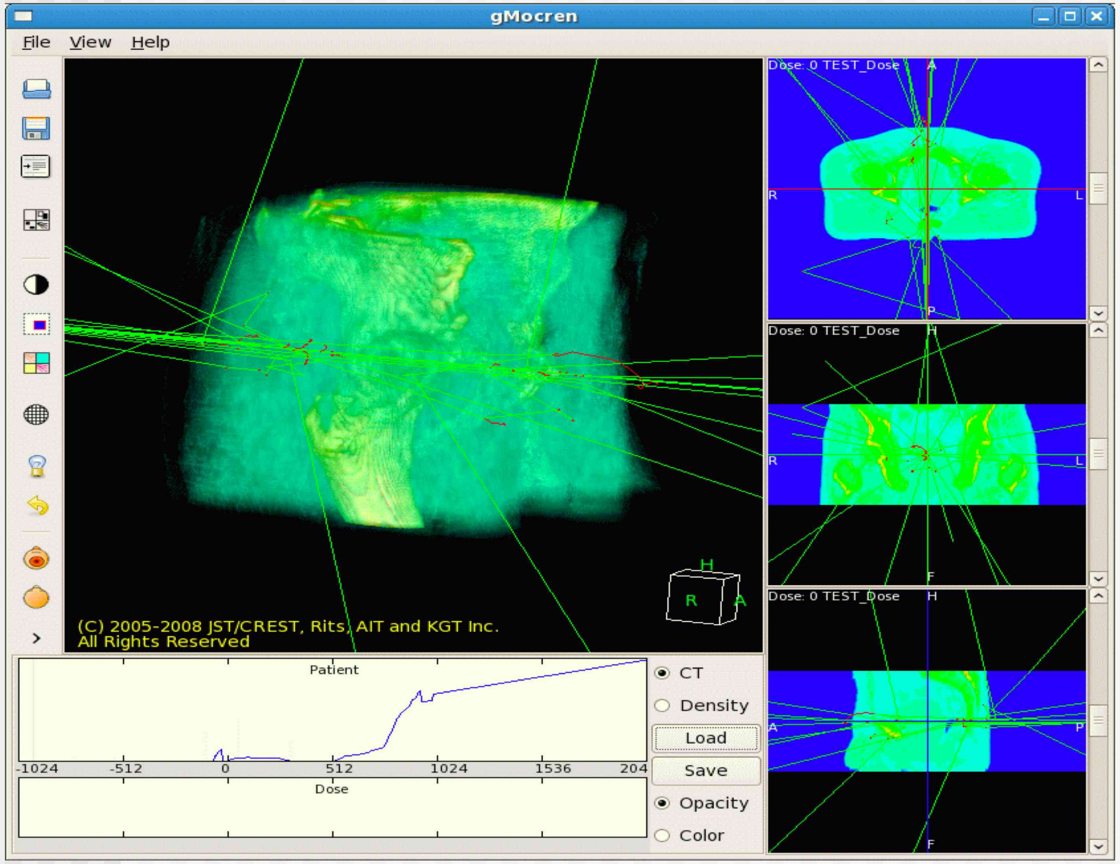
Regular Structures

- Specialised navigation

Regular Patterned geometries

- Typical use-case: DICOM phantoms for medical physics studies
  - Regular pattern of volumes
    - In particular: three-dimensional grids of boxes
    - Up to hundreds millions voxels parameterised by material
    - Challenge for CPU time in initialisation and consumed memory
  - Geometry optimisation for tracking redundant (regular structure)
    - Could avoid to build optimisation tree (*smart-voxels*)
      - Big gain in CPU at initialisation and consumed memory
    - Could consider specialised algorithm for determining neighboring voxels in the structure to interserct at tracking time
    - Could optionally further reduce number of voxels in the structure by collapsing neighboring voxels with same material
- Solution: **`G4RegularNavigation`**

G4RegularNavigation



Tracks on regular structure visualized with gMocren

- Algorithm automatically activated for geometries defined as regular structures
  - Requires the specification of a special kind of parameterisation, through `G4PhantomParameterisation`
  - Location of a point inside a voxel can be done in a fast way, transforming the position to the coordinate system of the container volume and doing a simple calculation
  - Optimisation can be optionally provided by skipping the frontiers of two voxels with same material assigned, so that bigger steps can be done

    `SetSkipEqualMaterials(bool);`

    - From `G4PhantomParameterisation`
    - To avoid when the number of materials is very big or when the physical step is small compared to the voxel dimensions

Defining a regular geometry…

- Need to first create an object of type **G4PhantomParameterisation** (with voxels and materials), **param**, where to …
  - define the voxel dimensions in the parameterisation
    - `param->SetVoxelDimensions( halfX, halfY, halfZ );`
  - specify the number of voxels in the three dimensions
    - `param->SetNoVoxel( nVoxelX, nVoxelY, nVoxelZ );`
  - define the list of materials to associate to voxels according to indeces
    - `param->SetMaterialIndices( mateIDs );`
- A "container" volume (box) for the voxel structure has to be defined
  - The voxel structure must completely fill the container box
  - Its physical volume be assigned to the phantom parameterisation, assuring that voxels are completely filling the structure
    - `param->BuildContainerSolid( cont_phys );`
    - `param->CheckVoxelsFillContainer( x, y, z );`
  - Assign the parameterisation as a normal parameterised volume (e.g. **patient_phys**) and set it as a regular structure
    - `patient_phys->SetRegularStructureId( 1 );`

PART V

Regions

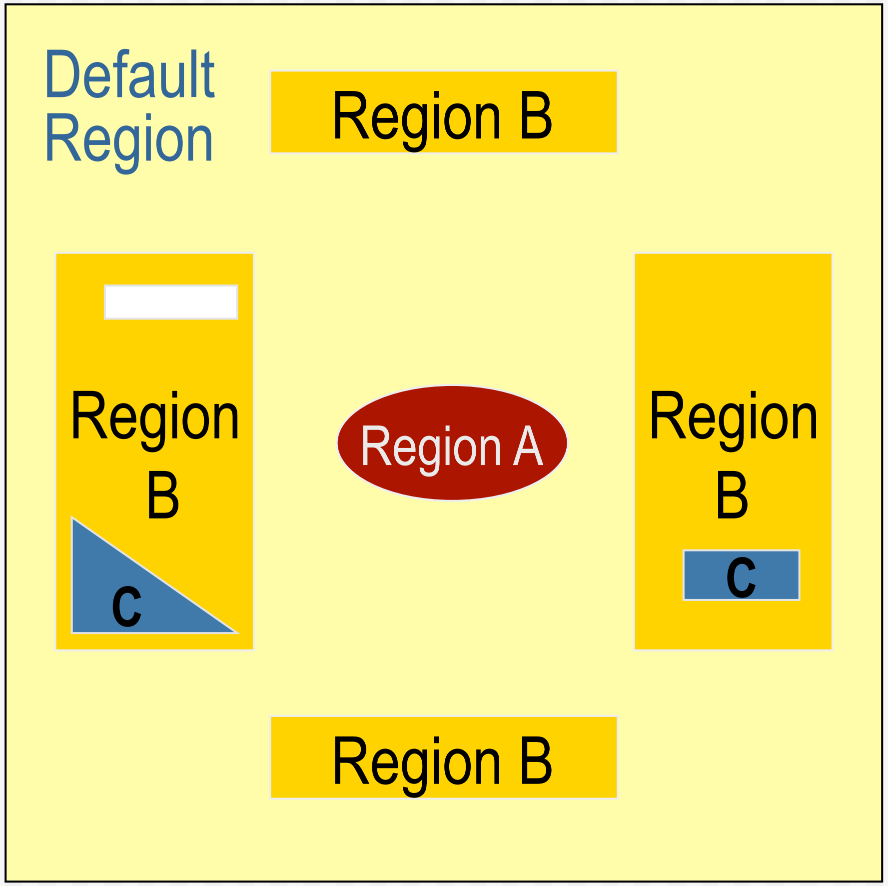
- *Cuts by Region*
- *Concept of Region*
- *Example*

Cuts by Region

- Geant4 used to have 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-,gamma)
- 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

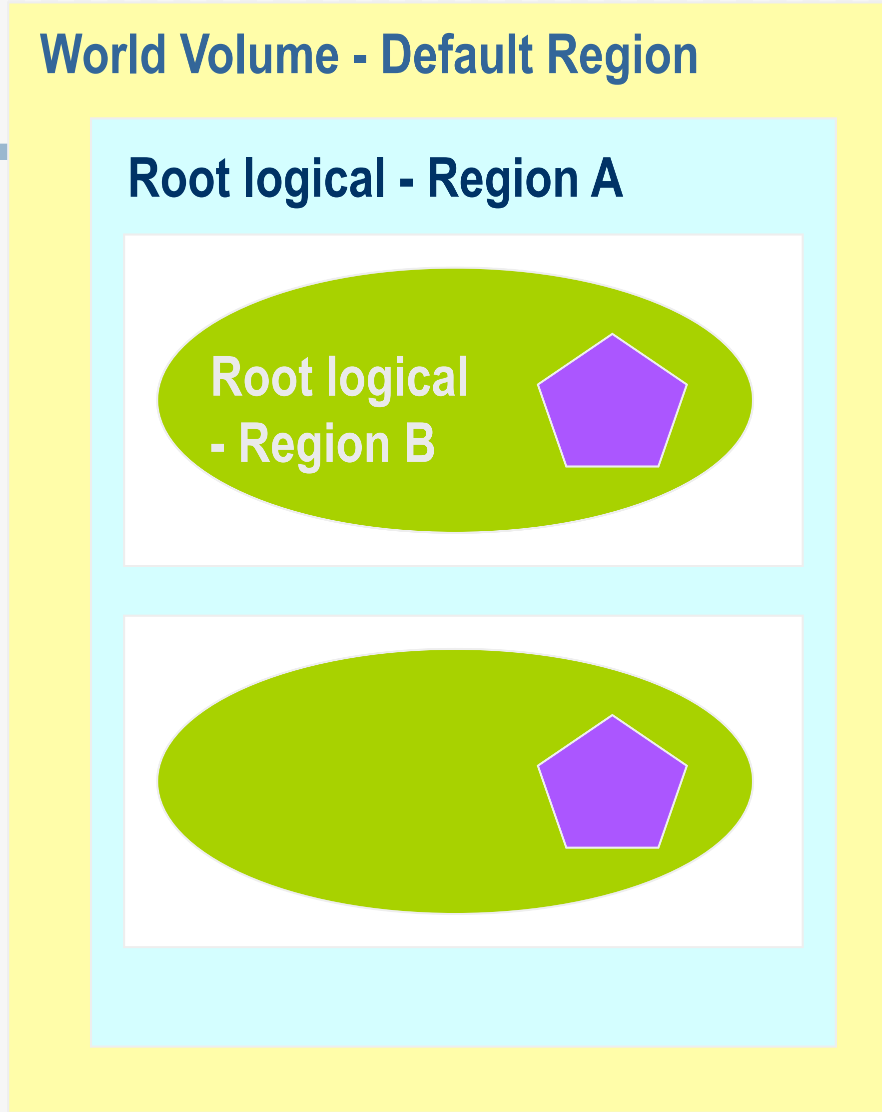
Detector Region

- 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



Setting up regions: example

```
// Create a region
G4Region* emCalorimeter = new G4Region("EM-Calorimeter");
// Attach a logical volume to the region
emCalorimeter->AddRootLogicalVolume(emCalorimeterLV);
[…]
// Retrieve the region by its name
G4Region* region
  = G4RegionStore::GetInstance()->GetRegion("EM-Calorimeter");
// Create production cuts
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+"));
// Attach cuts to the region
region->SetProductionCuts(cuts);
```

PART V

Geometry customisation

- *User defined solids*

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*

Computation of area and geometrical volume

- Any <u>solid</u> must provide the ability to compute its own surface area and geometrical volume:

```
G4double GetSurfaceArea();
G4double GetCubicVolume();
```

- Must return an estimation of the solid area and volume in internal units
- Overloaded by the concrete solid implementation to perform exact computation of the quantity
  - Should eventually "cache" the computed value, such that it is NOT recomputed each time the method is called