

Geant 4

Visualisation, (G)UI and Analysis

<http://cern.ch/geant4>

The full set of lecture notes of this Geant4 Course is available at
<http://geant4.web.cern.ch/geant4/meetings/school2005/>

Contents (1)

■ Part 1: How to perform visualisation

- Introduction
- Visualisable Objects
- Visualisation Attributes
- Polyline and Marker
- Visualisation Drivers
- main() Function
- Visualisation Commands
- How to Visualise from C++ Codes
- Exercises
- Information

Contents (2)

■ Part 2: Geant4 GUI

- Select (G)UI
- Environmental variables
- Useful GUI Tools Released by Geant4 Developers

Contents (3)

■ Part 3: DAVID and DTREE

- Graphically detecting overlaps in geometry
- Display of the detector geometry tree

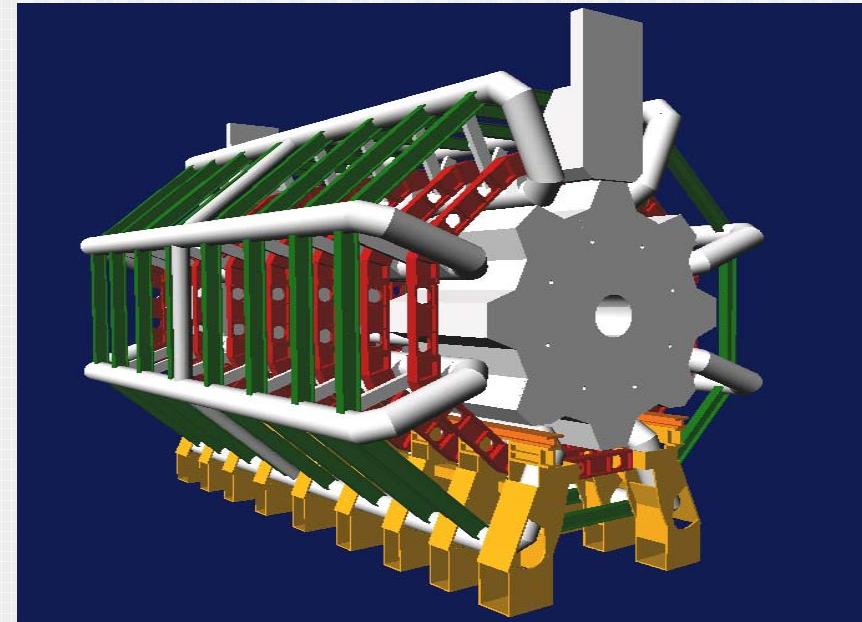
Contents (4)

■ **Part 4: AIDA and binding to analysis**

- AIDA abstract interfaces
- Geant4 setup and analysis tools

PART 1

Geant4 Visualisation



1. Introduction

- Geant4 Visualisation must respond to varieties of user requirements
 - Quick response to survey successive events
 - Impressive special effects for demonstration
 - High-quality output to prepare journal papers
 - Flexible camera control for debugging geometry
 - Highlighting overlapping of physical volumes
 - Interactive picking of visualised objects
 - ...

2. Visualisable Objects (1)

- Simulation data can be visualised such as:
 - Detector components
 - A hierarchical structure of physical volumes
 - A piece of physical volume, logical volume, and solid
 - Particle trajectories and tracking steps
 - Hits of particles in detector components
- Visualisation is performed either with commands or by writing C++ source codes of user-action classes

2. Visualisable Objects (2)

- You can also visualise other user defined objects such as:
 - A polyline, that is, a set of successive line segments (example: coordinate axes)
 - A marker which marks an arbitrary 3D position (example: eye guides)
 - Texts
 - character strings for description
 - comments or titles ...

2. Visualisable Objects (3)

- A typical application is the DrawTrajectory and DrawHit methods
- The default G4VTrajectory::
DrawTrajectory(G4int imode) draws positive/negative/neutral particle trajectories blue/red/green with yellow markers.
- The user must write DrawHit (see later)

3. Visualisation Attributes

- Control the appearance of visualisable objects
 - Colour, visibility, forced-wireframe style, etc
 - A set of visualisation attributes is held by the class G4VisAttributes
- A G4VisAttributes object is assigned to a visualisable object with its method
`SetVisAttributes()` :

- `experimentalHall_logical`
 - > `SetVisAttributes(G4VisAttributes::Invisible)`

3.1 Constructors of G4VisAttributes

The following constructors are supported by class G4VisAttributes:

- G4VisAttributes G4VisAttributes ()
- G4VisAttributes (G4bool visibility)
- G4VisAttributes (const G4Colour& colour)
- G4VisAttributes (G4bool visibility,
 const G4Colour& colour)

or use Set methods...

3.2 Visibility

- A boolean flag (`G4bool`) to control the visibility of objects
- Access function
 - `G4VisAttributes::SetVisibility`
`(G4bool visibility)`
 - If `false` is given as argument, objects are culled (not shown) if culling is on.
 - The default value of visibility is `true`.
 - The default is culling of invisible objects.

3.3 Colour (1)

- Class **G4VisAttributes** holds its colour entry as an instance of class **G4Colour**
 - An equivalent class name, **G4Color**, is also available
- **G4Colour** is instantiated by giving RGB components and opacity(alpha):
 - `G4Colour::G4Colour(G4double red = 1.0,
G4double green = 1.0, G4double blue = 1.0,
G4double alpha = 1.0)`
 - The default arguments define a white object that will be opaque when drawn in solid mode (opacity does not affect wireframe mode)

3.3 Colour (2)

- Access functions of
G4VisAttributes to set **G4Colour**

- SetColour(const G4Colour& colour)
- SetColour(G4double red,
 G4double green,
 G4double blue,
 G4double alpha = 1.)
- (or SetColor)

3.4 Assigning `G4VisAttributes` to a logical volume

- Class `G4LogicalVolume` holds a pointer of `G4VisAttributes`
- Access functions of `G4LogicalVolume`
 - `SetVisAttributes (const G4VisAttributes* pva)`
- Remember, `G4VisAttributes` are held by a pointer; it must have a life as least as long as the holder...

Sample C++ Code:

----- C++ source codes: Assigning G4VisAttributes to a logical volume

```
// Instantiation of a logical volume
myTargetLog =
    new G4LogicalVolume( myTargetTube,BGO,
                         "TLog", 0, 0, 0);

// Instantiation of a set of visualization
// attributes with cyan colour
G4VisAttributes * calTubeVisAtt =
    new G4VisAttributes(G4Colour(0.,1.,1.));
// Set the forced wireframe style
calTubeVisAtt->SetForceWireframe(true);
// Assignment of the visualization attributes
// to the logical volume
myTargetLog->SetVisAttributes(calTubeVisAtt);
```

4. Polyline and Marker

- Polyline and marker are defined in the `graphics_reps` category
- They are available to model 3D scenes for visualisation

4.1 Polyline

- A set of successive line segments
- Defined with a class `G4Polyline`
- Used to visualise tracking steps, particle trajectories, coordinate axes, etc (but note availability of commands: `/vis/scene/add/trajectories` etc.)
- `G4Polyline` is defined as a list of `G4Point3D` objects.

Sample C++ Code:

```
//-- C++ source code: An example of defining a line segment

// Instantiation
G4Polyline x_axis;

// Vertex positions
x_axis.append ( G4Point3D ( 0., 0., 0.) );
x_axis.append ( G4Point3D ( 5. * cm, 0., 0.) );

// Color
G4Colour red ( 1.0, 0.0, 0.0 ); // color for x-axis
G4VisAttributes att ( red );
x_axis.SetVisAttributes( att );

//-- end of C++ source code
```

4.2 Marker (1)

- Set a mark to an arbitrary 3D position
- Usually used to visualise hits of particles
- Designed as a 2-dimensional primitive with shape (square, circle, etc), color, and special properties of
 - (a) always facing the camera and
 - (b) having the possibility of its size (diameter) defined either in real 3D or 2D screen units (pixels)

4.2 Marker (2)

■ Kinds of markers

- Square : **G4Square**
- Circle : **G4Circle**
- Text : **G4Text**

■ Constructors

- **G4Circle (const G4Point3D& pos)**
- **G4Square (const G4Point3D& pos)**
- **G4Text (const G4String& text,
const G4Point3D& pos)**

4.2 Marker (3)

- Each marker class inherits class **G4VMarker**
- All access functions of **G4VMarker** are available. For example,
 - `SetPosition(const G4Point3D&)`
 - `SetWorldSize(G4double real_3d_size)`
 - `SetScreenSize(G4double 2d_size_pixel)`
 - `SetScreenRadius(G4double r)`
 - etc.

Sample C++ Code:

Definition of a small red circle as a marker :

```
G4Circle circle(position);
// Instantiate a circle with its 3D position. The
// argument "position" is defined as G4Point3D instance
circle.SetScreenDiameter (1.0);
circle.SetFillStyle (G4Circle::filled);
// Make it a filled circle
G4Colour colour(1.,0.,0.);           // Define red color
G4VisAttributes attrs(colour);
// Define a red visualization attribute
circle.SetVisAttributes(attrs);
// Assign the red attribute to the circle
//-- end of C++ source code
```

4.3 Draw methods of G4VVisManager

- G4VVisManager is the user interface to the concrete vis manager. It resides in the intercoms category.
- Always protect your code by obtaining a pointer to the concrete instance through G4VVisManager ::GetConcreteInstance (it also ensures a valid view is available)

4.3 Draw methods of G4VVisManager (2)

- The Draw methods are available for:
 - G4Circle
 - G4NURBS
 - G4Polyhedron
 - G4Polyline
 - G4Polymarker
 - G4Scale
 - G4Square
 - G4Text
 - G4LogicalVolume
 - G4PhysicalVolume
 - G4Solid
 - Also G4VHit and G4VTrajectory for convenience, but use commands, such as /vis/scene/add/trajectories.

A typical DrawHit method

```
void ExN02TrackerHit::Draw()
{
    G4VVisManager* pVVisManager = G4VVisManager::GetConcreteInstance();
    if(pVVisManager)
    {
        G4Circle circle(pos);
        circle.SetScreenSize(2.);
        circle.SetFillStyle(G4Circle::filled);
        G4Colour colour(1.,0.,0.);
        G4VisAttributes attrs(colour);
        circle.SetVisAttributes(attrs);
        pVVisManager->Draw(circle);
    }
}
```

5. Visualisation Drivers

- Visualisation drivers are interfaces to 3D graphics software
- You can select your favorite one(s) depending on your purposes such as
 - Seeing detector and events
 - Preparing precise figures for journal papers
 - Publication of results on Web
 - Debugging geometry
 - Demonstration
 - Etc

5.1 Available Graphics Software

- Geant4 provides visualisation drivers:
 - DAWN : Technical High-quality PostScript output
 - OpenGL: Quick and flexible visualisation
 - OpenInventor: Interactivity, virtual reality, etc
 - RayTracer : Photo-realistic rendering
 - VRML: Interactivity, 3D graphics on Web
 - HepRep: connection to variety of viewers

5.2 Available Visualisation Drivers

- DAWN(FILE) → Fukui Renderer DAWN
- OPENGLX(m) → OpenGL with Xlib (Xm)
- HepRep → HepRep viewers
- OIX → OpenInventor with Xlib
- RayTracer → JPEG files
- VRML → VRML 1.0/2.0
- A/GAGTree → geometry hierarchy
- Etc (see “help /vis/open”)

5.3 How to Use Visualisation Drivers

- Users can *install* visualisation driver(s) by setting G4VIS_BUILD... environment variables before building Geant4 libraries (use Configure)
- Similarly, G4VIS_USE... environment variables before compiling user code.
- For example, Configure produces:
 - setenv G4VIS_USE_DAWN 1
 - setenv G4VIS_USE_OPENGLX 1
 - setenv G4VIS_USE_VRML 1
- Note: force a rebuild/recompile if you change environment

6. main() Function (1)

- Instantiate a visualisation manager, e.g., **G4VisExecutive** (but you may write your own).
- Near the start of your program, e.g., in **main()** :
 - Include the header file of the chosen visualisation manager
 - Instantiate and initialise the visualisation manager. The **Initialise()** method instantiates the chosen visualisation driver “factories”
 - Delete the visualisation manager at the end
 - You can use the C macro “**G4VIS_USE**”, which is automatically set if you incorporate a visualisation driver in compilation

6. main() Function (2)

- From a typical `main()` function :

```
// Include the header file of your vis manager
#ifndef G4VIS_USE
    #include "G4VisExecutive.hh"
#endif

...
// Instantiate and initialize the vis manager
#ifndef G4VIS_USE
    G4VisManager* visManager = new G4VisExecutive;
    visManager->Initialize();
#endif

...
// Delete the visualisation manager
#ifndef G4VIS_USE
    delete visManager;
#endif
```

7. Visualisation commands

- Here, we introduce some frequently-used built-in visualisation commands
- See command web page
<http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/Control/commands.html>
- For simplicity, we assume that the Geant4 executable is compiled, incorporating DAWN, OPENGLX, and VRMLFILE drivers
 - `setenv G4VIS_USE_DAWN 1`
 - `setenv G4VIS_USE_OPENGLX 1`
 - `setenv G4VIS_USE_VRMLFILE 1`

7.1 Scene, Scene Handler, Viewer

- In order to use visualisation commands, you need to understand the concepts of “scene”, “scene handler”, and “viewer”.
- **Scene:** *A set of visualisable 3D objects*
- **Scene handler:** *Interprets the scene for a particular graphics system*
- **Viewer:** *Image generator*
- Each scene handler is assigned to a scene
- Each viewer is assigned to a scene handler
- “visualisation driver”
= “scene handler” + “viewer”

7.1(2) Scene Objects

- Physical volume, which is assumed to include descendants to unlimited or specified depth
- Logical volume, showing daughters, Boolean components, voxels and readout geometry
- Ghost volume
- Trajectories, drawn at end of event
- Hits, drawn at end of event
- Axes
- Logo
- Scale
- Text
- User action

7.1(3) Standard View

- The scene computes its “bounding sphere”
- The viewer computes the “standard view” that shows the whole scene
- Pan, dolly and zoom from there
- `/vis/viewer/reset` restores the standard view

7.1(4) Culling Policy

- There exists a primitive culling (elimination) mechanism
- `/vis/viewer/set/culling`
 - global (T), enables all other culling options
 - invisible (T)
 - covered daughters (F)
 - density (F) (lower than given density)
- Defaults in brackets (T = true, F = false)

7.2 Steps of Visualisation

- Step 1: Create a scene handler and a viewer
- Step 2: Create an empty scene
- Step 3: Add objects to the created scene
- Step 4: Attach the current scene handler
to the current scene
- Step 5: Set camera parameters, drawing
style (wireframe/surface), etc
- Step 6: “Execute” the visualisation
- Step 7: Declare the end of visualisation

7.3 An Example of Visualising Detector

```
# Invoke the OGLIX driver:  
# Create a scene handler and a viewer.  
    /vis/open OGLIX  
# Set camera and drawing style  
    /vis/viewer/reset  
    /vis/viewer/viewpointThetaPhi 70 20  
    /vis/viewer/set/style      wireframe  
# Visualize of the whole detector geometry  
# The "/vis/drawVolume" creates a scene, adds the  
# world volume to it, and makes viewer render  
# visualisation.  
    /vis/drawVolume  
# Declare the end of visualisation  
    /vis/viewer/update
```

7.4 An Example of Visualizing Events

```
# Store particle trajactories for visualisation  
    /tracking/storeTrajectory  
# Invoke the DAWN driver: Create a scene  
# handler and a viewer.  
    /vis/open DAWN  
# Camera setting, and drawing style selection,  
# if necessary ...  
  
# Create a new empty scene  
    /vis/scene/create  
# Add the world volume and trajectories to the  
# current scene  
    /vis/scene/add/volume  
    /vis/scene/add/trajectories  
# Let the viewer visualise the scene, and declare  
# the end of visualisation  
    /run/beamOn 10
```

7.5 /vis/open Command

- Command
 - Idle> /vis/open <driver_tag_name>
 - The “driver_tag_name” is a name which shows “driver name” + “mode”
- Action: Create a visualisation driver
 - In other words, create a scene hander and a viewer
- Example: Creating the OPENGLX driver in the immediate mode:
 - Idle> /vis/open OGLIX
- How to list available driver_tag_name
 - Idle> help /vis/open
or
Idle> help /vis/sceneHandler/create

7.6 /vis/viewer/... commands

■ Commands

■ Viewpoint setting:

```
Idle> /vis/viewer/viewpointThetaPhi  
      <theta_deg> <phi_deg>
```

■ Zooming

```
Idle> /vis/viewer/zoom <scale_factor>
```

■ Initialization of camera parameters:

```
Idle> /vis/viewer/reset
```

7.7 /vis/viewer/set/... commands

- `/vis/viewer/set/style <style_name>`
 - The “`style_name`” can be “wireframe” or “surface”
- `/vis/viewer/set/all <viewer-name>`
 - The viewer name is the short name, e.g., `viewer-0`. (Use `/vis/viewer/list` to see.)
 - This copies all the view parameters from named viewer to current viewer.
 - E.g: Find a good view with OpenGL, render to high quality with DAWN.

7.8 /vis/drawVolume and /vis/viewer/update Commands

■ Commands:

- Idle> /vis/drawVolume <physical-volume-name>
(Default: world)
Idle> /vis/viewer/update
- Note that /vis/viewer/update should be executed to declare end of visualisation.
- You can add visualisation commands of, say, coordinate axes between the two commands. For example,
 - Idle> /vis/drawVolume
Idle> /vis/scene/add/axes <Ox> <Oy> <Oz>
 <length> <unit>
 - Idle> /vis/viewer/update

7.9 Commands to Visualize Events

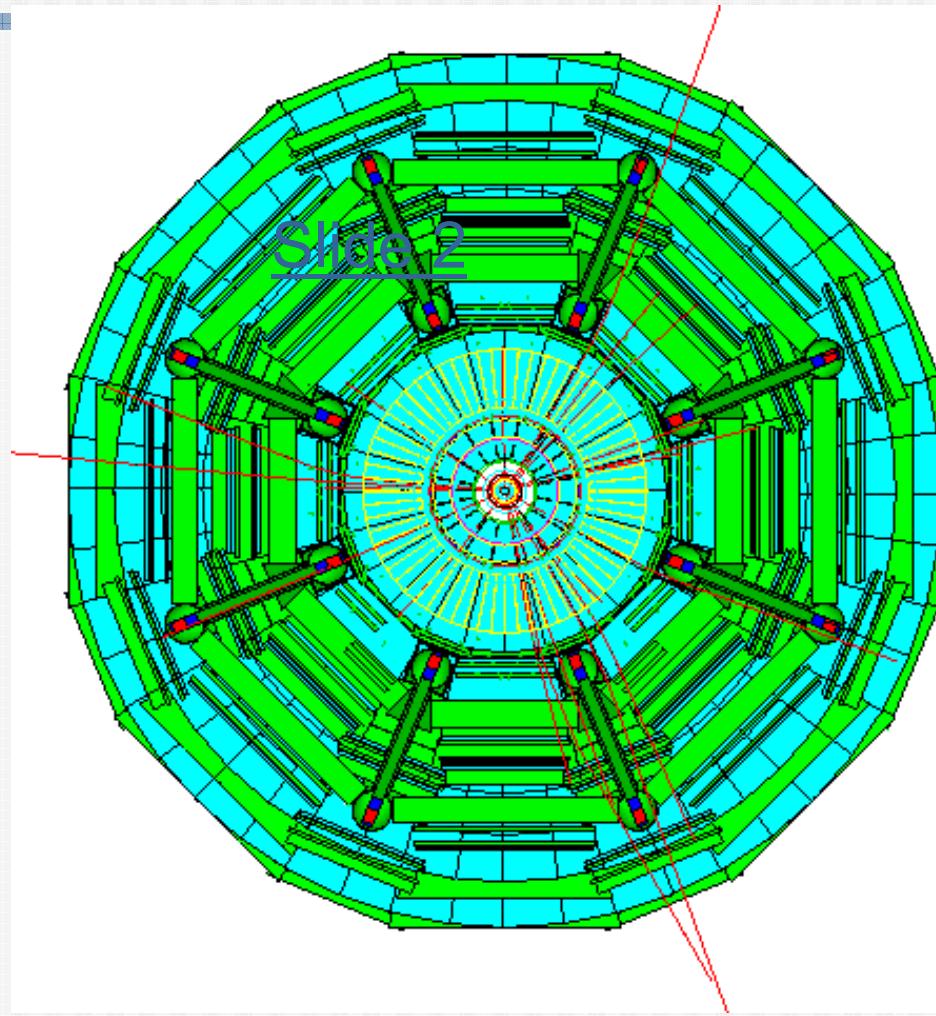
■ Commands

- Idle> /tracking/storeTrajectory 1
- Idle> /vis/scene/add/trajectories
- Idle> /run/beamOn <number_of_events>

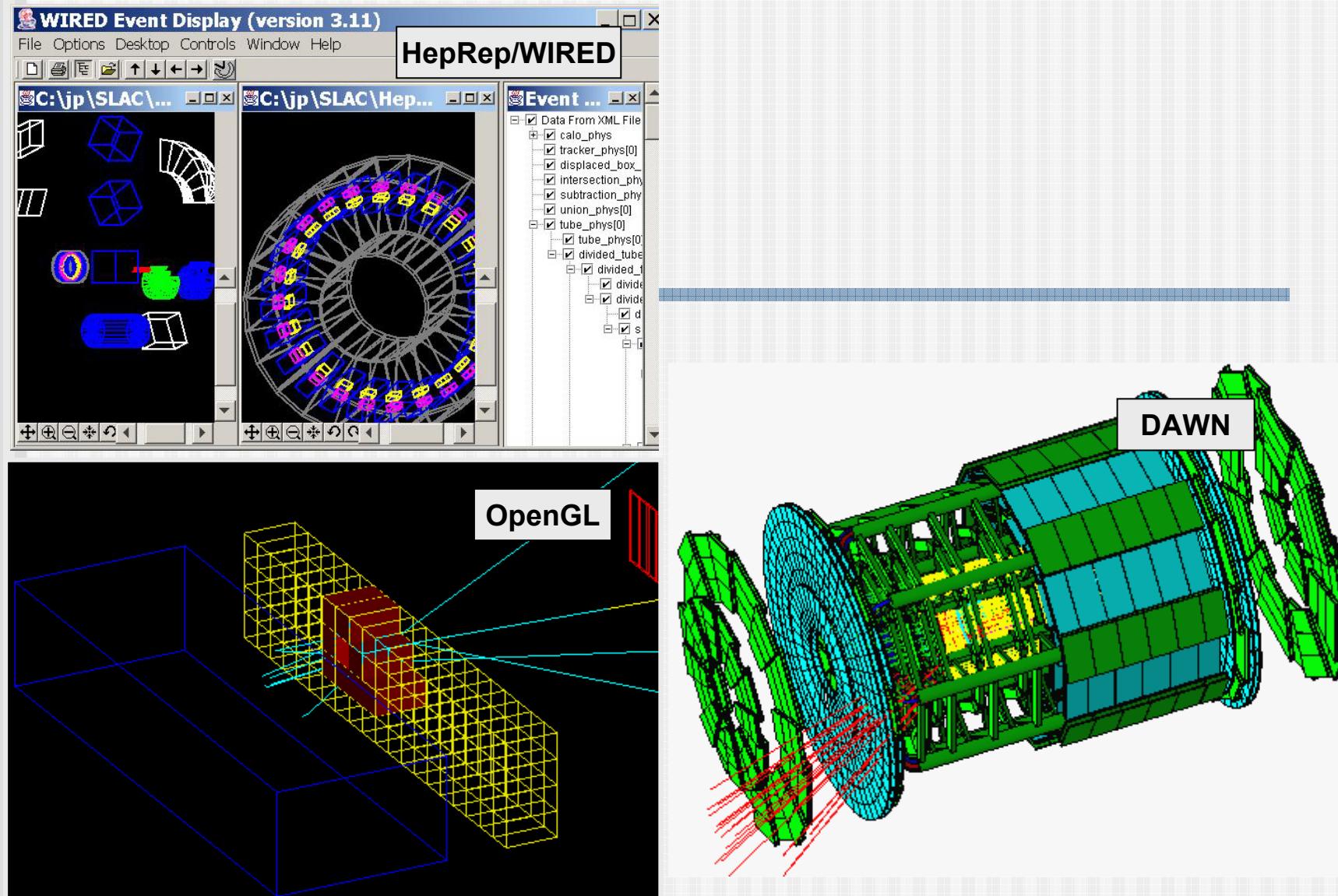
■ Action:

- Automatic visualisation of events

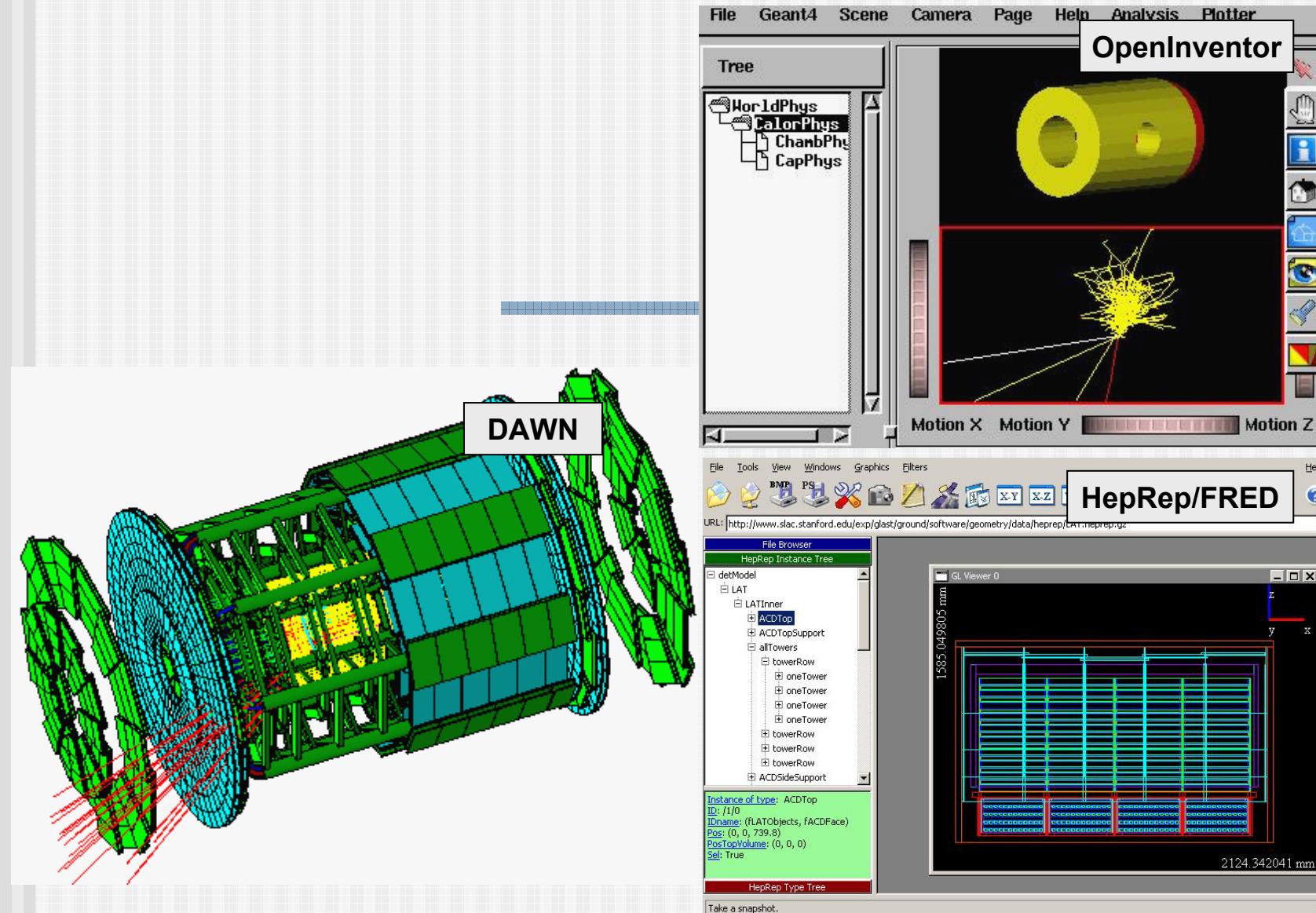
Sample Visualisation (1)

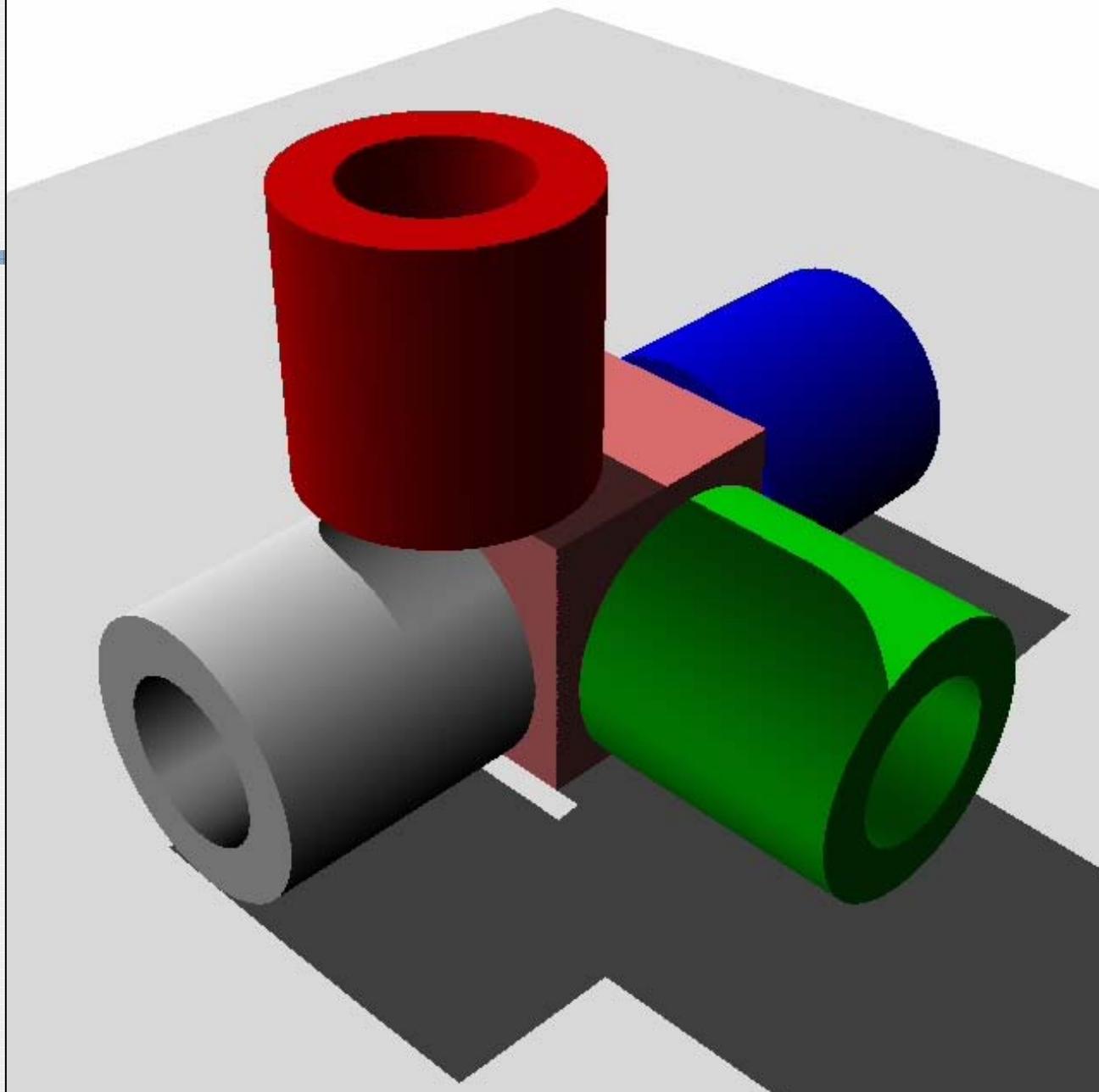


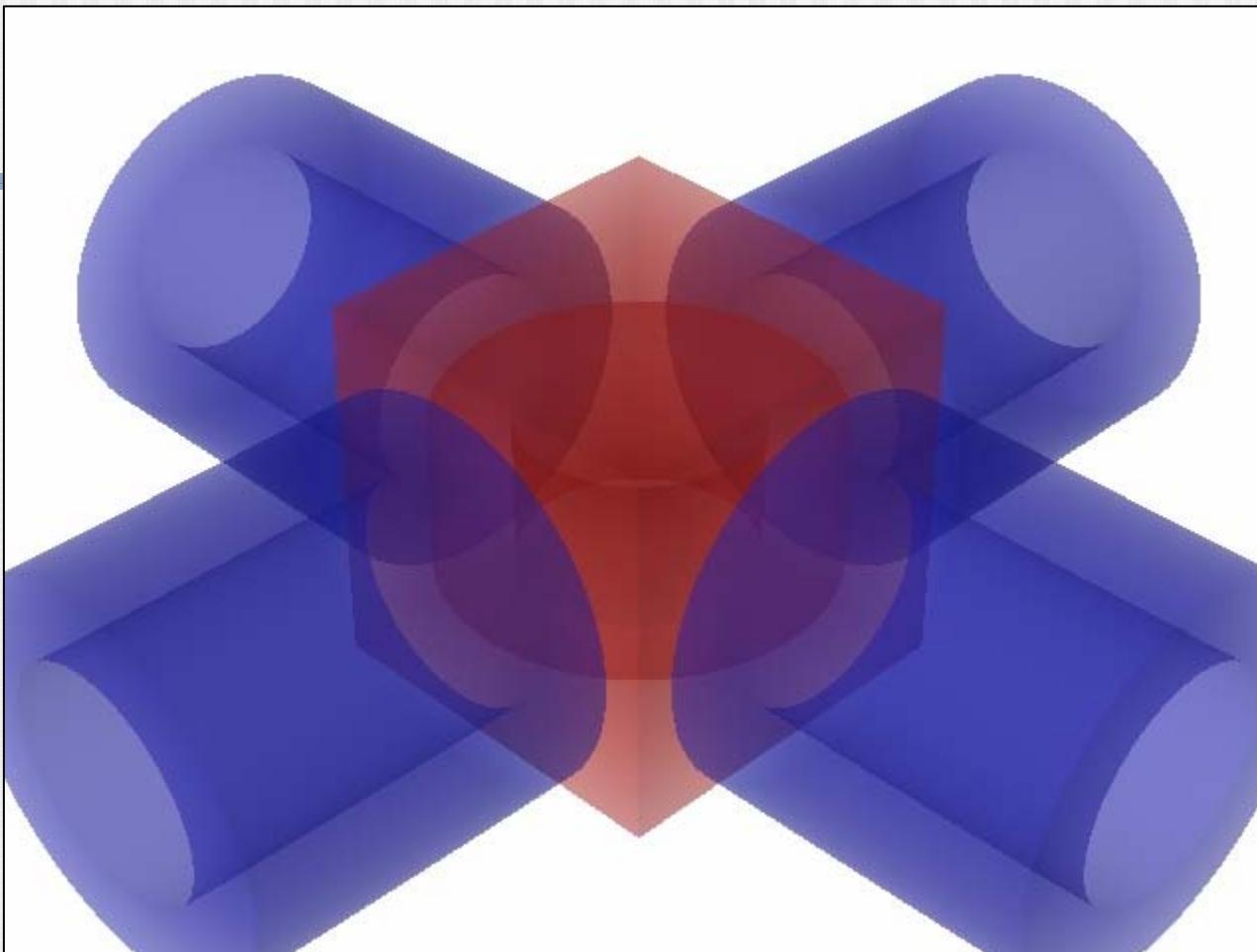
From Introduction to Geant4 Visualization by Joseph Perl, SLAC

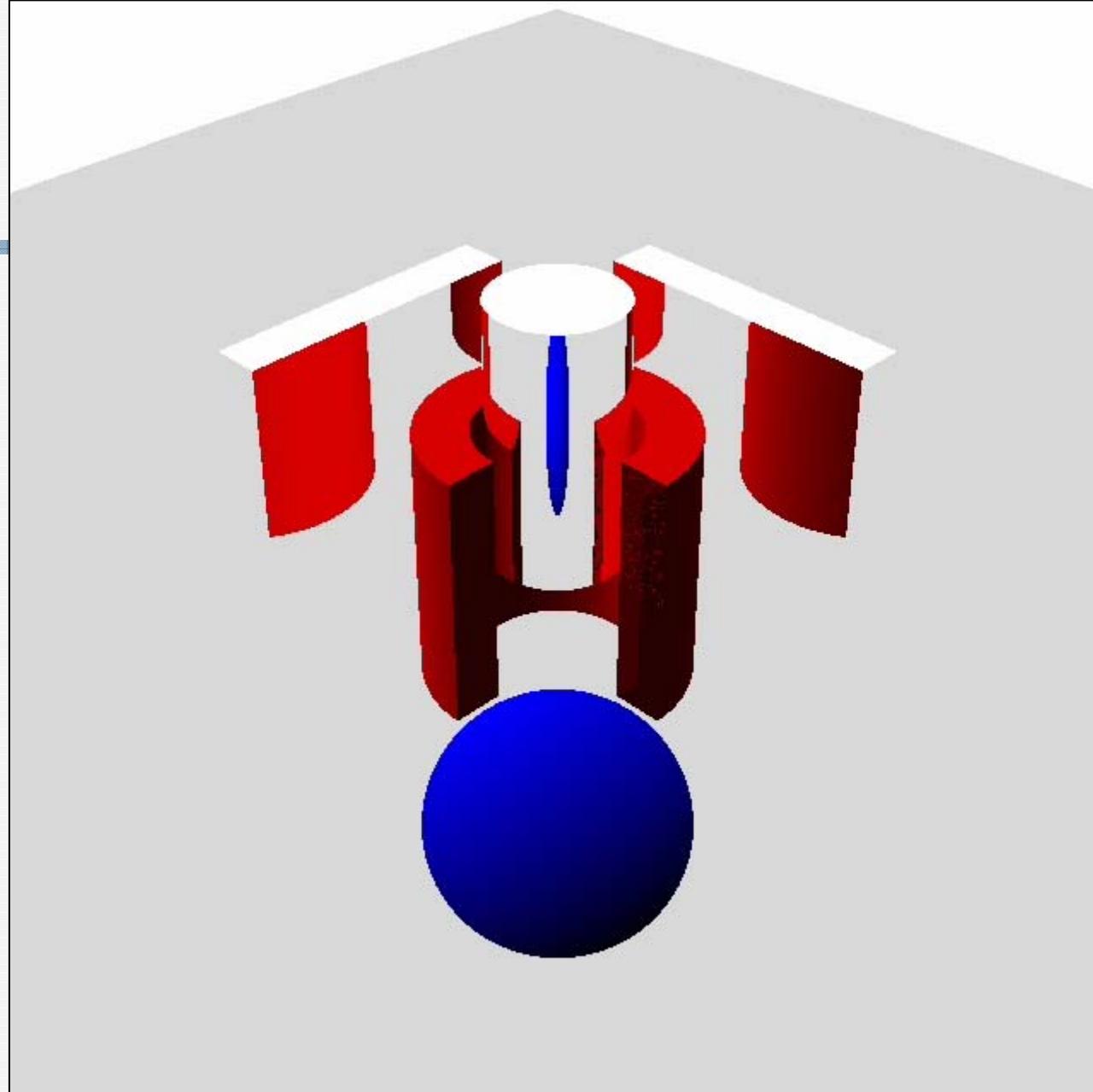


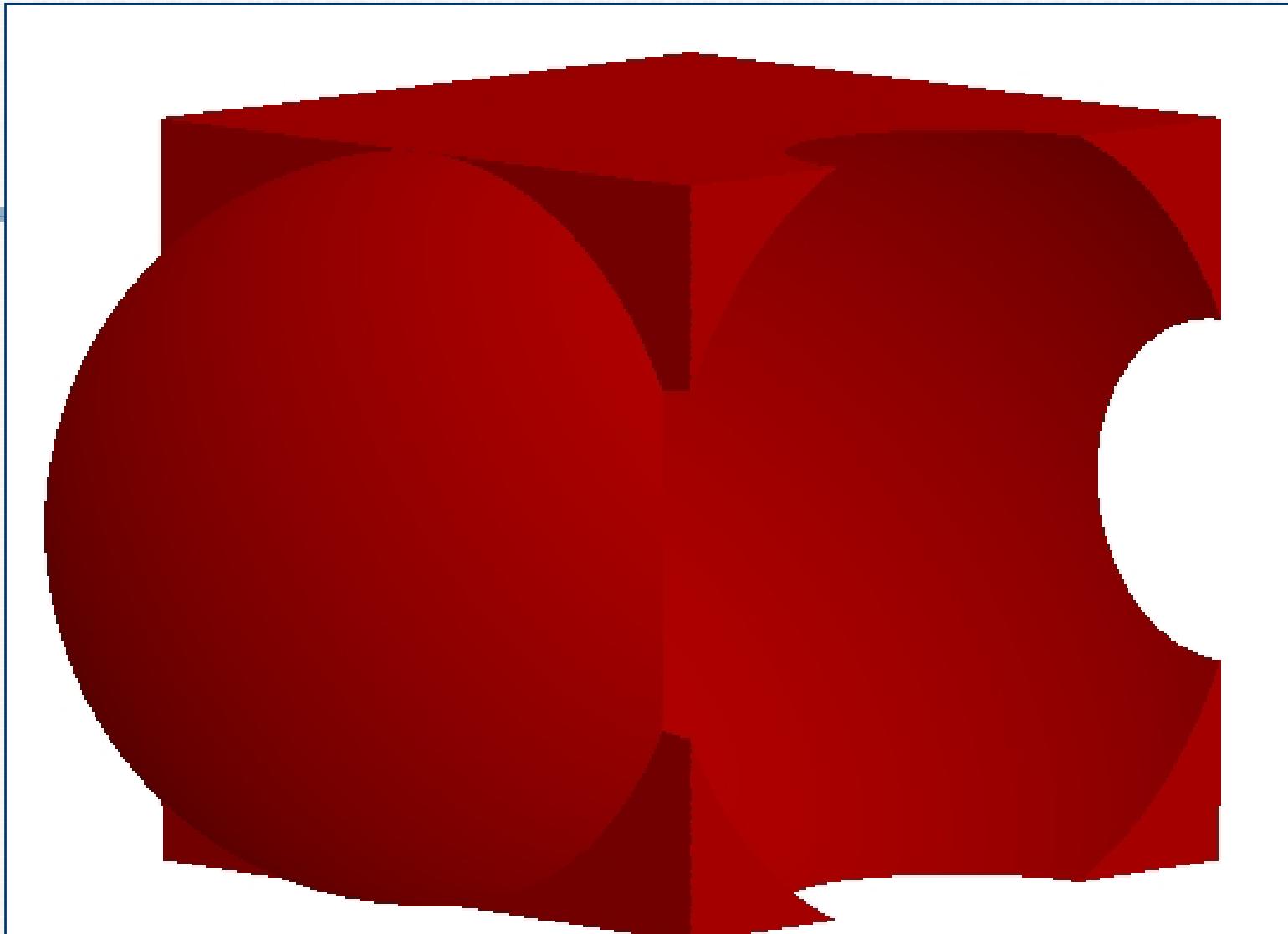
From Introduction to Geant4 Visualization by Joseph Perl, SLAC











8. Visualisation from C++ codes

- It is also possible to perform visualisation from the C++ code
- You can describe the visualisation commands in C++ codes via the `ApplyCommand()` method of the UI manager, as for any other command:
 - `pUI->ApplyCommand("/vis/...");`
- Or you can use `Draw()` methods of visualizable classes

9. Exercises

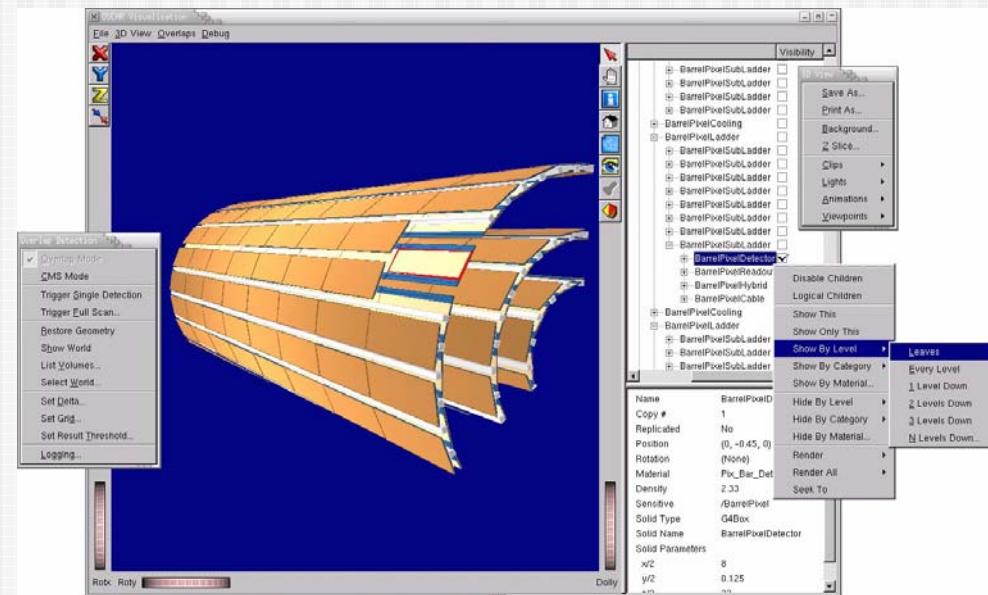
- Read and execute sample visualisation macros for examples/novice/N03
 - The macro files are “`exN03VisX.mac`”, where $X=0,1,2,\dots$
 - Explanation of macros is all described in the macro files as comment lines

10. Information

- Geant4 User Guide (and source codes)
- On-line documentation on Geant4 visualisation
 - <http://cern.ch/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/Visualization>
- For commands
 - <http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/Control/commands.html>
- SLAC tutorial
 - <http://geant4.slac.stanford.edu/SLACTutorial04/>

PART 2

Geant4 GUI



1. Select (G)UI (1)

- In the `main()`, according to the computer environments, construct a `G4UIsession` concrete class provided by Geant4 and invoke its `sessionStart()` method.
- Example:

```
■ G4UIsession* session=0;
if (argc==1)
    // Define UI session for interactive mode.
{
    // G4UITerminal is a (dumb) terminal
    session = new G4UITerminal;
}
```

1. Select (G)UI (2)

- Geant4 provides the following interfaces for various (G)UI:
 - **G4UITerminal**: C-shell like character terminal
 - **G4UITcsh**: tcsh-like character terminal with command completion, history, etc
 - **G4UIGAG**: Java based GUI
 - **G4UIXM**: Motif-based GUI, command completion, etc
- Note for **G4UITcsh**:
 - Use **G4UITerminal** with argument **G4UITcsh*** :

```
session = new G4UITerminal (new G4UITcsh);
```

2. Environmental Variables

- Users should use Configure to select and plug in (G)UI (or by setting environmental variables)
 - `setenv G4UI_USE_GUINAME`
- Example:
 - `setenv G4UI_USE_TERMINAL 1`
 - `setenv G4UI_USE_GAG 1`
 - `setenv G4UI_USE_XM 1`
- Note that Geant4 library should be installed with setting the corresponding `G4UI_BUILD_GUINAME_SESSION`

3. Useful GUI Tools Released by Geant4 Developers

- **GGE**: Geometry editor based on Java GUI
 - <http://erpc1.naruto-u.ac.jp/~geant4>
- **GPE**: Physics editor based on Java GUI
 - <http://erpc1.naruto-u.ac.jp/~geant4>
- OpenScientist: Interactive environment
 - <http://www.lal.in2p3.fr/OpenScientist>

PART 3

Geant4
DAVID & DTREE

1. Volume-Overlapping Detection with DAVID (1)

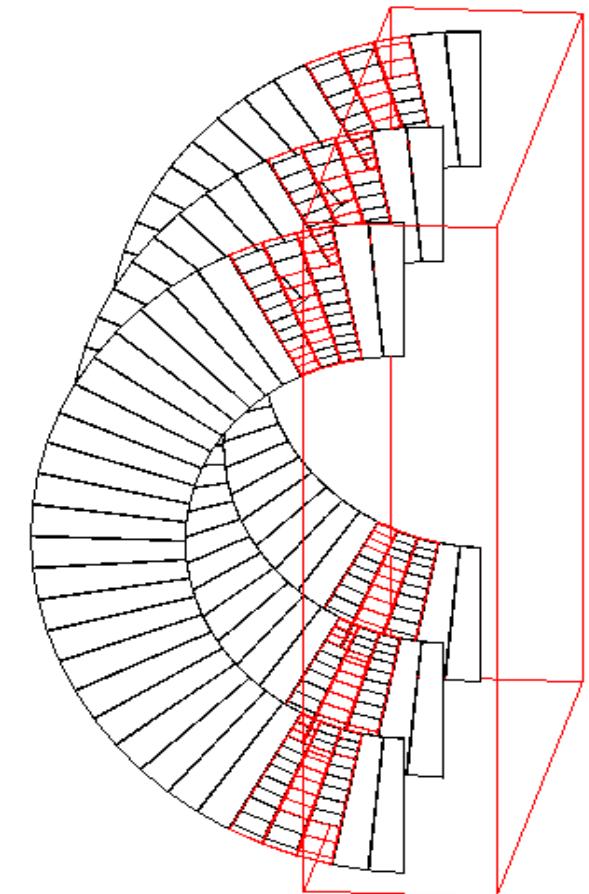
- DAVID (**DAWN**-based Visual **Volume** **Intersection** **Debugger**)
 - Automatically detects and highlights overlapping volumes
 - Precise visualization with DAWN
 - Interactive visualisation with VRML
 - DAVID also generates log files describing detailed information on the detected overlaps
 - Info & source:
 - <http://geant4.kek.jp/~tanaka>

1. Volume-Overlapping Detection with DAVID (2)

- Usage of DAVID
 - Switch the viewer of the DAWNFILE driver from renderer DAWN (default) to DAVID.
 - `setenv G4DAWNFILE_VIEWER david`
 - Then visualize volumes as usual with the DAWNFILE driver
 - Overlapping volumes (if any) are visualized
 - The view is stored in files `g4david.prim` (DAWN format) and `g4david.eps` (PostScript format)
 - Log file: `g4david.log`

1. Volume-Overlapping Detection with DAVID (3)

- Sample visualisation with overlapping volumes highlighted



1. Volume-Overlapping Detection with DAVID (4)

- Log file format

- PhysVolName.CopyNo Shape line_num
- The “line_num” is the line number of the overlapping volume in the DAWN-fomat file “g4.prim file” generated by Geant4

- Sample log file :

```
.....  
!!! INTERSECTED VOLUMES !!!  
caloPhys.0: Tubs: line 17  
caloPhys.1: Tubs: line 25  
.....
```

1. Volume-Overlapping Detection with DAVID (5)

- If no overlaps are detected, DAVID displays the following message:

```
-----  
!!! Number of intersected volumes : 0 !!!  
!!! Congratulations ! \(^o^)/ !!!  
-----
```

2. Visualising the Detector Geometry Tree (DTREE) (1)

- Selection of outputs:
 - ASCII-text format
 - GAG-window
 - XML file
- How to display a tree:
 - Idle> /vis/drawTree ! XXXTree
(XXXTree = ATree, GAGTree, XMLTree, etc)
(Default is Atree)
- Detail level is controlled with the “verbose” command:
 - /vis/XXXTree/verbose n

2. DTREE:Visualising Detector Geometry Tree (3-1)

- **ASCII Tree (ATree) :** verbose level 0 (default)

Format: PV_name + copy_number

World

"Calorimeter", copy no. 0

"Layer", copy no. -1 (10 replicas)

"Absorber", copy no. 0

"Gap", copy no. 0

2. DTREE:Visualising Detector Geometry Tree (3-2)

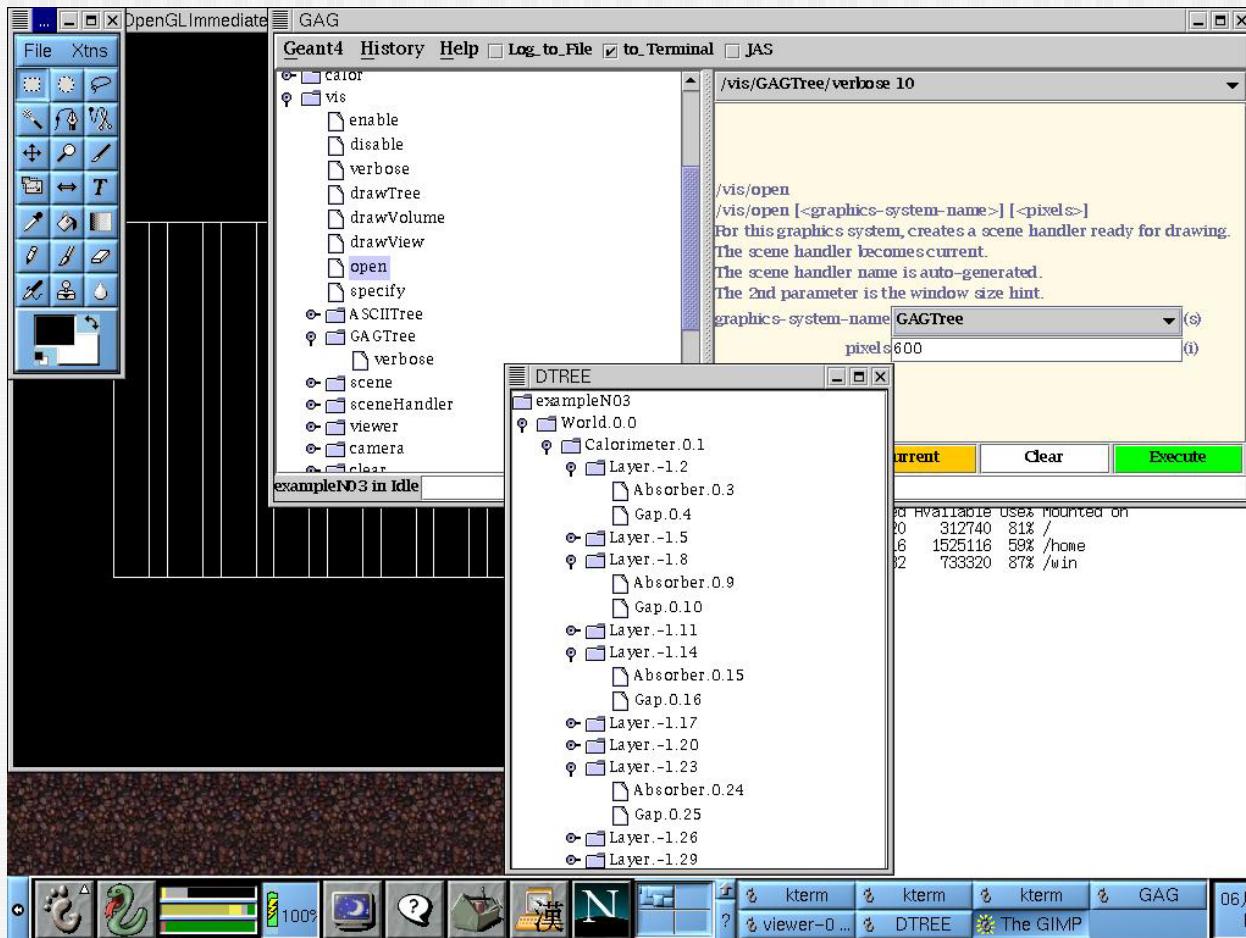
■ ASCII Tree (ATree)

- For other outputs, see guidance (help /vis/ASCIITree/verbose) or header of output
- Possibilities — with or without repetition
 - Physical volume and corresponding logical volume & solid
 - Volume
 - Mass of tree to specified depth
 - Mass of all descendants

```
"thePit":0 / "thePit" / "aBox" (G4Box), 8000 m3 , 0.1 mg/cm3  
"theMother1":0 / "aTest1" / "aBox" (G4Box), 1000 m3 , 0.1 mg/cm3  
Calculating mass(es)...  
Overall volume of "thePit":0, is 8000 m3  
Mass of tree, ignoring daughters at depth 1 and below, is 800 kg
```

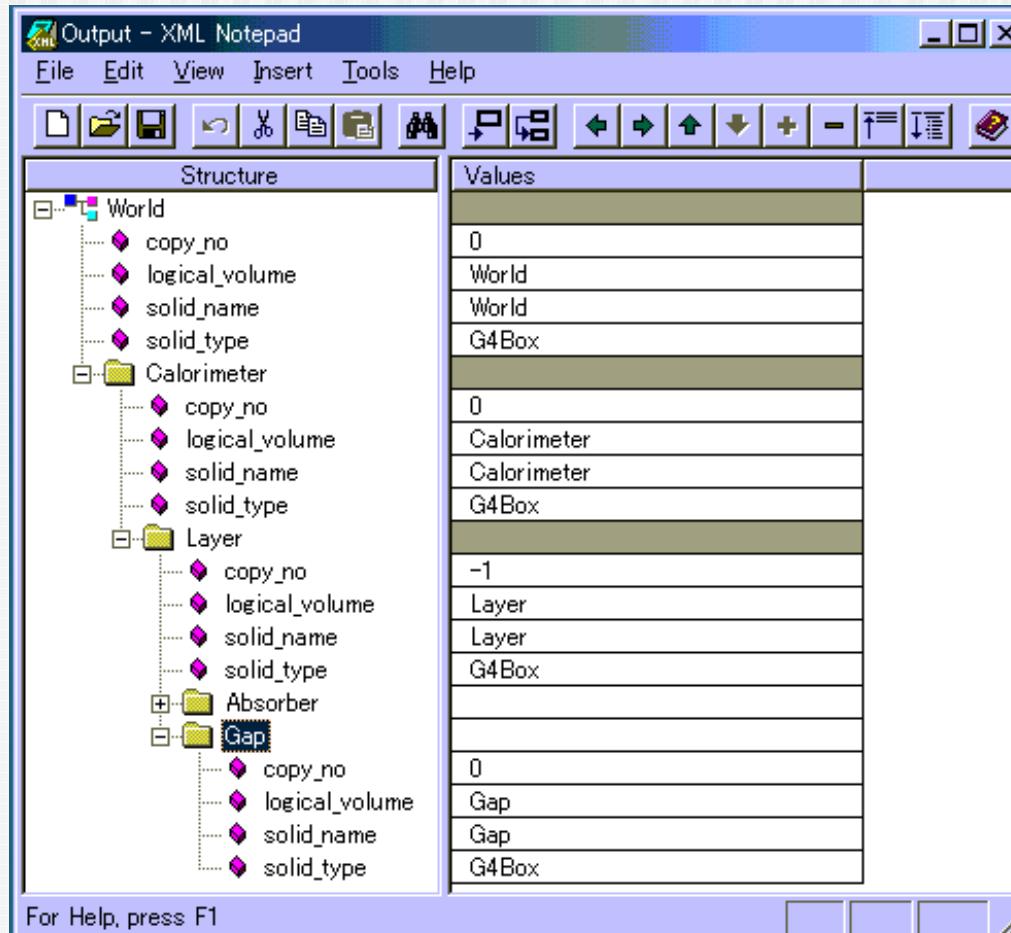
2. DTREE:Visualising Detector Geometry Tree (4)

■ GAG Tree



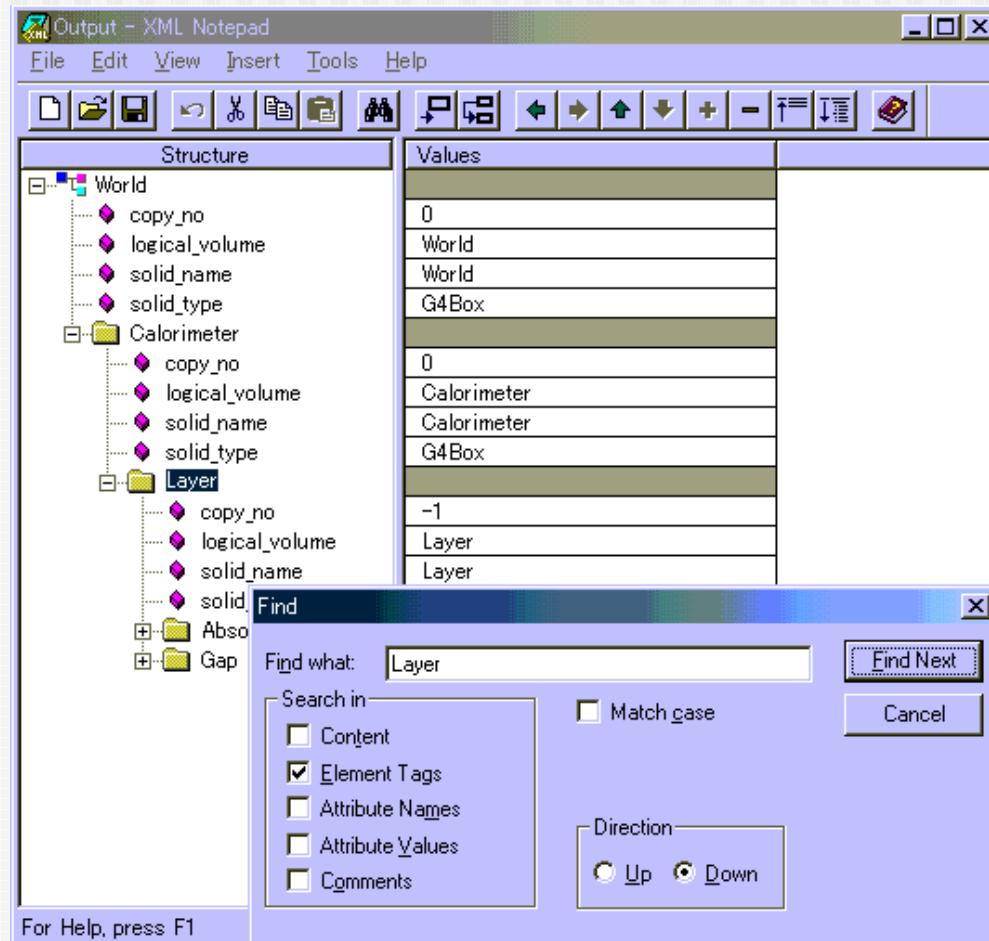
2. DTREE:Visualising Detector Geometry Tree (5-1)

■ XML Tree



2. DTREE:Visualising Detector Geometry Tree (5-2)

■ XML Tree (find)



PART 4

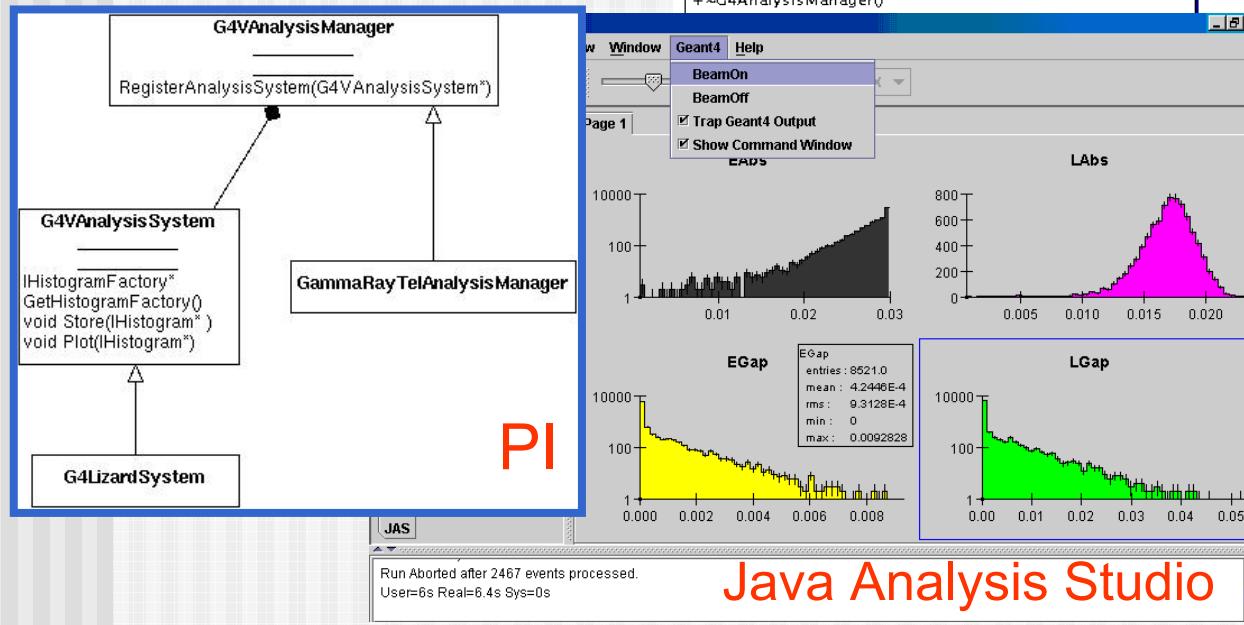
Geant4
AIDA & analysis

1. Interface to AIDA

Through
abstract interfaces

- ↓ No dependence
- ↓ Minimize coupling of components

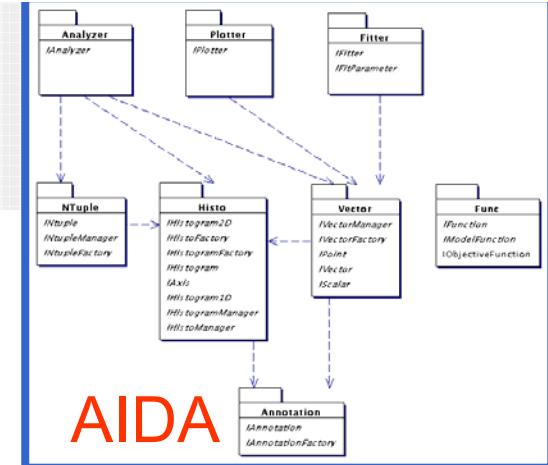
AIDA & Analysis Tools



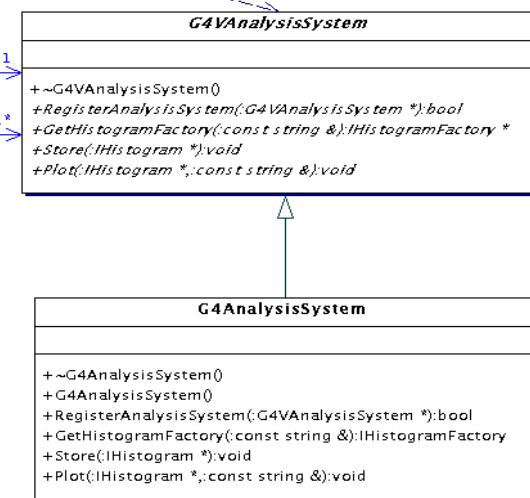
Java Analysis Studio

Visualisation & (G)UI - Geant4 Course

75



AIDA



2. Interfacing to Geant4

- AIDA (**A**bstract **I**nterfaces for **D**ata **A**nalysis) can be used in Geant4 by selecting the environmental variable **G4ANALYSIS_USE**
 - Requires AIDA headers installed in the system
 - Requires an AIDA compliant tool for analysis
- Tools for analysis compliant with AIDA interfaces currently are:
 - PI (Physicist Interfaces for AIDA Analysis)
 - JAS (Java Analysis Studio)
 - Open Scientist Lab

3. References ...

- AIDA
 - <http://aida.freehep.org>
- PI
 - <http://cern.ch/PI/>
- JAS (Java Analysis Studio)
 - <http://jas.freehep.org>
- Open Scientist Lab
 - <http://www.lal.in2p3.fr/OpenScientist>