Geometry 3

I.Hrivnacova
IPN, Orsay

Most slides thanks to M. Asai, SLAC

Cours Geant4 @ Paris 2007
4 - 8 June 2007

Contents

- Geometry checking tools
- Basics of Touchable
- Region

Contents

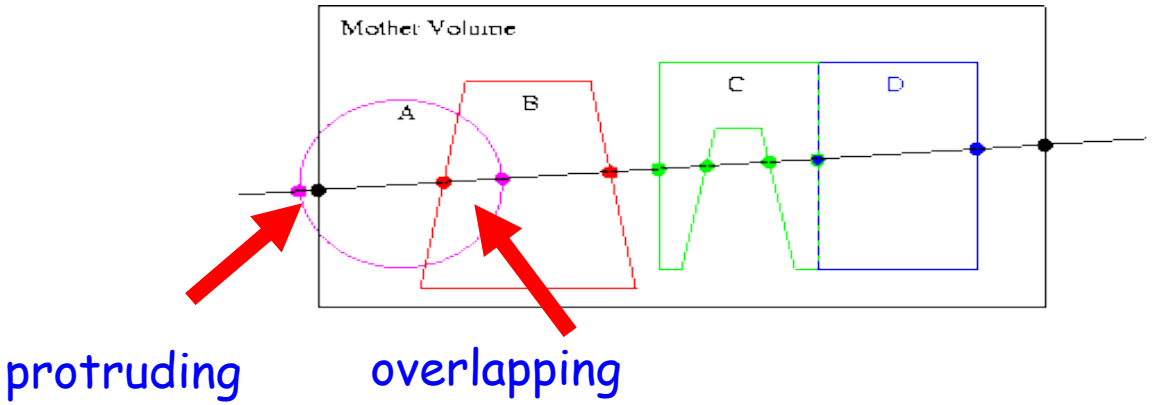
Geometry checking tools

Basics of Touchable

Region

Malformed Geometries

- A protruding volume is a contained daughter 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, neither protruding nor overlapping.
  - The behavior of navigation is unpredictable for such cases.



Debugging 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
  - Optional checks at construction
  - Kernel run-time commands
  - Graphical tools (DAVID, OLAP)

Optional checks 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 the construction.
  - Some number of points are randomly sampled on the surface of creating volume.
  - Each of these points are examined
    - If it is outside of the mother volume, or
    - If it is inside of already existing other volumes in the same mother volume.
  - This check requires lots of CPU time, but it is worth to try at least once when you implement your geometry of some complexity.

Debugging run-time commands

Built-in run-time commands to activate verification tests for the user geometry:

- To start verification of geometry for overlapping regions based on a standard grid setup, limited to the first depth level:
  - `geometry/test/run` or `geometry/test/grid_test`
- To apply the grid test to all depth levels (may require lots of CPU time!)
  - `geometry/test/recursive_test`
- To shoot lines according to a cylindrical pattern
  - `geometry/test/cylinder_test`
- To shoot a line along a specified direction and position
  - `geometry/test/line_test`
- To specify position for the `line_test`
  - `geometry/test/position`
- To specify direction for the `line_test`
  - `geometry/test/direction`

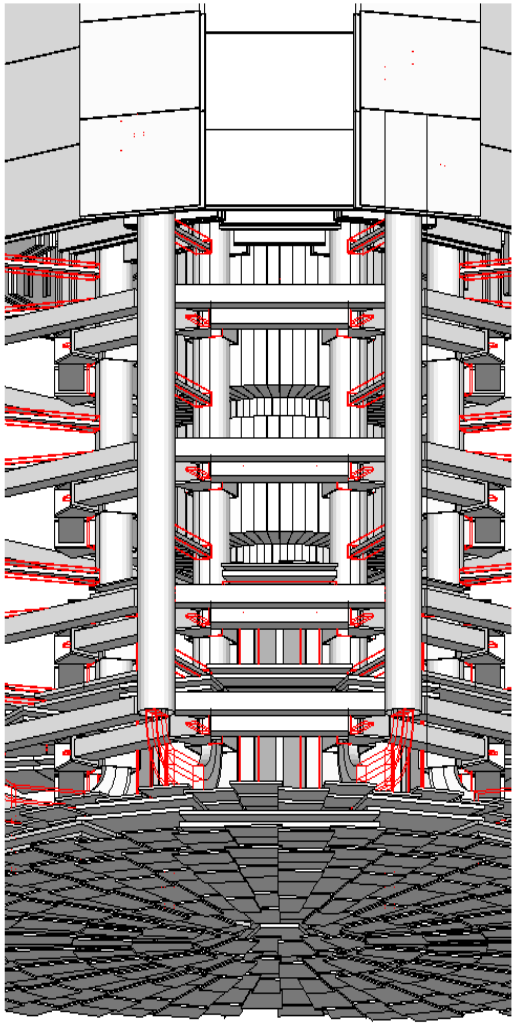
Debugging run-time commands

Example of output:

```
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 Geometries
DAVID

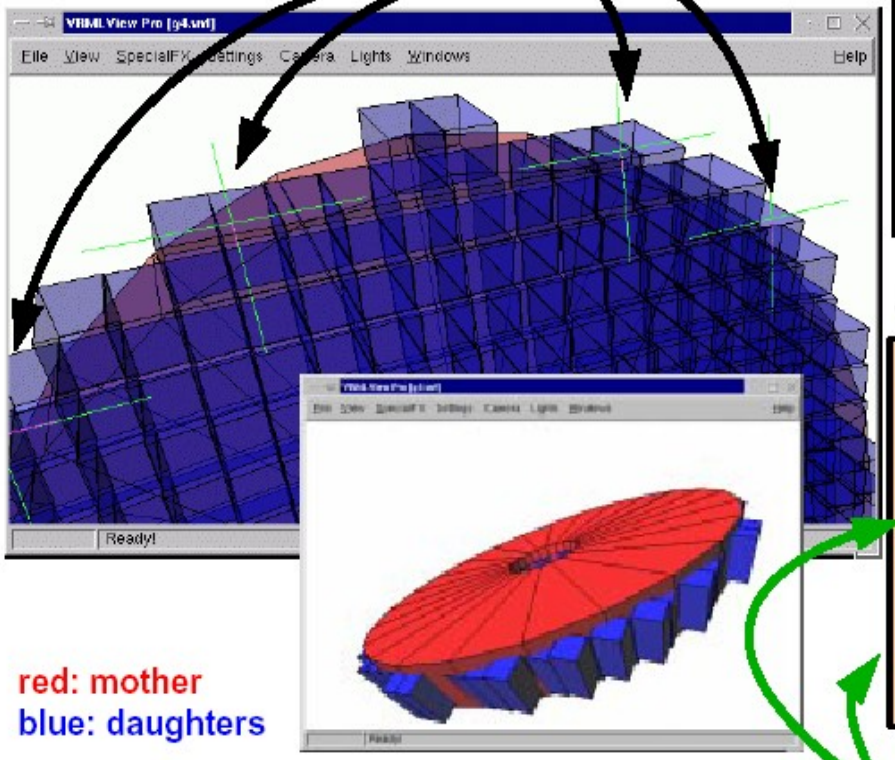
- DAVID is a graphical debugging tool for detecting potential intersections of volumes
- Accuracy of the graphical representation can be tuned to the exact geometrical description.
  - Physical-volume surfaces are automatically decomposed into 3D polygons
  - Intersections of the generated polygons are 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/~tanaka



Debugging Geometries
OLAP

- Stand-alone batch application
  - Provided as extended example; Can be combined with a graphical environment and GUI

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:

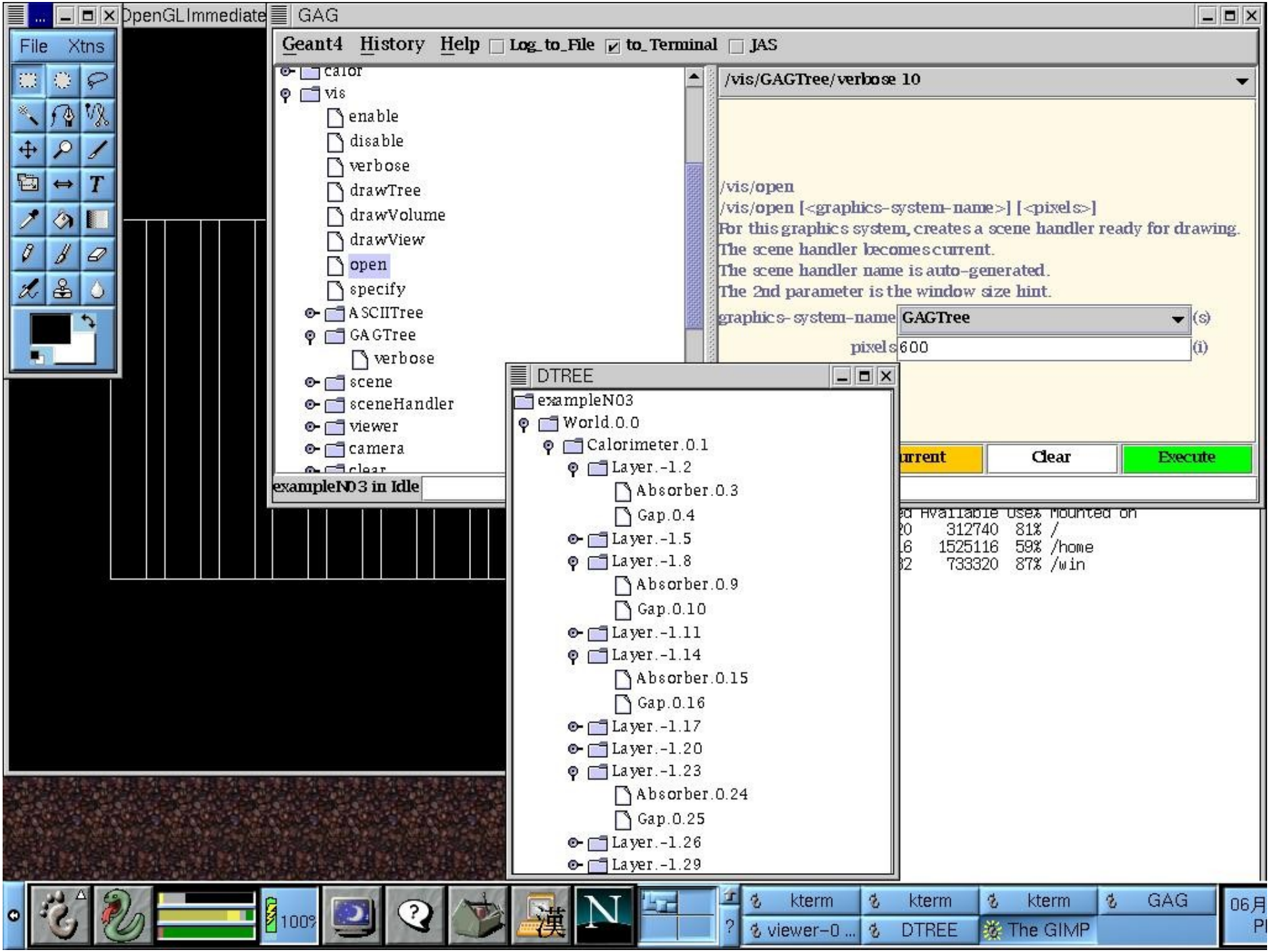
```
delta=59.3416
vol 1: point=(560.513,1503.21,-141.4)
vol 2: point=(560.513,1443.86,-141.4)
A -> B:
[0]:  ins=[2]  PVName=[NewWorld:0] Type=[N] ...
[1]:  ins=[0]  PVName=[ECalEndcap:0] Type=[N] ..
[2]:  ins=[1]  PVName=[ECalEndcapO7:38] Type=[N]

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

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

Debugging Geometries

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
- Provide G3 DTREE capabilities and more


OpenGLImmediate
File Xtns
GAG
Geant4 History Help Log_to_File to_Terminal JAS
color
vis
enable
disable
verbose
drawTree
drawVolume
drawView
open
specify
ASCIITree
GAGTree
verbose
scene
sceneHandler
viewer
camera
/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
Available Use% Mounted on
312740 81% /
1525116 59% /home
733320 87% /win
100%
kterm
kterm
kterm
GAG
viewer-0 ...
DTREE
The GIMP

Contents

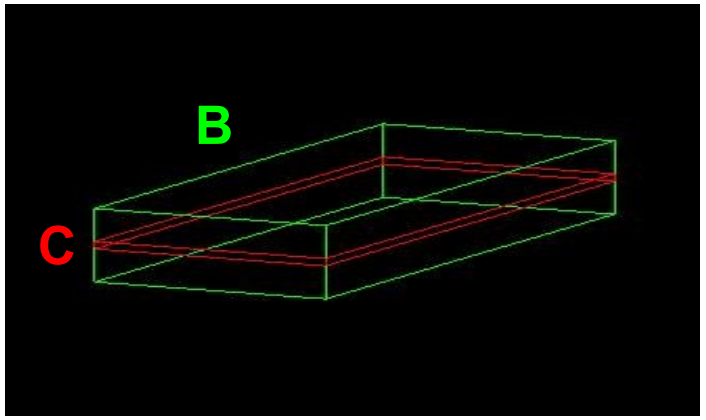
Geometry checking tools

Basics of Touchable

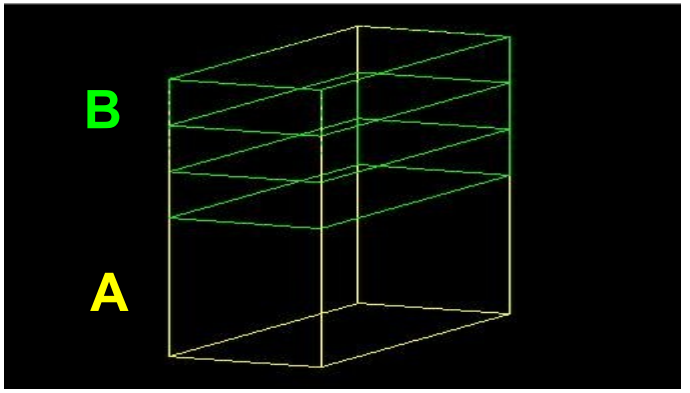
Region

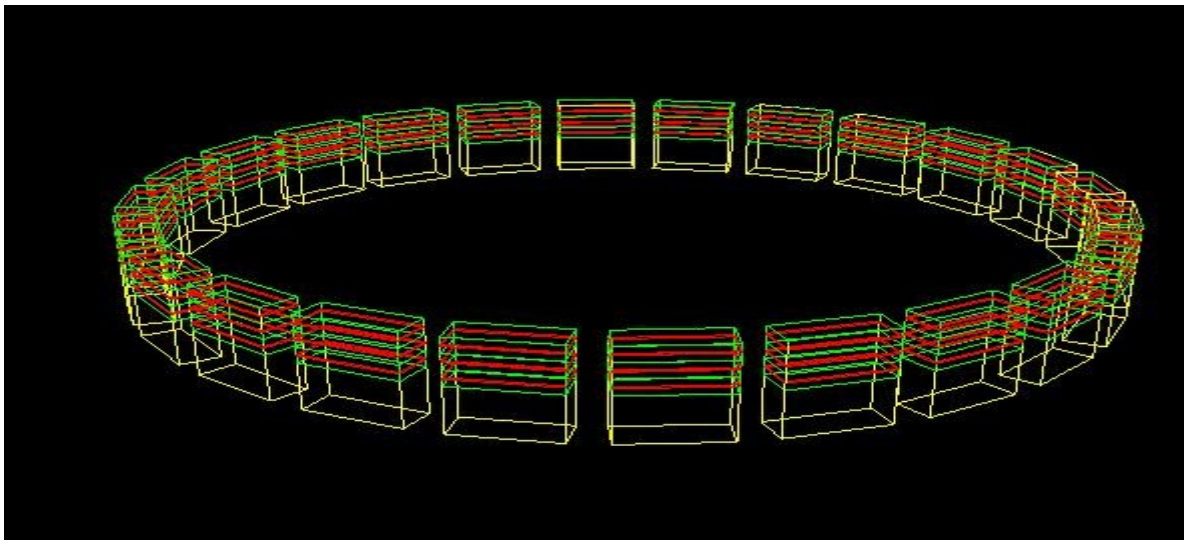
Basics Of Touchables

- 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

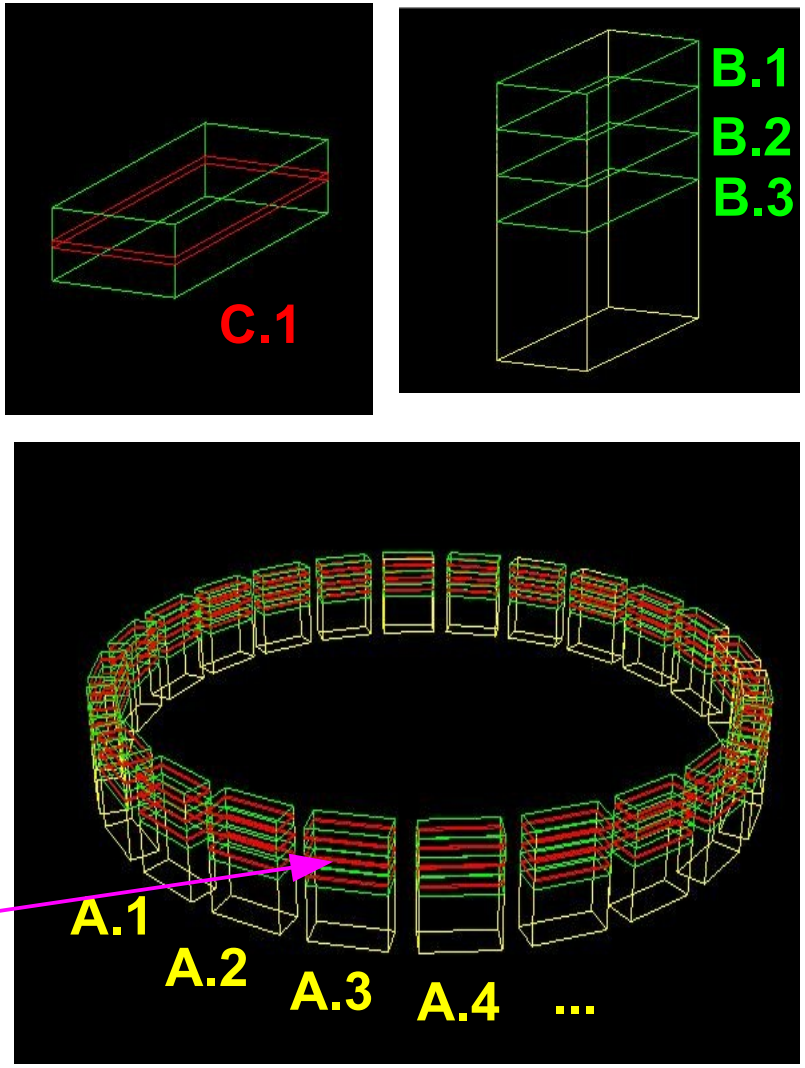




- The volume A has a 24 positions in the world
- While in the 'logical' geometry tree the volume C is represented by just one physical volume, in the real world there are many C 'volumes'
- *How can we then identify these volumes C ?*

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



Basics of Touchables

G4VTouchable

- 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

Basics of Touchables

Accessing touchable in Tracking

- Full geometrical information available via touchable
  - to processes, to user code: sensitive detectors, hits
- During tracking, the current touchable can be accessed via G4Step object

```
MySteppingAction::UserSteppingAction(const G4Step* step)
{
  // Get touchable from step
  const G4VTouchable* touchable
        = step->GetPreStepPoint()->GetTouchable();

  // Get copy numbers
  G4int copyNo_B = touchable->GetCopyNumber(1);
  G4int copyNo_A = touchable->GetCopyNumber(2);
}
```

Use pre-step point rather than post-step point, as when crossing geometry boundary the post-step point belongs already to the next volume

Contents

Geometry checking tools

Basics of Touchable

Region

Region

- A region is a part of the geometry hierarchy, i.e. a set of geometry volumes, typically of a sub-system.
  - A logical volume can be a region.
- A region may hold its own user defined setting affecting tracking
  - It is useful in complex geometry setups, such as those found in large detectors in particle physics experiments, where different sub-systems may require different level of detail simulation

Please note :

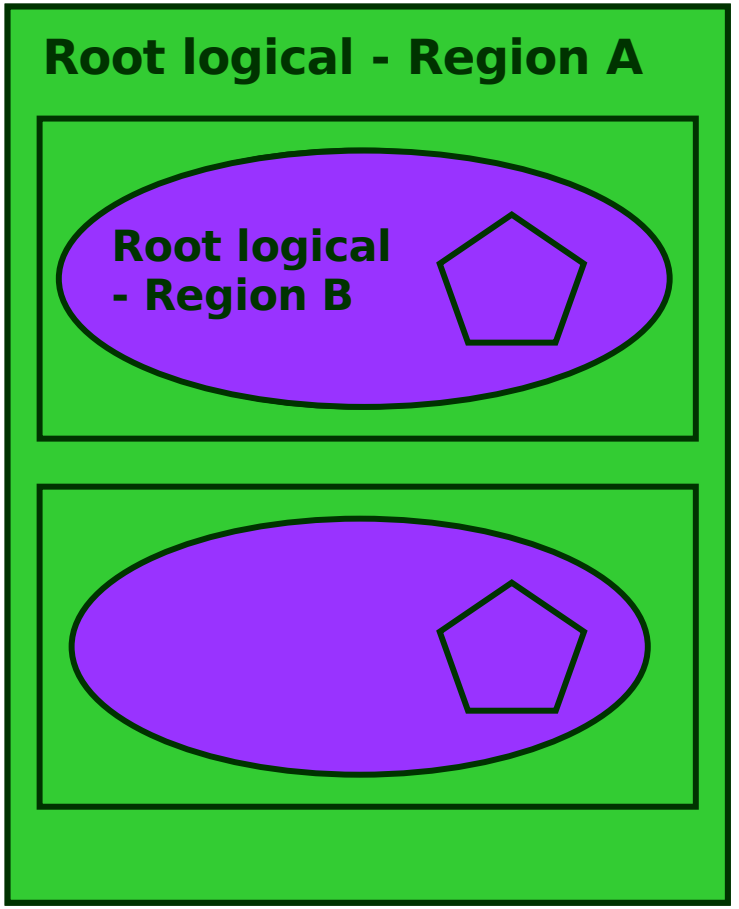
- World logical volume is recognized as the default region. User is not allowed to define a region to the world logical volume.

Region Properties

- A region may have its unique
  - Production thresholds (cuts)
  - User limits
    - Artificial limits affecting to the tracking, e.g. max step length, max number of steps, min kinetic energy left, etc.
    - You can set user limits directly to logical volume as well. If both logical volume and associated region have user limits, those of logical volume wins.
  - User region information
    - E.g. to implement a fast Boolean method to identify the nature of the region.
  - Fast simulation manager
  - Regional G4UserSteppingAction (a new feature in v 9.0)

Root Logical Volume

- A logical volume becomes a root logical volume once a region is assigned to it.
  - All daughter volumes belonging to the root logical volume share the same region, 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
G4Region

- A region is instantiated and defined by

```
G4Region* aRegion = new G4Region("region_name");
aRegion->AddRootLogicalVolume(aLogicalVolume);
```

  - Region propagates down to all geometrical hierarchy until the bottom or another root logical volume.
- The region can be accesed by its name via **G4RegionStore:**

```
G4Region* myRegion
  = G4RegionStore::GetInstance()->GetRegion("region_name");
```

Region
G4Region

- Production thresholds (cuts) can be assigned to a region in a user physics list

```
void MyPhysicsList::SetCuts()
{
  // Default production thresholds for the world volume
  SetCutsWithDefault();

  // Production thresholds for detector regions
  G4ProductionCuts* cuts = new G4ProductionCuts;
  cuts->SetProductionCut(cutValue);
  G4Region* region = G4RegionStore::GetInstance()
      ->GetRegion("myRegion");
  region->SetProductionCuts(cuts);
}
```