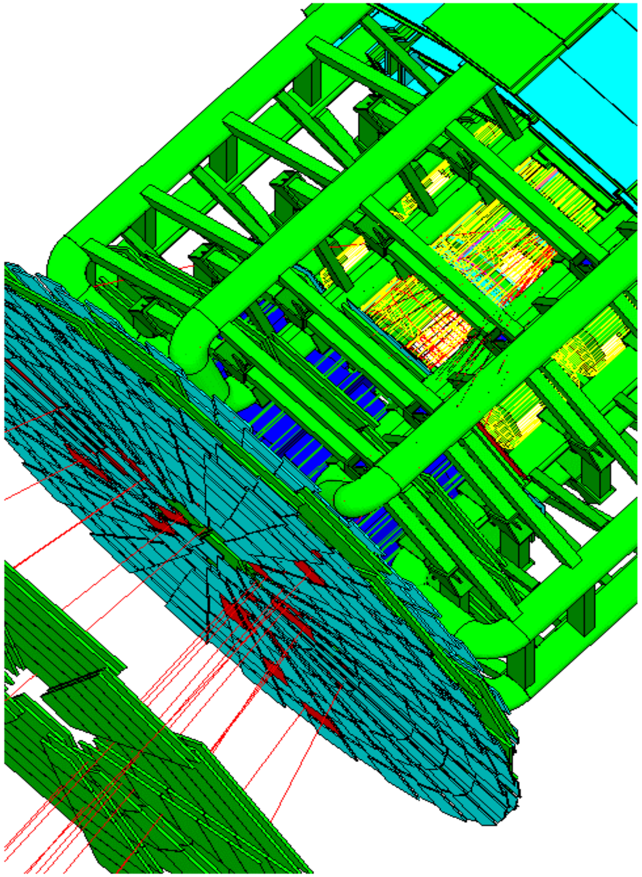
Geant 4

Detector Description – advanced features

http://cern.ch/geant4

The full set of lecture notes of this Geant4 Course is available at
http://www.ge.infn.it/geant4/events/nss2003/geant4course.html



PART IV

Detector Description:

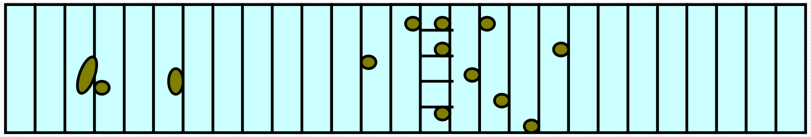
Optimisation technique & Advanced features

Detector Description
Advanced features

- *The optimisation technique*
- *Grouping volumes*
- *Reflections of volumes and hierarchies*
- *User defined solids*
- *Debugging tools*

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
```

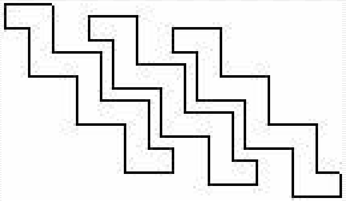
Visualising voxel structure

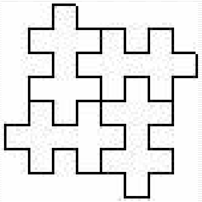
- The computed voxel structure can be visualized with the final detector geometry
  - Helper class **`G4DrawVoxels`**
  - Visualize voxels given a logical volume
    - `G4DrawVoxels::DrawVoxels(const G4LogicalVolume*)`
  - Allows setting of visualization attributes for voxels
    - `G4DrawVoxels::SetVoxelsVisAttributes(…)`
  - 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`

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

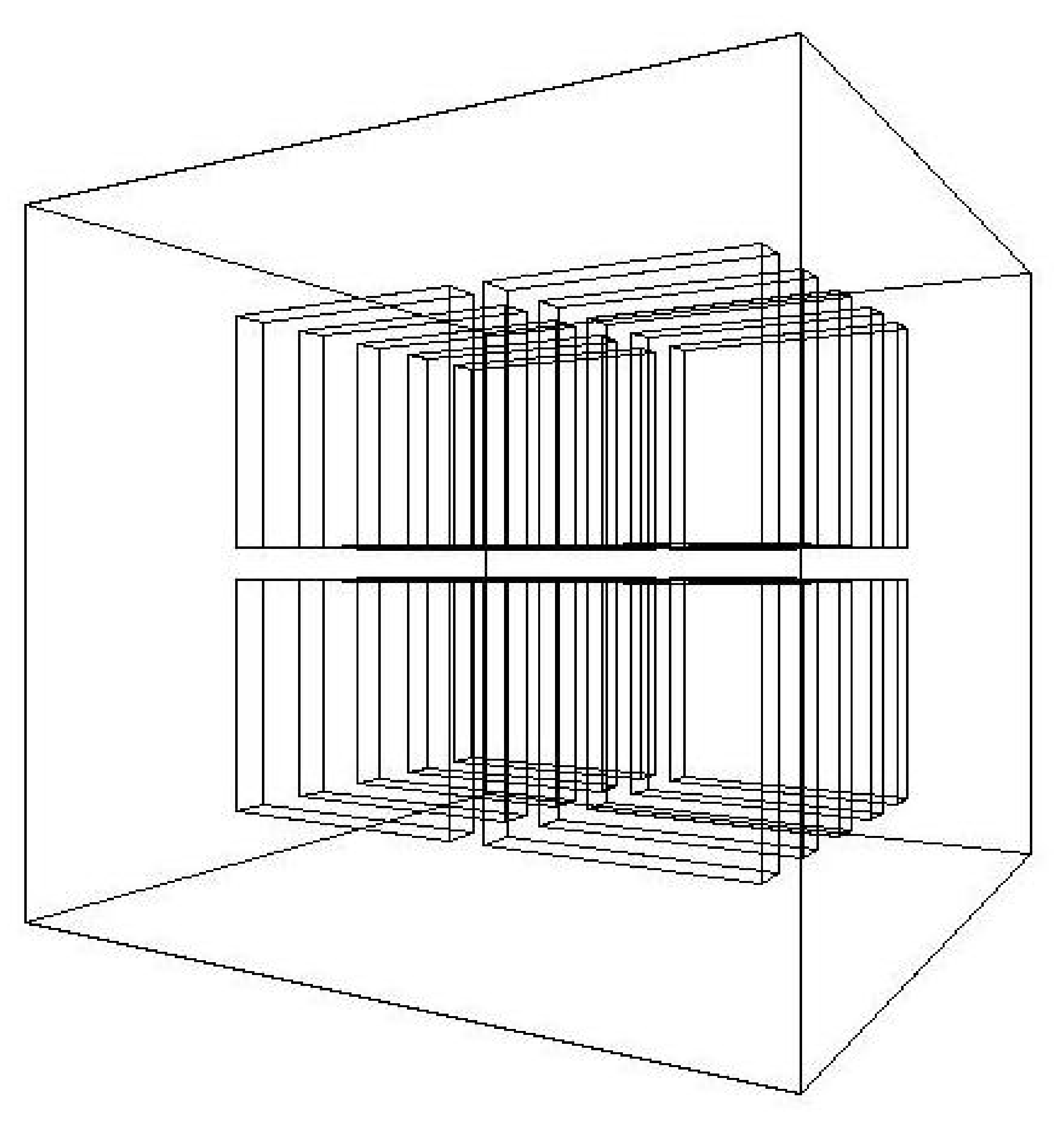
```
G4AssemblyVolume( G4LogicalVolume* volume,
                  G4ThreeVector& translation,
                  G4RotationMatrix* rotation);
```

- 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

- `G4ReflectedSolid`
  - utility class representing a solid shifted from its original reference frame to a new *reflected* 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:
  1. Performs the transformation decomposition
  2. Generates a new reflected solid and logical volume
     - Retrieves it from a map if the reflected object is already created
  3. Transforms any daughter and places them in the given mother
  4. 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
```

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;
```

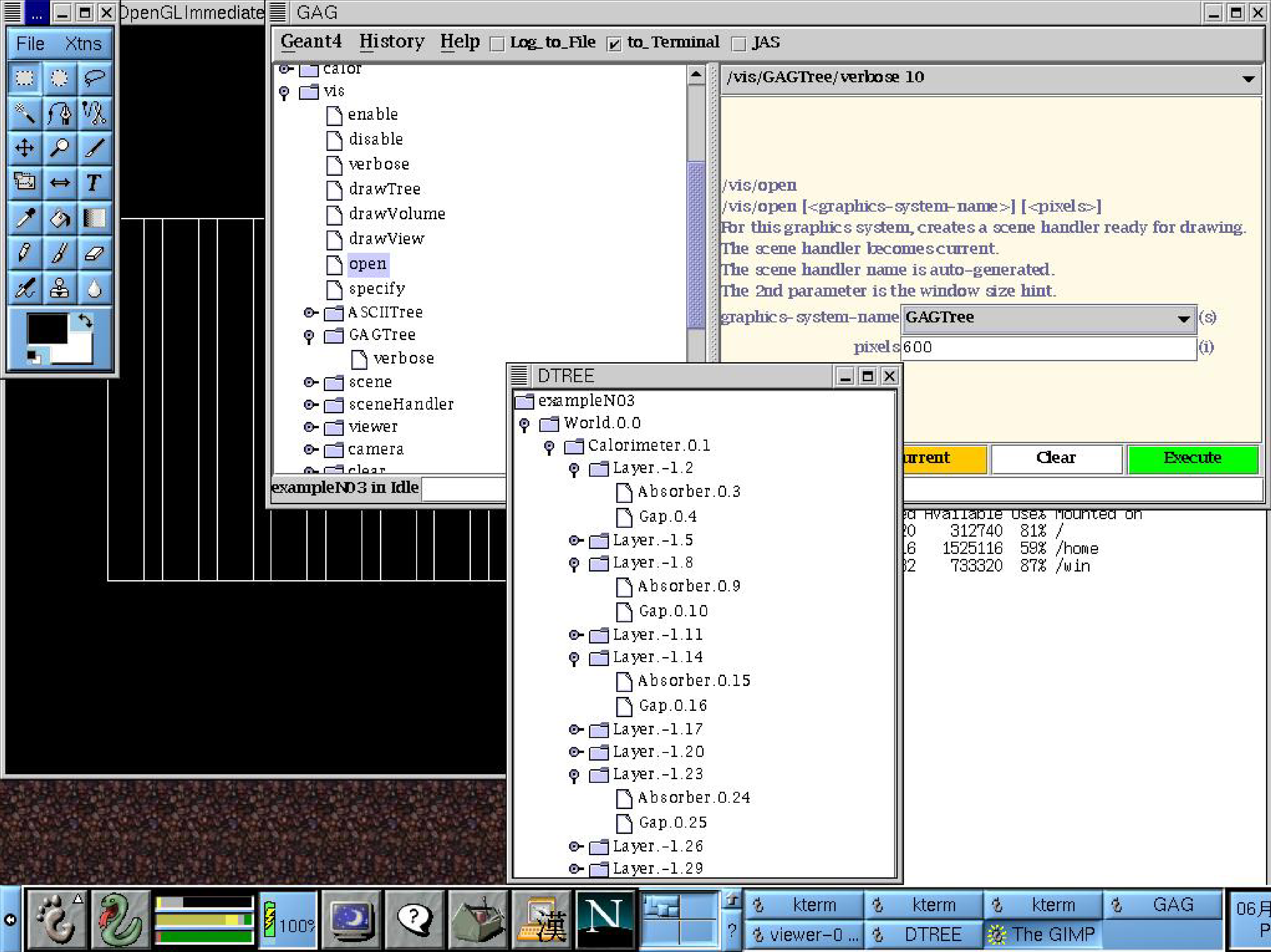
- *Provides extent (bounding box) as possible hint to the graphics view*

GGE (Graphical Geometry Editor)

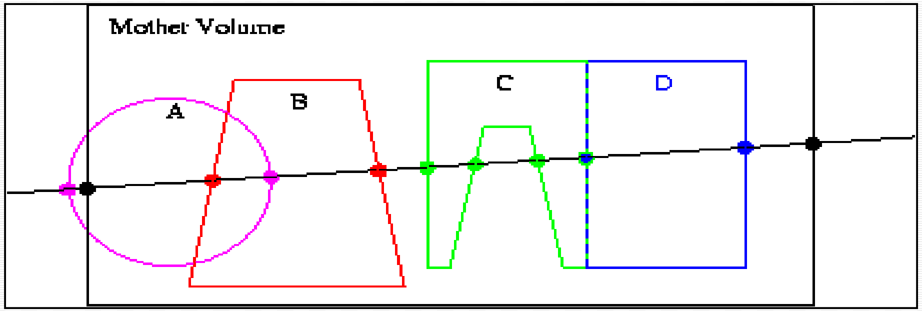
- Implemented in JAVA, GGE is a graphical geometry editor compliant to Geant4. It allows to:
  - Describe a detector geometry including:
    - materials, solids, logical volumes, placements
  - Graphically visualize the detector geometry using a Geant4 supported visualization system, e.g. DAWN
  - Store persistently the detector description
  - Generate the C++ code according to the Geant4 specifications
- GGE can be downloaded from Web as a separate tool:
  - `http://erpc1.naruto-u.ac.jp/~geant4/`

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


OpenGLImmediate
File Xtns
GAG
Geant4 History Help Log_to_File to_Terminal JAS
calor
vis
enable
disable
verbose
drawTree
drawVolume
drawView
open
specify
ASCIITree
GAGTree
verbose
scene
sceneHandler
viewer
camera
exampleN03 in Idle
/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
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

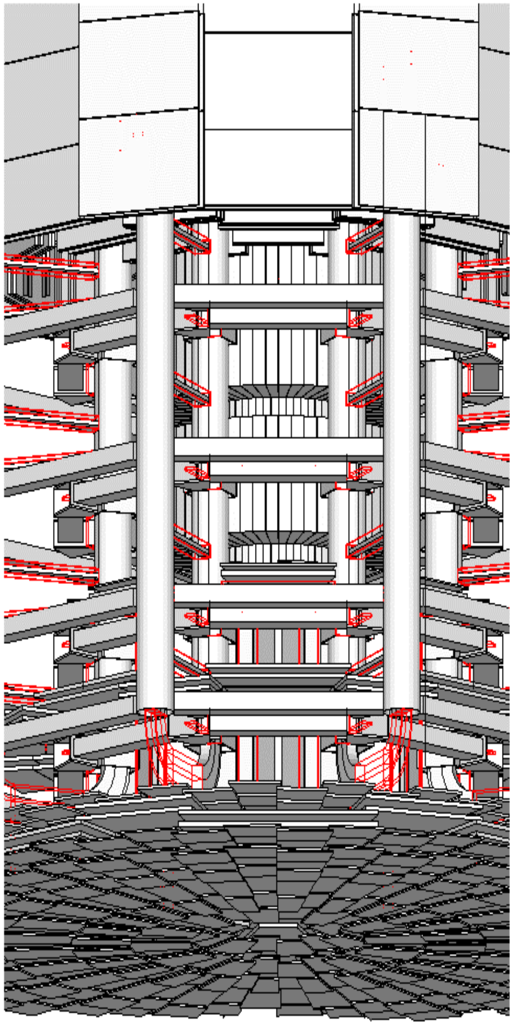
Debugging geometries



- 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
- 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: 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:

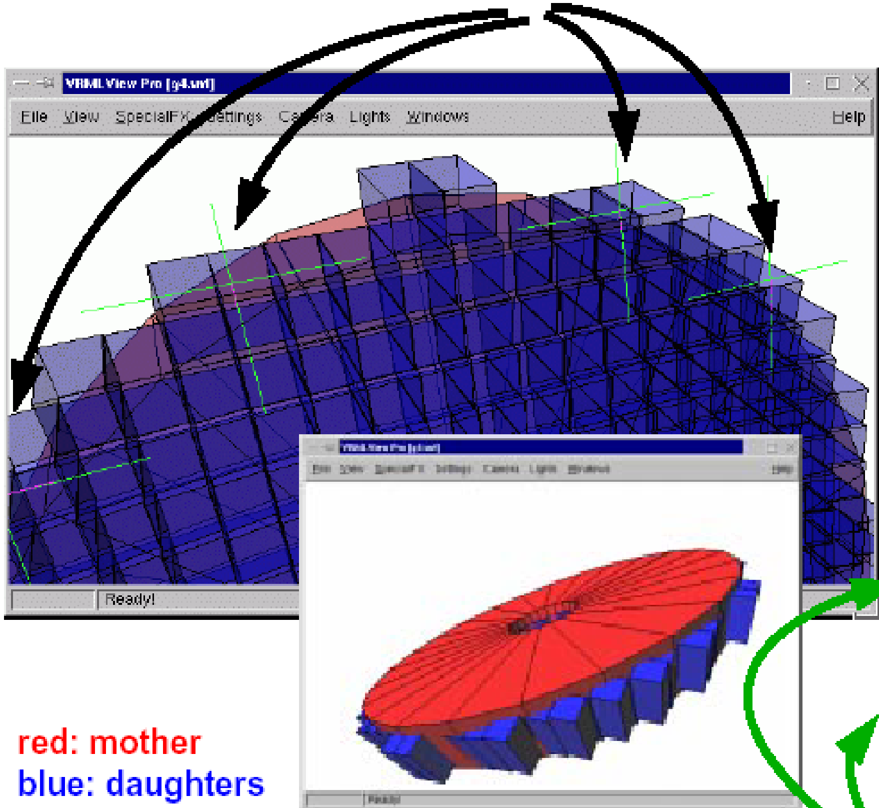
```
GeomTest: no daughter volume extending outside mother detected.
GeomTest Error: Overlapping daughter volumes
    The volumes Tracker[0] and Overlap[0],
    both daughters of volume World[0],
    appear to overlap at the following points in global coordinates: (list truncated)
  length (cm)    -----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

- Uses 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=[ECalEndcap07:38] Type=[N]

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

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