GEANT4 BEGINNERS COURSE

GSSI, L'Aquila (Italy)
27-30 June 2016

Visualization

tutorial course

Introduction

- Geant4 Visualization must respond to varieties of **user requirements**
  - Quick response to survey geometry and events
  - Impressive special effects for demonstration
  - High-quality output for publications
  - Flexible camera control for debugging geometry
  - Tools for highlighting overlapping of physical volumes
  - Interactive picking of visualised objects
  - ...

- To get such a flexibility Geant4 <u>supports</u> several different external visualization systems

Visualizable Objects

- Simulation data you may like to see:
    - Detector components
    - Geometry hierarchy
    - A piece of physical volume, logical volume, and solid
    - Particle trajectories and tracking steps

    - Hits of particles in detector components
- Can also visualize 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)
    - Text
        - character strings for description
        - comments or titles ...
- Visualisation is performed either with commands (macro or interactive) or by writing C++ source codes of user-action classes

Visualization Attributes

- Necessary to customize visualization, not included in geometry
  - Colour, visibility, wireframe/solid style, etc
- A `G4VisAttributes` class holds all visualization attributes to be assigned to a visualizable object

  ```
  G4VisAttributes* myVisAtt = new G4VisAttributes();
  ```

- To set attributes:

  ```
  G4bool visibility = false;   boolean visibility flag
                               by defalut is true

  myVisAtt->SetVisibility(visibility);    visualization is skipped
  G4Color red(1,0,0);
  myVisAtt->SetColor(red);
  …
  ```

- Default attributes are used if a visualizable object was not assigned its own attributes

4

Color

- Class **G4Color** allows to build colors; it is instantiated by giving RGB components to its constructor:

```
G4Colour::G4Colour(G4double r = 1.0,
                   G4double g = 1.0,
                   G4double b = 1.0 )
```

  - The default arguments define "white" color

  - For instance:
```
G4Color red(1.0, 0.0, 0.0);
G4Color blue(0.0, 0.0, 1.0);
G4Color yellow(1.0, 1.0, 0.0);
```

- Class **G4VisAttributes** can be instantiated directly with a color of your choice:
```
G4VisAttributes* myVisColor =
     new G4VisAttributes(G4Color(1.,0.,0.));
```

Assigning G4VisAttributes to a Logical Volume

- Once you have defined visualization attributes, they have to be assigned to the visualizable object, for example a volume of your detector

- Class **G4LogicalVolume** holds a pointer of **G4VisAttributes**

```
G4Colour brown(0.7, 0.4, 0.1);
G4VisAttributes* copperVisAtt =
    new G4VisAttributes(brown);
copperLV->SetVisAttributes(copperVisAtt);
```

6

Polyline

- A set of successive line segments
- Defined with a class **G4Polyline**
- Used to visualize tracking steps, particle trajectories, coordinate axes, any other user-defined polyline
- **G4Polyline** is defined as a list of **G4Point3D** objects → polygonal line vertices

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

// Instantiation
  G4Polyline x_axis;

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

//    Color
  G4Colour red ( 1.0, 0.0, 0.0 );
  G4VisAttributes att ( red );
  x_axis.SetVisAttributes( &att );
```

Marker

- Set a mark to an arbitrary 3D position
- Usually used to visualize hits of particles
- 2-dimensional primitive with **shape** (square, circle, text), **color**.
- Set marker properties with
  - `SetPosition( const G4Point3D& )`
  - `SetWorldSize( G4double real_3d_size )`
  - `SetScreenSize( G4double 2d_size_pixel )`
- 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)`

Drawn only by OpenGL drivers (excluding Windows OpenGL)

Example C++ code for marker:

```cpp
G4Point3D position(0,0,0);
G4Circle circle(position);
```
Create a circle in a given position

```cpp
circle.SetScreenDiameter(1.0);
circle.SetFillStyle (G4Circle::filled);
```
Set diameter and style

```cpp
G4Colour colour(1.,0.,0.);
G4VisAttributes attribs(colour);
circle.SetVisAttributes(attribs);
```
Set colour and vis attributes

9

G4 Visualisation Drivers

- **Visualization drivers** are interfaces of Geant4 to 3D graphics software

- You can select your favorite one(s) depending on your purposes
  - Demo
  - Preparing precise figures for journal papers
  - Publication of results on Web
 - Debugging geometry
  - Etc.

Available visualization drivers

Geant4 provides several visualization drivers tailored to different purposes

- Some of them work directly from Geant4
  - **OpenGL**
  - **Qt**
  - **OpenInventor**
  - **RayTracer**
  - **ASCIITree**
  - **Wt → Experimental, use with caution**
- For other, Geant4 will dump a file in a specific format that you can later visualize
  - **HepRep**
  - **DAWN**
  - **VRML**
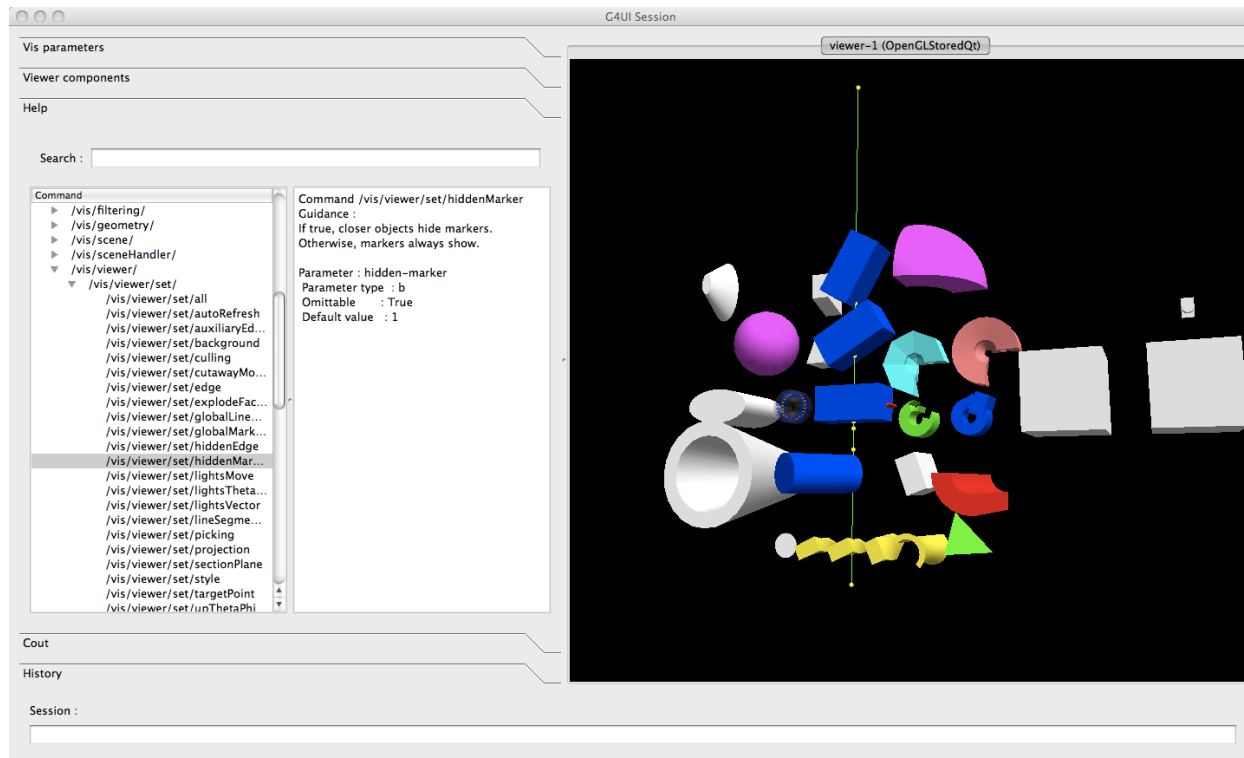 - **gMocren**

A quick overview ...

OpenGL

- View directly from Geant4
- Requires additional GL libraries (already included on most Linux and Windows systems)
- Rendered, photorealistic image with some interactive features
- zoom, rotate, translate
- Fast response
- Print to vector or pixel graphics
- Movies

Qt
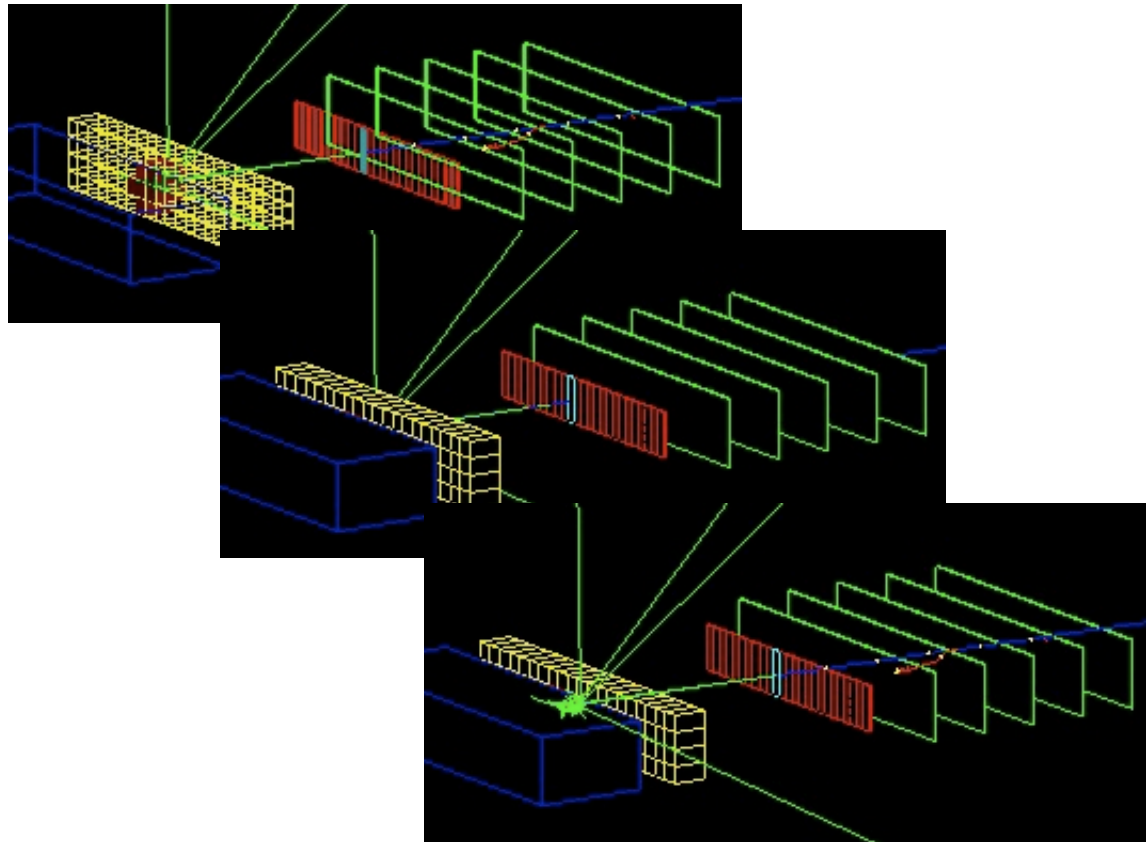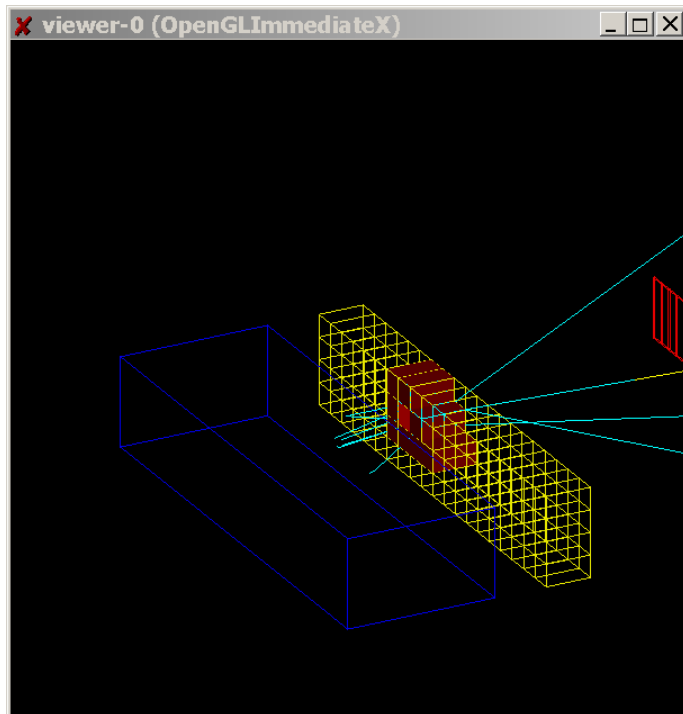
- View directly from Geant4
- Requires addition of Qt and GL libs (freely available on most operating systems)
- Rendered, photorealistic image
- Many interactive features
- zoom, rotate, translate
- Fast response
- Expanded printing ability (vector and pixel graphics)
- Easy interface to make Movies

13

OpenInventor

- Control from the OpenInventor GUI (view directly from Geant4)
- Requires addition of OpenInventor libs (freely available for most Linux and Windows systems)
- Rendered, photorealistic image
- Many interactive features
  - zoom, rotate, translate
  - click to "see inside" opaque volumes
  - click to show attributes (momentum, etc., dumps to standard output)
- Fast response
- Expanded printing ability (vector and pixel graphics)
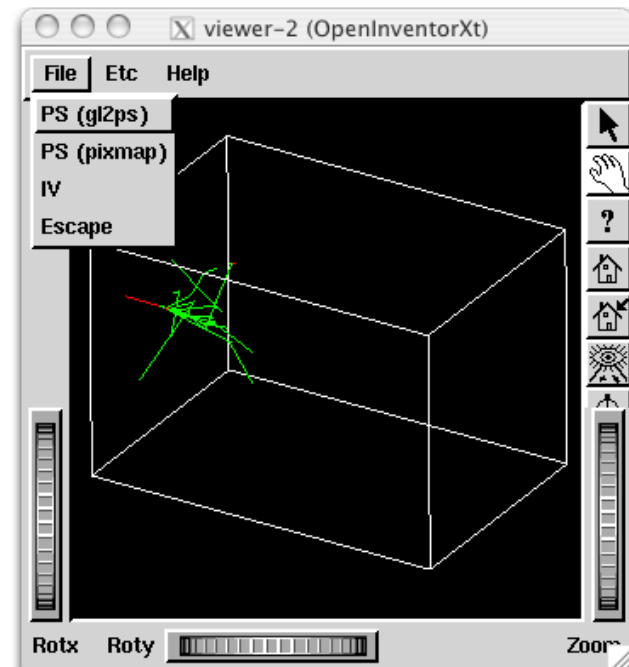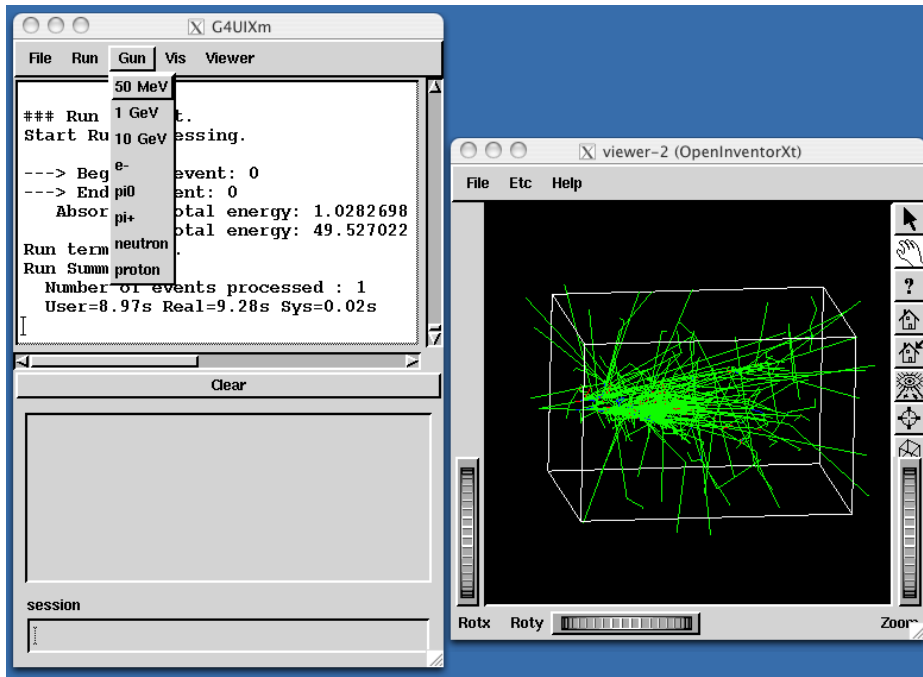
HepRep

- Create a file to view in the HepRApp Browser, WIRED4 Jas Plugin or FRED Event Display
- Requires one of the above browsers (freely available for all systems)
- Wireframe or simple area fills (not photorealistic)
- Many interactive features
  - zoom, rotate, translate
  - click to show attributes (momentum, etc.)
  - special projections (FishEye, etc.)
  - control visibility from hierarchical (tree) view of data
- Hierarchical view of the geometry
- Export to many vector graphic formats (PostScript, PDF, etc.)
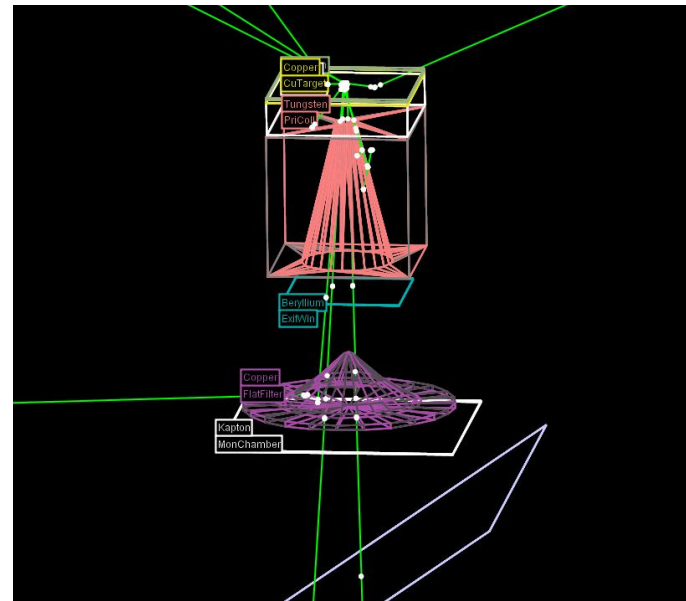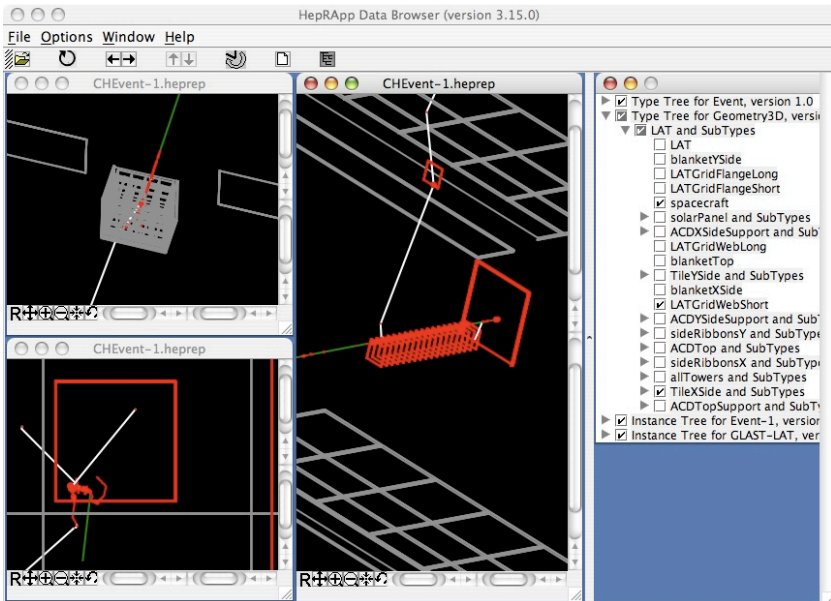
Dawn

- Create a file to view in the DAWN Renderer
- Requires DAWN, available for all Linux and Windows systems.
- Rendered, photorealistic image
- No interactive features once at PostScript stage
- Highest quality technical rendering - vector PostScript
- View or print from your favorite PostScript application
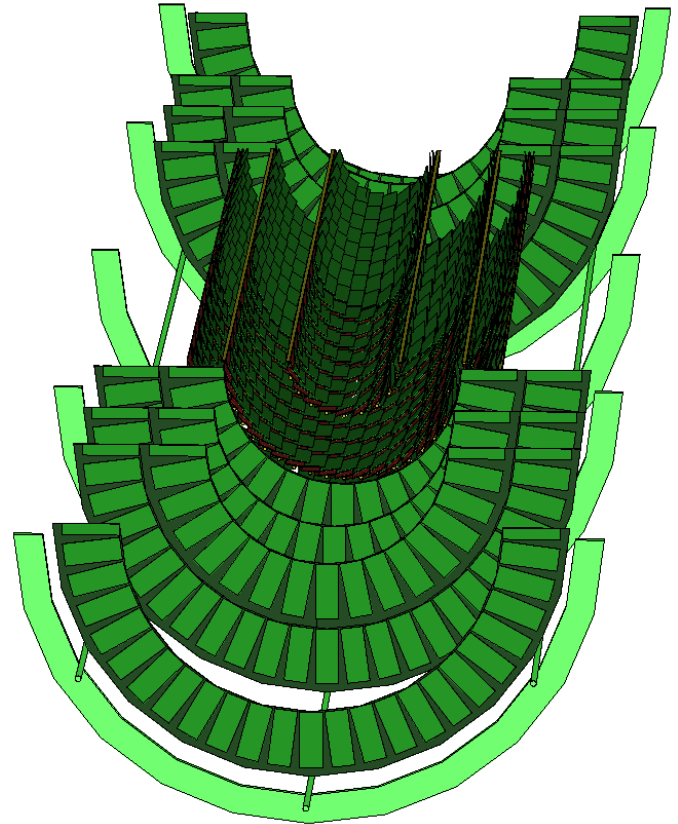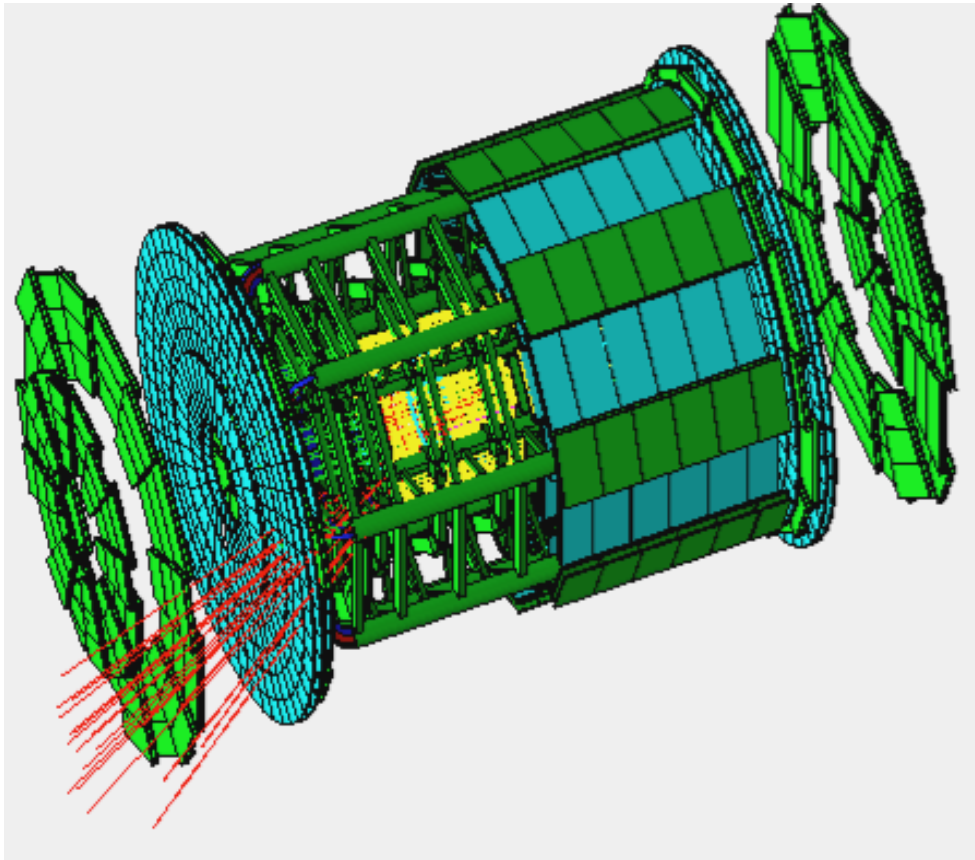
VRML

- Create a file to view in any VRML browser (some as web browser plug-ins).
- Requires VRML browser (many different choices for different operating systems).
- Rendered, photorealistic image with some interactive features
  - zoom, rotate, translate
- Limited printing ability (pixel graphics, not vector graphics)
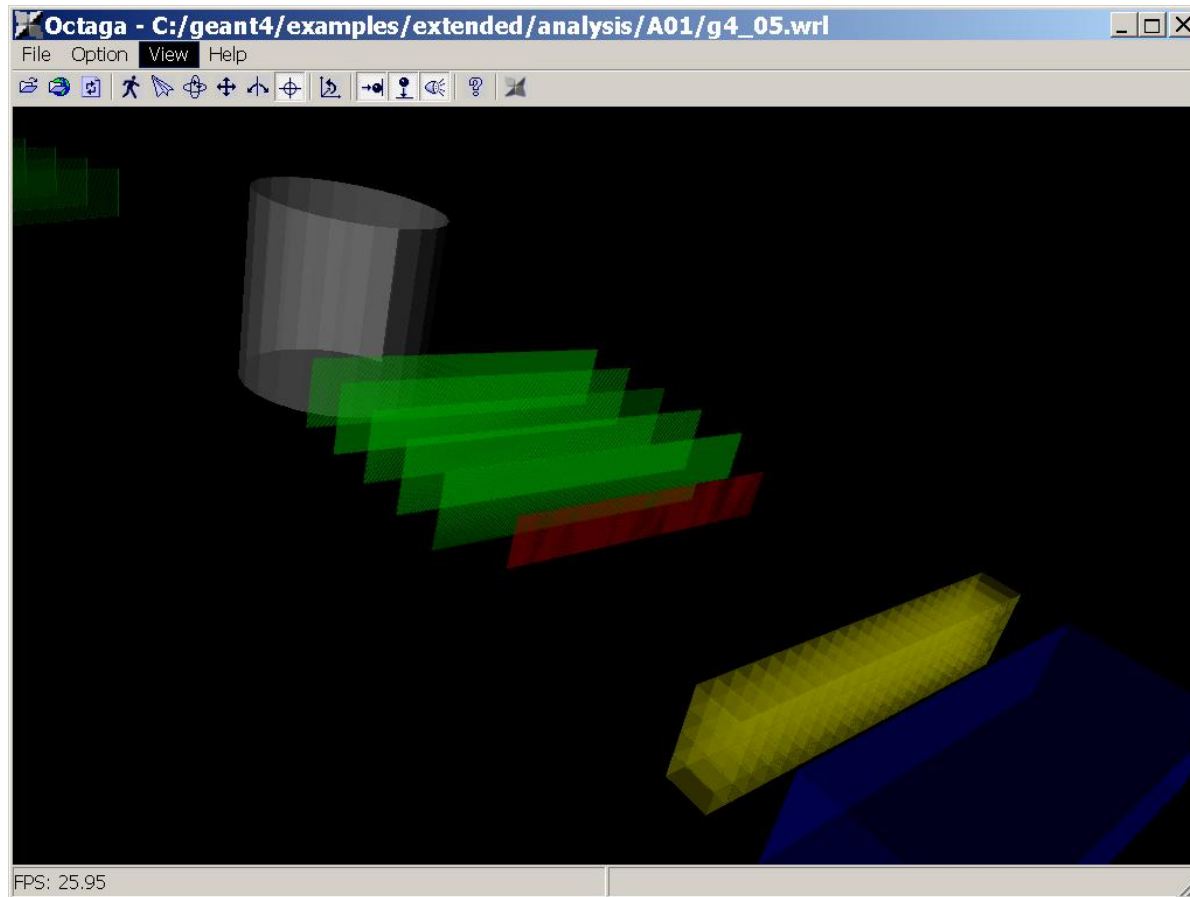
RayTracer

- Create a jpeg file (and with RayTracerX option, also draws to x window)
- Forms image by using Geant4's own tracking to follow photons through the detector
- **Can show geometry but not trajectories**
- **Can render any geometry that Geant4 can handle** (such as Boolean solids)
  - no other Vis driver can handle every case
- Supports shadows, transparency and mirrored surfaces
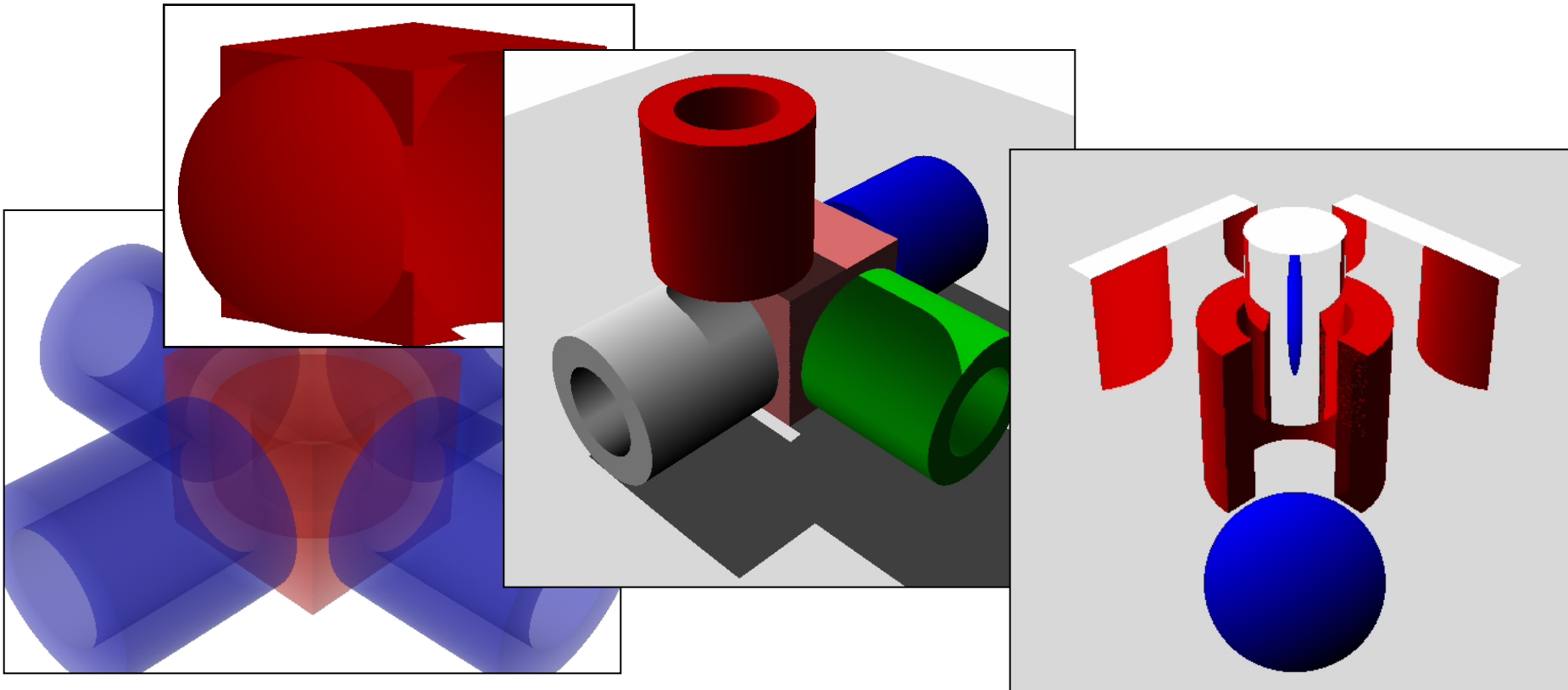
gMocren

- Create a file to be viewed in the gMocren browser.
- Requires gMocren, available for all Linux and Windows systems (with Mac coming soon)
- Can overlay patient scan data (from DICOM) with Geant4 geometry, trajectories and dose
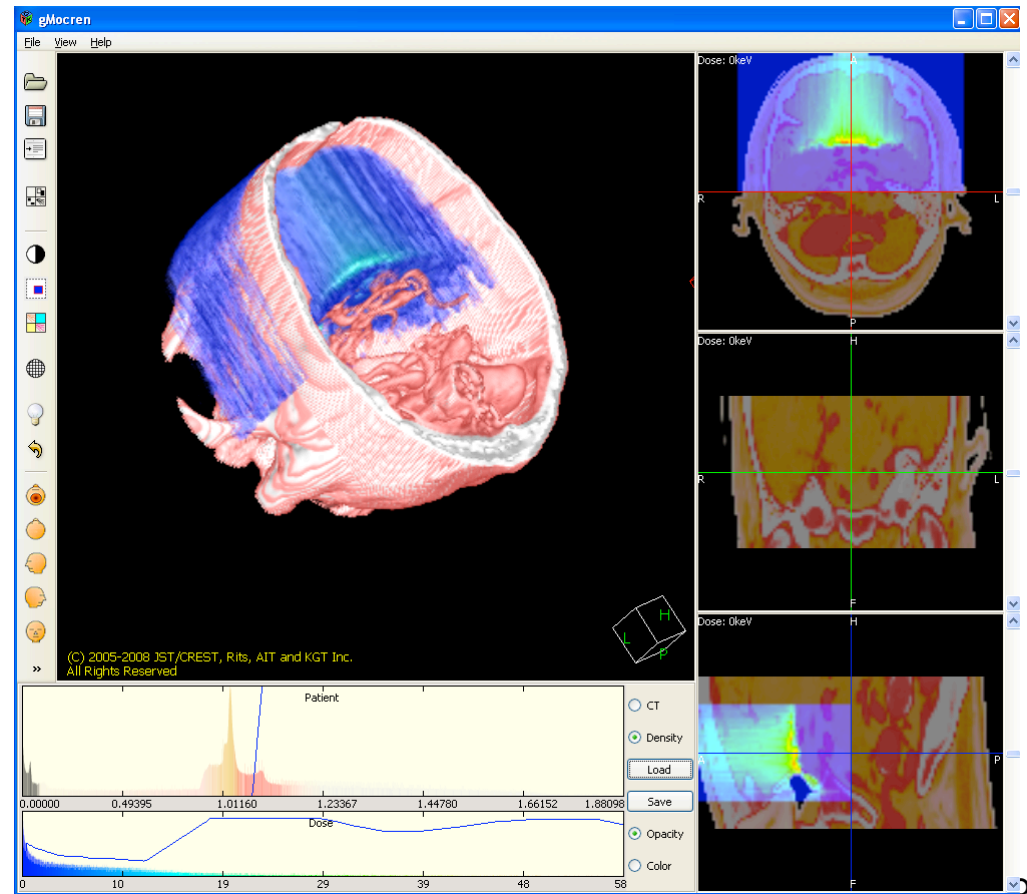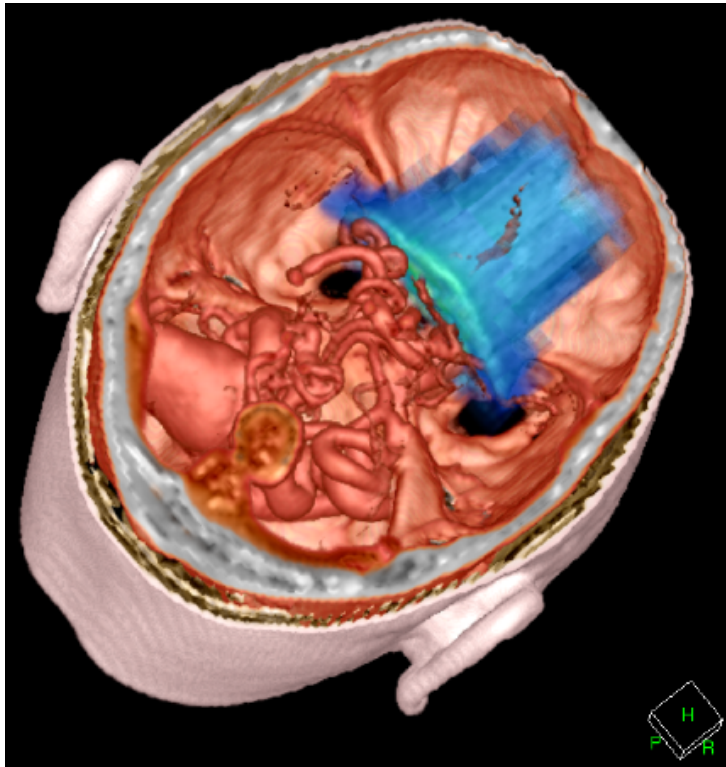
ASCIITree

- Text dump of the geometry hierarchy (not graphical)
- Control over level of detail to be dumped
- **Can calculate mass and volume of any hierarchy of volumes**

Ex.:

/vis/viewer/flush
- "worldPhysical":0
- "magneticPhysical":0
- "firstArmPhysical":0
- "hodoscope1Physical":0
- ...

 /vis/viewer/flush
- "worldPhysical":0
- "magneticPhysical":0
- "firstArmPhysical":0
- "hodoscope1Physical

Calculating mass(es)...
- Overall volume of "worldPhysical":0, is 2400 m3
- Mass of tree to unlimited depth is 22260.5 kg

Wt

- View directly from Geant4 across a Web browser.
- Requires addition of Wt libs (freely available on most operating systems)
- Require a Web browser with WebGL enable.
- Rendered, photorealistic image
- Many interactive features
- zoom, rotate, translate
- Fast response

WARNING: this driver is experimental and should be used with caution
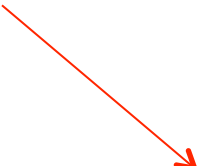
How to use visualization drivers

- Visualization should be switched on using the variable `G4VIS_USE`
- To select/use visualization driver(s) it is needed the proper environmental variable that you either set by hand or that is set for you by GNUMake or Cmake support scripts

- Example (DAWN, OpenGLXlib, and VRML drivers):
  - `setenv G4VIS_USE_DAWN  1`
  - `setenv G4VIS_USE_OPENGLX    1`
  - `setenv G4VIS_USE_VRML   1`

G4VisManager

- To make your Geant4 application perform visualization, you must instantiate and initialize "your" Visualization Manager in the main() function.

```
.....
// Your Visualization Manager
#include "G4VisExecutive.hh"
.....

// Instantiation and initialization of the Visualization Manager
#ifdef G4VIS_USE
G4VisManager* visManager = new G4VisExecutive;
visManager->Initialize();
#endif
.....
#ifdef G4VIS_USE
delete visManager;
#endif
```

Derive your own from
G4VisManager or simply use
G4VisExecutive

Useful definitions

In using the visualization in Geant4, it is useful to know the concept of "scene", "scene handler", and "viewer"

SCENE
visualizable raw 3D data

SCENE HANDLER
modeler of graphics data
processes raw data in a scene

VIEWER
generates images based on data
processed by a scene handler

DRIVER
OpenGL, Qt, HepRep, VRML, etc.
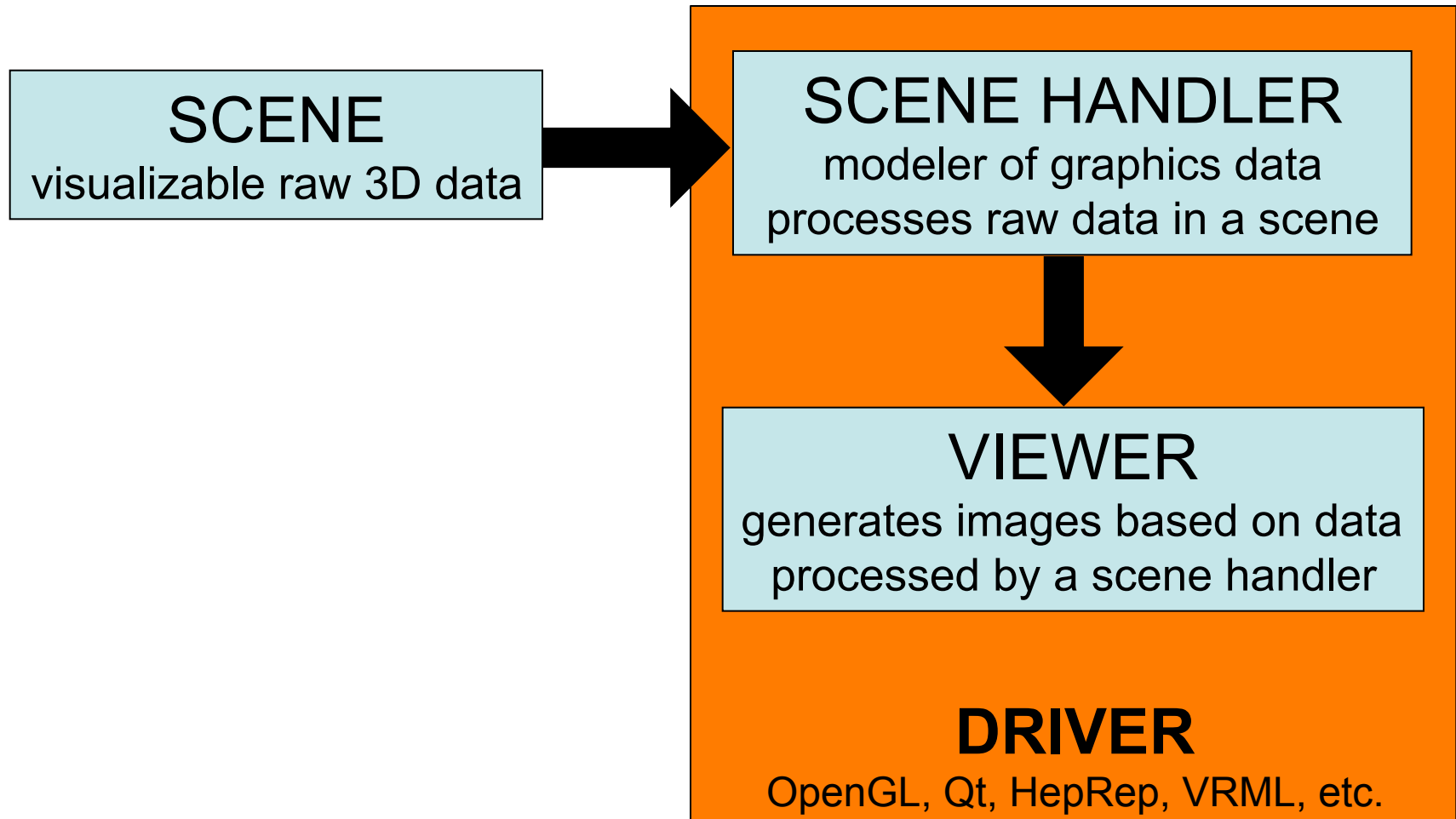
Visualization commands

There are some frequently-used built-in visualization commands in Geant4, that you may like to try

```
Command directory path : /vis/
 Sub-directories :
   /vis/ASCIITree/          Commands for ASCIITree control.
   /vis/heprep/             HepRep commands.
   /vis/rayTracer/          RayTracer commands.
   /vis/gMocren/            gMocren commands.
   /vis/ogl/                G4OpenGLViewer commands.
   /vis/modeling/           Modeling commands.
   /vis/filtering/          Filtering commands.
   /vis/geometry/           Operations on vis attributes of Geant4 geometry.
   /vis/set/                Set quantities for use in future commands where appropriate.
   /vis/scene/              Operations on Geant4 scenes.
   /vis/sceneHandler/       Operations on Geant4 scene handlers.
   /vis/touchable/          Operations on touchables.
   /vis/viewer/             Operations on Geant4 viewers.
 Commands :
   verbose *                Simple graded message scheme - digit or string (1st character defines):
   initialize *             Initialise visualisation manager.
   abortReviewKeptEvents *  Abort review of kept events.
   enable *                 Enables/disables visualization system.
   disable *                Disables visualization system.
   list *                   Lists visualization parameters.
   reviewKeptEvents *       Review kept events.
   drawTree * (DTREE)       Creates a scene consisting of this physical volume and
                            produces a representation of the geometry hieracrhy.
   drawView *               Draw view from this angle, etc.
   drawVolume *             Creates a scene containing this physical volume and asks the
                            current viewer to draw it.  The scene becomes current.
   open *                   Creates a scene handler ready for drawing.
   specify *                Draws logical volume with Boolean components, voxels and readout geometry.
```


Guidance is hierarchical, providing full detail on all commands

25

Commands to visualize detectors

`/vis/open OGLIX`   create scene handler + viewer (driver)

` help /vis/open`   show available drivers

`/vis/viewer/reset`
`/vis/viewer/set/viewpointThetaPhi 70 20`
`/vis/viewer/set/style wireframe`   set camera parameters
                                          set drawing style

`/vis/drawVolume`   set detector geometry as obj to
or   visualize, and registers it
`/vis/specify logicLAr`   set specific logical volume
                           for visualization

`/vis/viewer/flush`   close visualization

These commands can be given interactively or executed via macro. Most Geant4 examples include a vis.mac that you can inspect and use.

Commands to Visualize Events

`/tracking/storeTrajectory`   Store trajectories
                                     for visualization

`/vis/open DAWNFILE`   Scene handler and viewer
                                     (for ex. DAWN)

`/vis/scene/add/axes 0 0 0 500 mm`
`/…`   Optional settings
                                     (axes, viewpoint, etc.)

`/vis/scene/create`   Creates an empty scene

`/vis/scene/add/volume`   Adds world volume
`/vis/scene/add/trajectories`   Adds trajectoriies
`/vis/scene/add/hits`   Adds hits

`/run/beamOn 10`   Shoots events
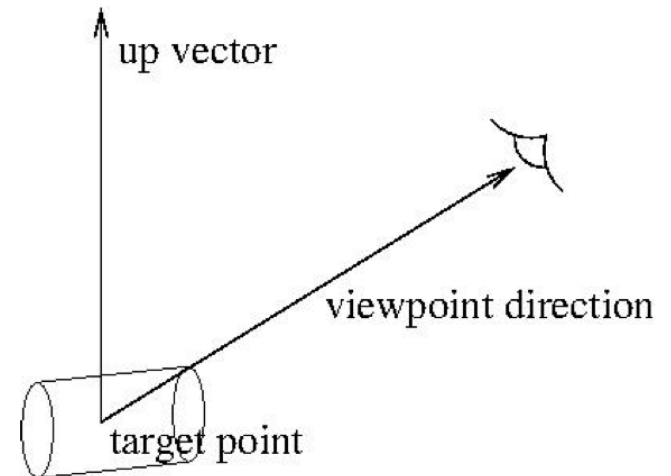   (end of visualization)

Some /vis/viewer/… commands

Camera settings

```
/vis/viewer/reset

/vis/viewer/viewpointThetaPhi <theta> <phi>

/vis/viewer/set/upVector    <x> <y> <z>

/vis/viewer/set/targetPoint <x> <y> <z>
```


Zooming

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

/vis/viewer/zoomTo <absolute_scale_factor>
```

Trajectory Filtering

Useful if you only want to view interesting trajectories discarding uninteresting ones.

- Soft filtering: trajectories are marked as invisible (but still written). Some drivers allows to toggle them back to visible

- Hard filtering: uninteresting trajectories are not even written. Useful to avoid huge graphics file

Available trajectory filtering models:

- G4TrajectoryChargeFilter (chargeFilter) → by electric charge

- G4TrajectoryParticleFilter (particleFilter) → by particle type

- G4TrajectoryOriginVolumeFilter (originVolumeFilter) → by trajectory originating volume

- G4TrajectoryAttributeFilter (attributeFilter) → by trajectory attribute

Multiple filters are automatically chained together

Filters can be configured either by commands or in compiled code

Filtering example I

Filter by particle type

`/vis/filtering/trajectories/create/particleFilter`

`/vis/filtering/trajectories/particleFilter-0/add gamma`   only gammas pass

`/vis/filtering/trajectories/particleFilter-0/invert true`   invert to pass anything other than gammas

`/vis/filtering/trajectories/particleFilter-0/active false`   inactivate filter

Filter by charge

`/vis/filtering/trajectories/create/chargeFilter`

`/vis/filtering/trajectories/chargeFilter-0/add 0`   only neutrals pass

`/vis/filtering/trajectories/chargeFilter-0/reset true`   reset filter

`/vis/filtering/trajectories/chargeFilter-0/add -1`   reconfigure to pass only negatively charged trajectories

List all configured filters

`/vis/filtering/trajectories/list`

Filtering example I

Filter by attribute

Only particle with momentum in 2.5MeV and 1000 MeV range pass

```
/vis/filtering/trajectories/create/attributeFilter
/vis/filtering/trajectories/attributeFilter-0/setAttribute IMag
/vis/filtering/trajectories/attributeFilter-0/addInterval 2.5 MeV 1000 MeV
```

Trajectory Drawing

- Trajectory drawing styles are specified through trajectory drawing models
- A user-defined trajectory drawing model can override the default context according to the properties of a given trajectory

Available trajectory drawing models:
- G4TrajectoryGenericDrawer (generic)
- G4TrajectoryDrawByCharge (drawByCharge)→ by electric charge
- G4TrajectoryDrawByParticleID (drawByParticleID)→ by particle type
- G4TrajectoryDrawByOriginVolume (drawByOriginVolume) → by trajectory originating volume
- G4TrajectoryDrawByAttribute (drawByAttribute) → by trajectory attribute

Drawing Modeling Examples

Modeling by charge

Set positively and negatively charged trajectories green; set neutral trajectories to white

```
/vis/modeling/trajectories/create/drawByCharge
/vis/modeling/trajectories/drawByCharge-0/set 1 green
/vis/modeling/trajectories/drawByCharge-0/set -1 green
/vis/modeling/trajectories/drawByCharge-0/set 0 white
```

Modeling by attribute

Set red color for particles created by Bremsstrahlung

```
/vis/modeling/trajectories/create/drawByAttribute
/vis/modeling/trajectories/drawByAttribute-0/setAttribute CPN
/vis/modeling/trajectories/drawByAttribute-0/addValue brem_key eBrem
/vis/modeling/trajectories/drawByAttribute-0/brem_key/setLineColour red
```

Thanks for your attention

Summary

- Geant4 can be used to visualize set-ups, tracks and other objects (e.g. axes, markers)
- A number of visualization drivers is available, each with its pros and cons
- Visualization can be controlled interactively or by macro, using Geant4 built-in commands
- Several advanced commands for specific visualization requirements are available

Polyline and Marker

- Polyline and marker are defined in the graphics_reps category

- They are available to model 3D scenes for visualization

Filtering by attribute example

/vis/modeling/trajectories/drawByAttribute-0/setAttribute IMag

Momentum filter

/vis/modeling/trajectories/drawByAttribute-0/addInterval interval1 0.0 keV 2.5MeV
/vis/modeling/trajectories/drawByAttribute-0/addInterval interval2 2.5 MeV 5 MeV
/vis/modeling/trajectories/drawByAttribute-0/addInterval interval3 5 MeV 7.5 MeV
/vis/modeling/trajectories/drawByAttribute-0/addInterval interval4 7.5 MeV 10 MeV
/vis/modeling/trajectories/drawByAttribute-0/addInterval interval5 10 MeV 12.5 MeV
/vis/modeling/trajectories/drawByAttribute-0/addInterval interval6 12.5 MeV 10000 MeV

Momentum interval based colour scale

/vis/modeling/trajectories/drawByAttribute-0/interval1/setLineColourRGBA 0.8 0 0.8 1
/vis/modeling/trajectories/drawByAttribute-0/interval2/setLineColourRGBA 0.23 0.41 1 1
/vis/modeling/trajectories/drawByAttribute-0/interval3/setLineColourRGBA 0 1 0 1
/vis/modeling/trajectories/drawByAttribute-0/interval4/setLineColourRGBA 1 1 0 1
/vis/modeling/trajectories/drawByAttribute-0/interval5/setLineColourRGBA 1 0.3 0 1
/vis/modeling/trajectories/drawByAttribute-0/interval6/setLineColourRGBA 1 0 0 1

Configure visualisation properties

/vis/filtering/trajectories/create/attributeFilter
/vis/filtering/trajectories/attributeFilter-0/setAttribute IMag
/vis/filtering/trajectories/attributeFilter-0/addInterval 2.5 MeV 1000 MeV

Momentum filter

/vis/filtering/trajectories/create/particleFilter
/vis/filtering/trajectories/particleFilter-0/add gamma

Gamma filter