Detector Description:

**Geant 4**

Visualization Attributes
Optimisation & Debugging techniques

http://cern.ch/geant4
PART IV

Visualization

- Visualization attributes
- GGE & geometry tree
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 & 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
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 is provided as a separate tool in Geant4
  - As part of the MOMO Java environment suite
    - geant4/environments/MOMO/MOMO.jar
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
PART IV

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

<table>
<thead>
<tr>
<th>Percent</th>
<th>Memory</th>
<th>Heads</th>
<th>Nodes</th>
<th>Pointers</th>
<th>Total CPU</th>
<th>Volume</th>
</tr>
</thead>
<tbody>
<tr>
<td>91.70</td>
<td>1k</td>
<td>1</td>
<td>50</td>
<td>50</td>
<td>0.00</td>
<td>Calorimeter</td>
</tr>
<tr>
<td>8.30</td>
<td>0k</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>0.00</td>
<td>Layer</td>
</tr>
</tbody>
</table>
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 \texttt{G4LogicalVolume} or through provided set methods
    - \texttt{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 \texttt{G4PVParameterised}
  - Using 3D voxelisation along the 3 Cartesian axes
    - Specifying in \texttt{kUndefined} in the constructor for \texttt{G4PVParameterised}
PART IV

Debugging geometries

*Debugging tools*
- Optional checks at Construction
- DAVID
- Run-time commands
- OLAP
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
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 require lots of **CPU time**
- Depending on the complexity of geometry
- Can also be forced on a specific physical volume through the method:

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

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

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)^

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>----- start position (cm) -----</th>
<th>----- end position (cm) -----</th>
</tr>
</thead>
<tbody>
<tr>
<td>240</td>
<td>-240</td>
<td>-145.5</td>
</tr>
</tbody>
</table>

Which in the mother coordinate system are:

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>----- start position (cm) -----</th>
<th>----- end position (cm) -----</th>
</tr>
</thead>
<tbody>
<tr>
<td>. .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which in the coordinate system of Tracker[0] are:

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>----- start position (cm) -----</th>
<th>----- end position (cm) -----</th>
</tr>
</thead>
<tbody>
<tr>
<td>. .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Which in the coordinate system of Overlap[0] are:

<table>
<thead>
<tr>
<th>length (cm)</th>
<th>----- start position (cm) -----</th>
<th>----- end position (cm) -----</th>
</tr>
</thead>
<tbody>
<tr>
<td>. .</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

Detector Description: Visualization, optimisation & debugging - Geant4 Course

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=[ECALEndcap07:38] Type=[N]
B -> A:
[0]: ins=[2] PVName=[NewWorld:0] Type=[N] ...
```

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

graphical indication of detected overlaps

red: mother
blue: daughters

daughters are protruding their mother