Geant 4

Detector Description: Basics

http://cern.ch/geant4

PART IV

Detector Description: tools

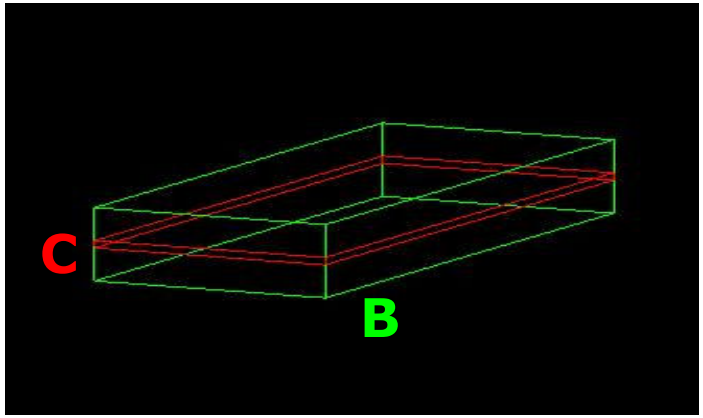
- *How to identify a volume in the hierarchy*
- *Optimisation techniques*
- *Debugging tools*

Identifying a volume

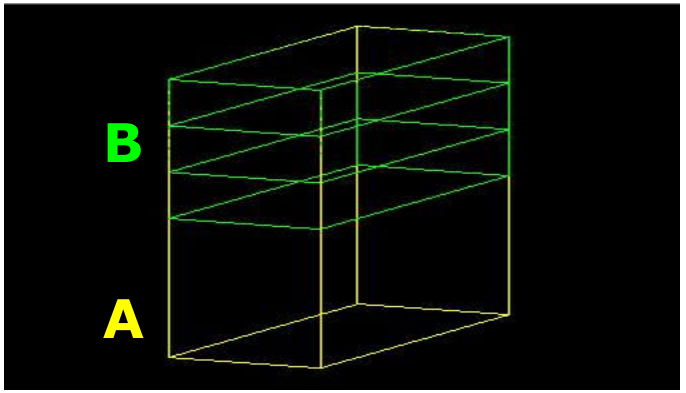
- *Touchables*

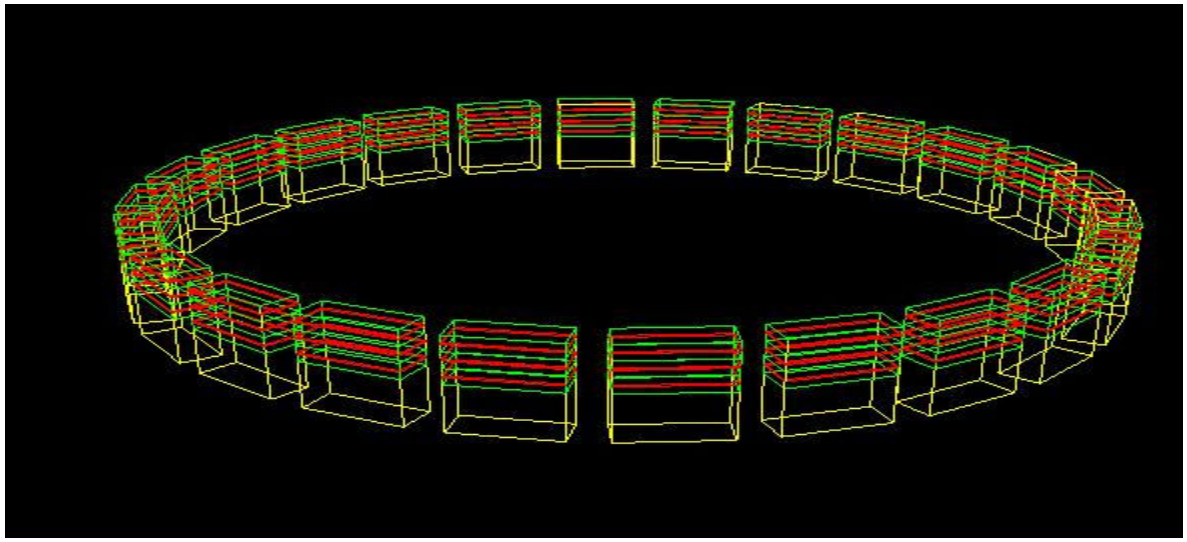
How to identify a volume in a unique way?

- Suppose a geometry is made of sensitive layers C which are placed in a volume B



- Volume B is a daughter volume of a divided volume *A*

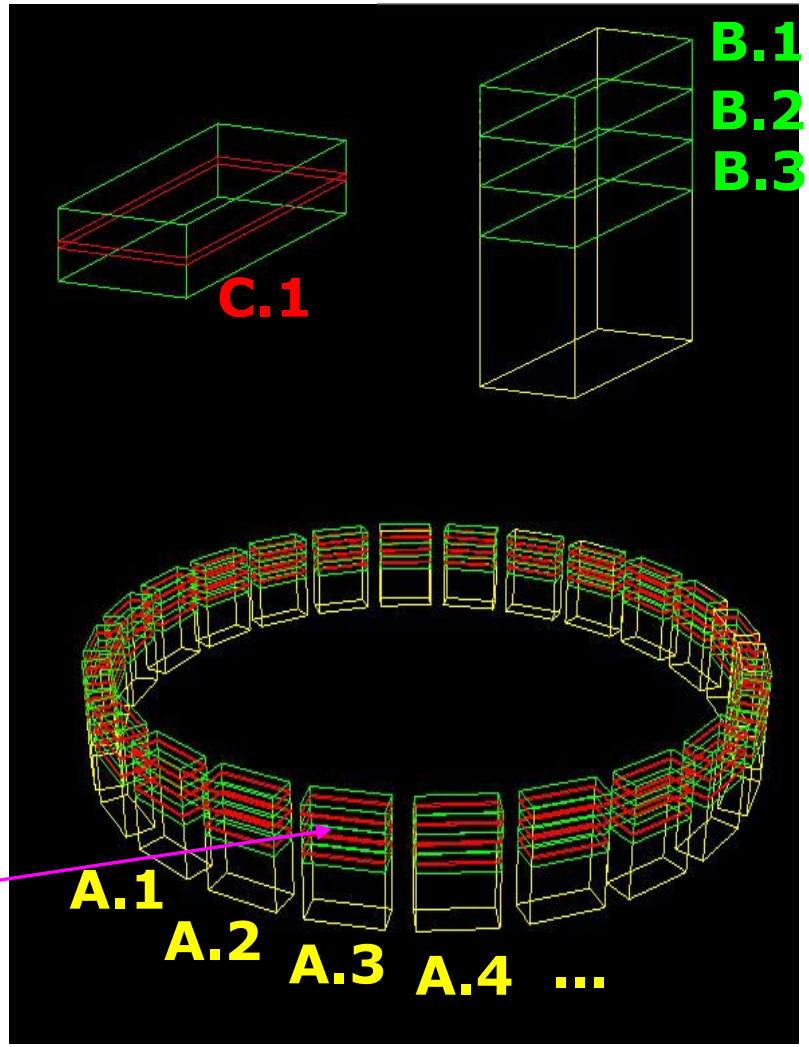




- Volume *A* has 24 positions in the world
- While in the 'logical' geometry tree volume *C* is represented by just one physical volume, in the real world there are many *C* 'volumes'
- *How can we then identify these volumes C ?*

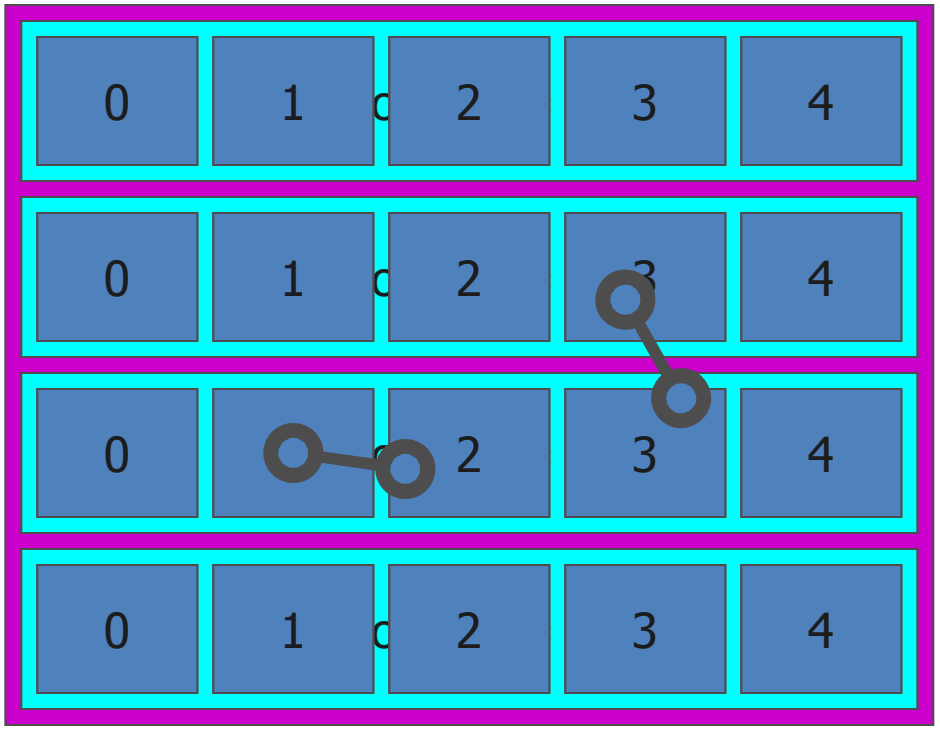
Basics of Touchables

- A touchable for a volume serves the purpose of providing a unique identification for a detector element
- It is a geometrical entity (volume or solid) which has a unique placement in a detector description
  - It can be uniquely identified by providing the copy numbers for all daughters in the geometry hierarchy
  - In our case these are
    - CopyNo of C in B: **1**
    - CopyNo of B in A: **1,2,3**
    - CopyNo of A in the world: **1, .., 24**
  - Example of touchable identification:
    - **A.3/B.2/C.1**



Copy numbers

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



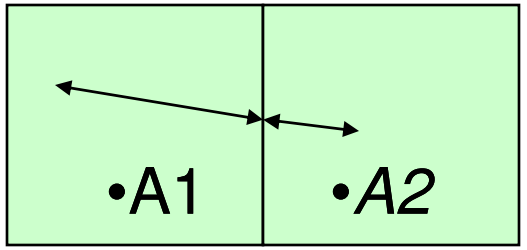
  - *Remember*: geometrical information in `G4Track` is identical to "PostStepPoint". You cannot get the correct copy number for "PreStepPoint" if you directly access to the physical volume
- Use touchable to get the proper copy number, transformation matrix,…

What a touchable provides ?

- **`G4VTouchable`** - a base class for all touchable implementations – defines the following 'requests' (methods) which all touchable have to respond, where depth means always the number of levels up in the tree to be considered:
  - depth = 0 : the bottom level (volume C in B)
  - depth = 1 : the level of its mother volume (volume B in A)
  - depth = 2 : the grandmother volume (volume A in world)
- **`GetCopyNumber(G4int depth =0)`**
  - **returns the copy number of the given level**
- **`GetTranslation(G4int depth = 0); GetRotation(G4int depth=0)`**
  - **return the components of the volume's transformation**
- **`GetSolid(G4int depth =0)`**
  - **returns the solid**
- **`GetVolume(G4int depth =0)`**
  - **returns the physical volume**

Benefits of Touchables in track

- Full geometrical information available
  - to processes, to sensitive detectors, to hits
- All the geometrical information of the particular step should be taken from "**`PreStepPoint`**"
  - Available in **`G4TouchableHistory`** object
    - Copy-number, transformations
  - Accessible via Handles (or *smart-pointers*) to touchables
  - Note: the geometrical information associated with **`G4Track`** is basically the same as "**`PostStepPoint`**"



Touchable & Steps

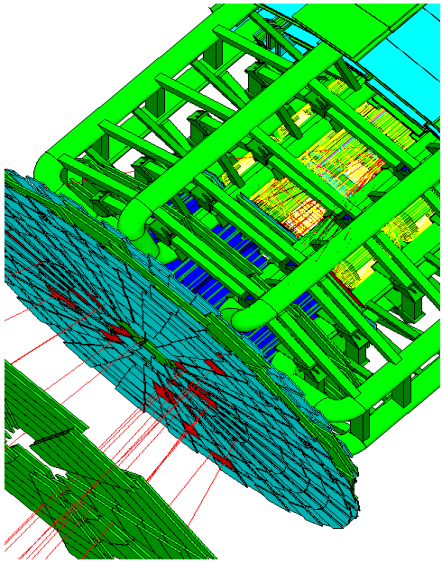
- **`G4Step`** has two **`G4StepPoint`** objects as its starting and ending points. All the geometrical information of the particular step should be got from "**`PreStepPoint`**"
  - Geometrical information associated with **`G4Track`** is basically same as "**`PostStepPoint`**"
- Each **`G4StepPoint`**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

How to use Touchables …

- **`G4TouchableHistory`** has information of the geometrical hierarchy at the point

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

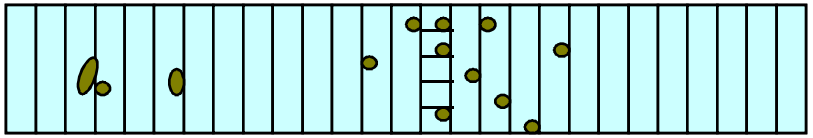
Optimisation Techniques



- *Smart voxels*

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

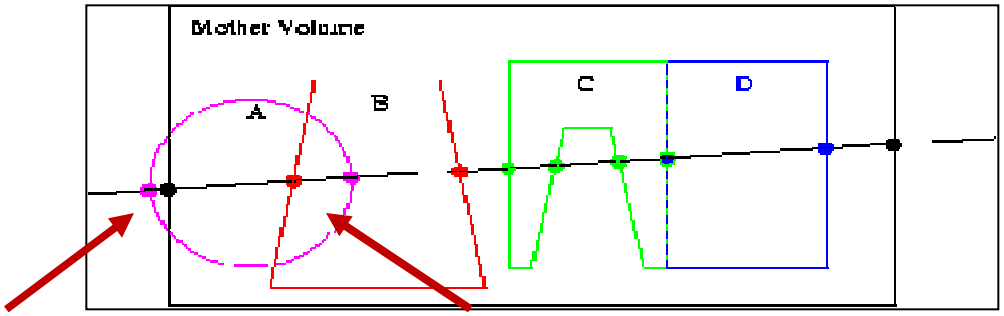
- Detector regions may be excluded from optimisation (ex. for debug purposes)
  - Optional argument in constructor of `G4LogicalVolume` or through provided set methods
    - `SetOptimisation`/`IsToOptimise()`
  - Optimisation is turned on by default
- Optimisation for parameterised volumes can be chosen
  - 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`

Debugging geometries

Debugging tools

- *Optional checks at Construction*
- *DAVID*
- *Run-time commands*
- *OLAP*

Debugging geometries



protruding ***overlapping***

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

Debugging tools:
Overlapping check at Construction

- Constructors of **G4PVPlacement** and **G4PVParameterised** have an optional argument **pSurfChk**:

```
G4PVPlacement(G4RotationMatrix* pRot, …, G4bool pSurfChk=false);
```

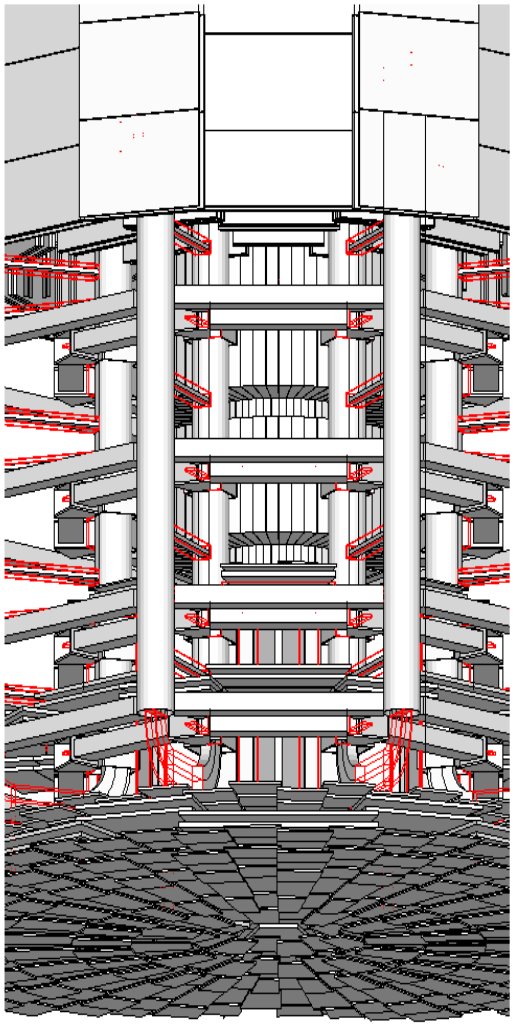
- If this flag is true, overlap check is done at construction
  - A number of points (1000 by default) are randomly sampled on the surface of the volume being created
  - Each of these points are examined
    - if outside of the mother volume, or
    - if inside of already existing other volumes in the same mother volume
  - NOTE: this check may requires lots of **CPU time**
    - Depending on the complexity of geometry
  - Can also be forced on a specific physical volume though the method:

```
G4bool CheckOverlaps(G4int points=1000, G4double tol=0, G4bool verbose=true);
```

- **Worth to try** when first implementing a geometry of some complexity !

Debugging tools: DAVID

- DAVID is a graphical debugging tool for detecting potential intersections of volumes
- Accuracy of the graphical representation can be tuned to the exact geometrical description.
  - physical-volume surfaces are automatically decomposed into 3D polygons
  - intersections of the generated polygons are parsed.
  - If a polygon intersects with another one, the physical volumes associated to these polygons are highlighted in color (red is the default).
- DAVID can be downloaded from the Web as external tool for Geant4
  - http://geant4.kek.jp/GEANT4/vis/DAWN/About_DAVID.html



Debugging run-time commands

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

`geometry/test/run [recursion_flag]` or

`geometry/test/grid_test [recursion_flag]`

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

`geometry/test/cylinder_test [recursion_flag]`

  - shoots lines according to a cylindrical pattern

`geometry/test/line_test [recursion_flag]`

  - to shoot a line along a specified direction and position

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

  - to specify position & direction for the **`line_test`**

- Resolution/dimensions of grid/cylinders can be tuned

Debugging run-time commands - 2

- Example layout:

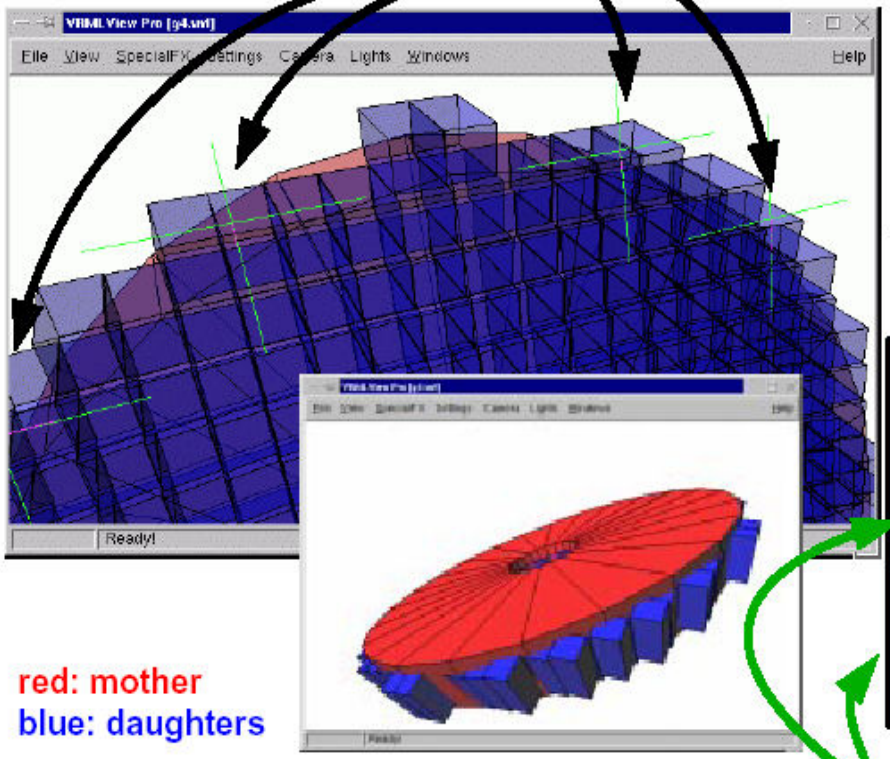
```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list truncated)
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    240           -240      -145.5      -145.5     0       -145.5      -145.5
Which in the mother coordinate system are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
Which in the coordinate system of Tracker[0] are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
Which in the coordinate system of Overlap[0] are:
  length (cm)    ----- start position (cm) -----  ----- end position (cm) -----
    . . .
```

Debugging tools: OLAP

- Adopt 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)