UI Interface and visualization

2nd GEANT4 international school and ROOT analysis concepts

November 17-21, Catania

Steering the simulation

It can be done in three different ways:

✓everything hard-coded in the C++ source (also the number of events to be shot). You need to re-compile for any change (not very smart, actually!)

✓batch session (via an ASCII macro)

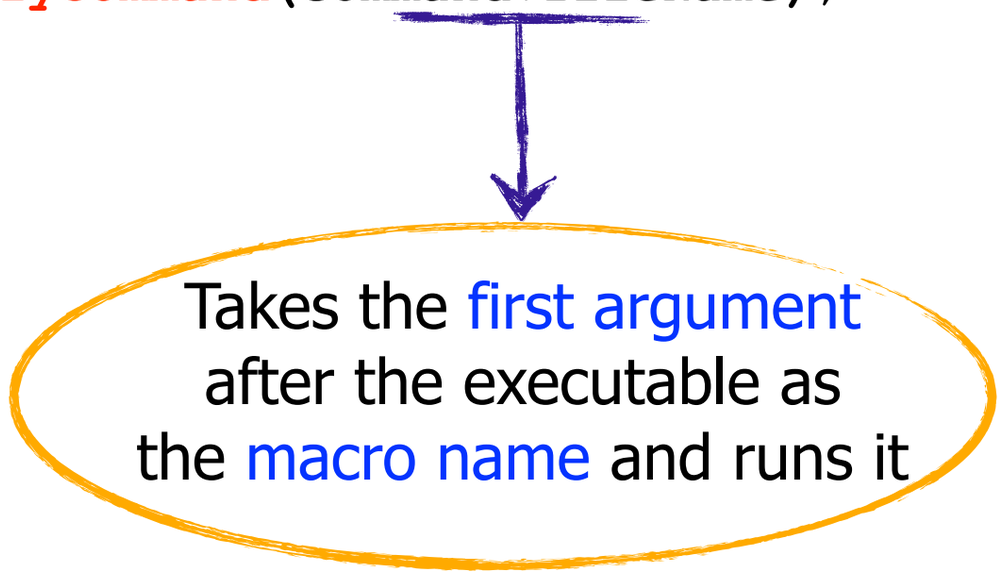
✓commands captured from an interactive session

In batch mode:

- read command from command line

  In the main():

```
G4UImanager* UI = G4UImanager::GetUIpointer();
G4String command = "/control/execute";
G4String fileName = argv[1];
UI->ApplyCommand(command+fileName);
```



- Your executable can be run as

```
./myExecutable mymacro.mac
```

- To execute a macro interactively:

```
/control/execute mymacro.mac
```

Interactive mode

- You can decide the interface to use
- All of them must be derived from the abstract class **`G4UIsession`**

In the **`main()`**, according to the computer environments, construct a **`G4UIsession`** concrete class provided by Geant4 and invoke its **`SessionStart()`** method

- Geant4 provides several interfaces for various (G)UI:
  `G4UIterminal`, **`G4UItcsh`**, **`G4UIGAG`**, **`G4UIXm`**, **`G4UIQt`**

For istance:

```
G4UIsession* session=0;

if (argc==1)
 {
    session = new G4UIterminal;
    session->SessionStart();
    delete session;
 }
```

Or (better) use the **`G4UIExecutive`**

The **`G4UIExecutive`** takes care of selecting the most appropriate UI given the system environment

An example of interactive session – let G4UIExecutive choose

For instance: in the `main()`

```
G4UIExecutive* session =
  new G4UIExecutive(argc, argv);
```

← Create an instance of the G4UIExecutive

```
if (argc==1)
 {
   session->SessionStart();
   delete session;
```

If there are no arguments after the executable, starts an interactive session

Start the session gives the prompt

Don’t forget to delete it

Built-in user commands

Geant4 provides a number of general-purpose user interface commands which can be used:

- ✓ interactively via a (G)UI

  For istance:

```
/run/setCut [value] [unit]
/run/beamOn 100
```

- ✓ in a macro file

Within C++ code using the `ApplyCommand()` method of `G4UImanager`

```
G4UImanager::GetUIpointer()->ApplyCommand("/run/setCut 1 cm");
```

A complete list of built-in commands is available in the Geant4 Application Developers Guide, Chapter 7.1

User-defined commands

If built-in commands are not enough, you can make your own

Geant4 provides several command classes, all derived from **`G4UIcommand`**, according to the type of argument they take

```
G4UIcmdWithoutParameter
G4UIcmdWithABool
G4UIcmdWithADouble
G4UIcmdWithADoubleAndUnit
...
```

User-defined commands

Commands have to be defined in messenger classes, that **inherit from `G4UImessenger abstract class`**

- Define the command in the constructor:

```
G4UIcmdWithADoubleAndUnit* fSizeCmd =
   new G4UIcmdWithADoubleAndUnit
   ("/window/size",this);

fSizeCmd->SetGuidance("Size of the window");
fSizeCmd->SetDefaultUnit("cm");
fSizeCmd->SetUnitCandidates("cm mm");
```

- Delete the command in the destructor

User-defined commands

- Define the action of the command in the **`SetNewValue()`** method of the messenger:

```
void MyMessenger::SetNewValue
   (G4UIcommand* cmd,G4String string)
{
  if (cmd == fSizeCmd)
  {
    G4double value = fSizeCmd
     ->GetNewDoubleValue(string);

    ...->DoSomething(value);
  }
```

Visualization

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 supports several different external visualization systems

Visualizable Objects

Simulation data you may like to see:

- Detectors components
- Geometry: solid, logic and physical volume
- 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 ...

Visualization Attributes

Necessary for visualization, but not included in geometry

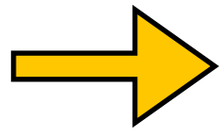
- Colour, visibility, wireframe/solid style, etc

✓A **`G4VisAttributes`** class holds all visualization attributes to be assigned to a visualizable object

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

To set attributes:

```
G4bool visibility = false;
myVisAtt->SetVisibility(visibility);
```



Visualization is skipped

✓A 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 )
```

For instance:

```
G4Color red(1.0, 0.0, 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

The visualization attributes have to be assigned to the visualizable object:

- The class **`G4LogicalVolume`** holds a pointer to the class **`G4VisAttributes`**

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

Polyline

- Defined with a class **`G4Polyline`** defined as a list of **`G4Point3D`** objects polygonal line vertices

➡ A set of successive line segments used to visualize tracking steps, particle trajectories, coordinate axes, any other user-defined polyline

Marker

- Set a mark to an arbitrary 3D position

➡ Usually used to visualize hits of particles

Set marker properties with:

```
SetPosition( const G4Point3D& )
SetWorldSize( G4double real_3d_size )
SetScreenSize( G4double 2d_size_pixel )
```

G4 Visualisation Drivers

- **Visualization drivers** are interfaces of Geant4 to 3D graphics software
- You can select your favorite one(s) depending on your purposes

Some of them work directly from Geant4:

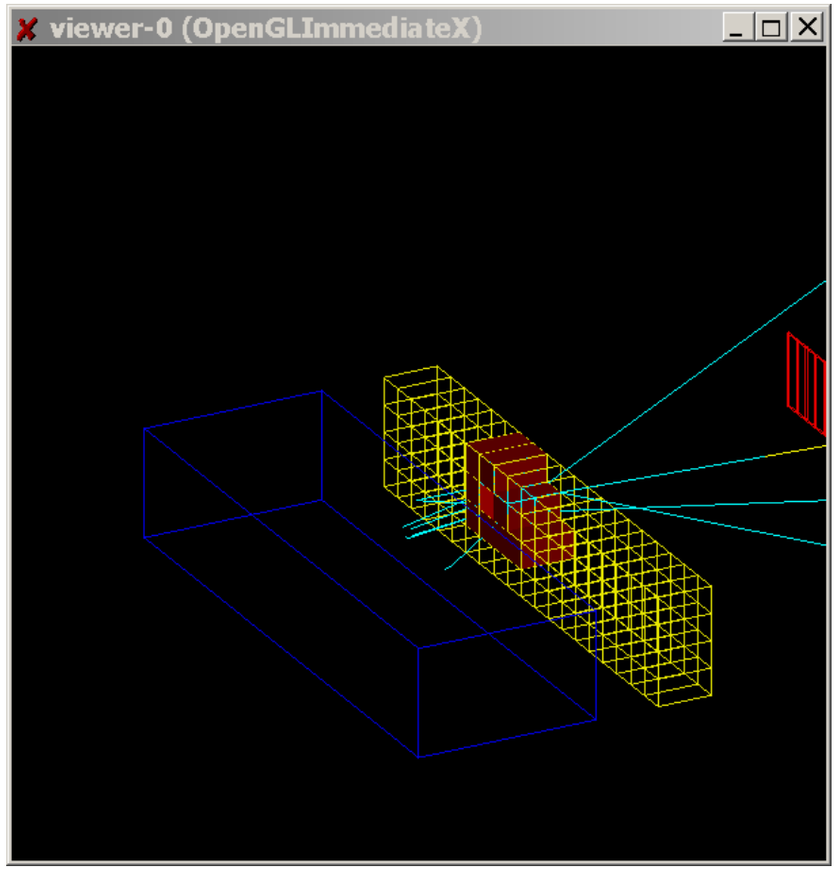
- ✓**OpenGL**
- ✓**Qt**
- **OpenInventor**
- **RayTracer**
- **ASCIITree**
- **Wt Experimental, use with caution**

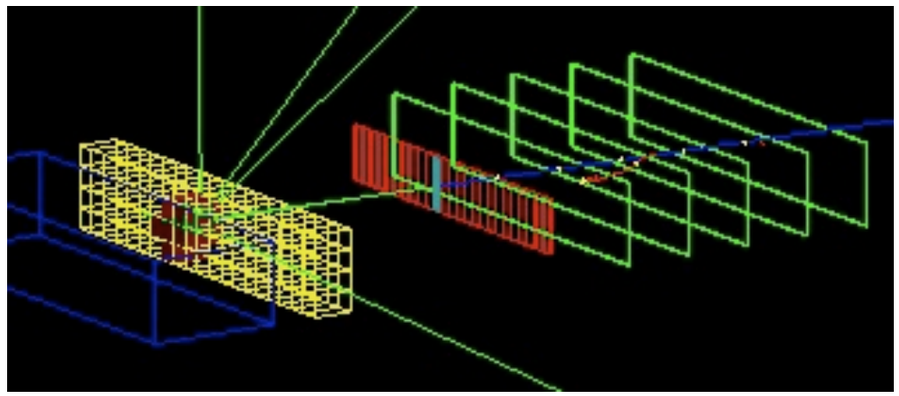
For other, Geant4 will dump a file in a specific format that you can later visualize

- **HepRep**
- **DAWN**
- **VRML**
- **gMocren**

OpenGL

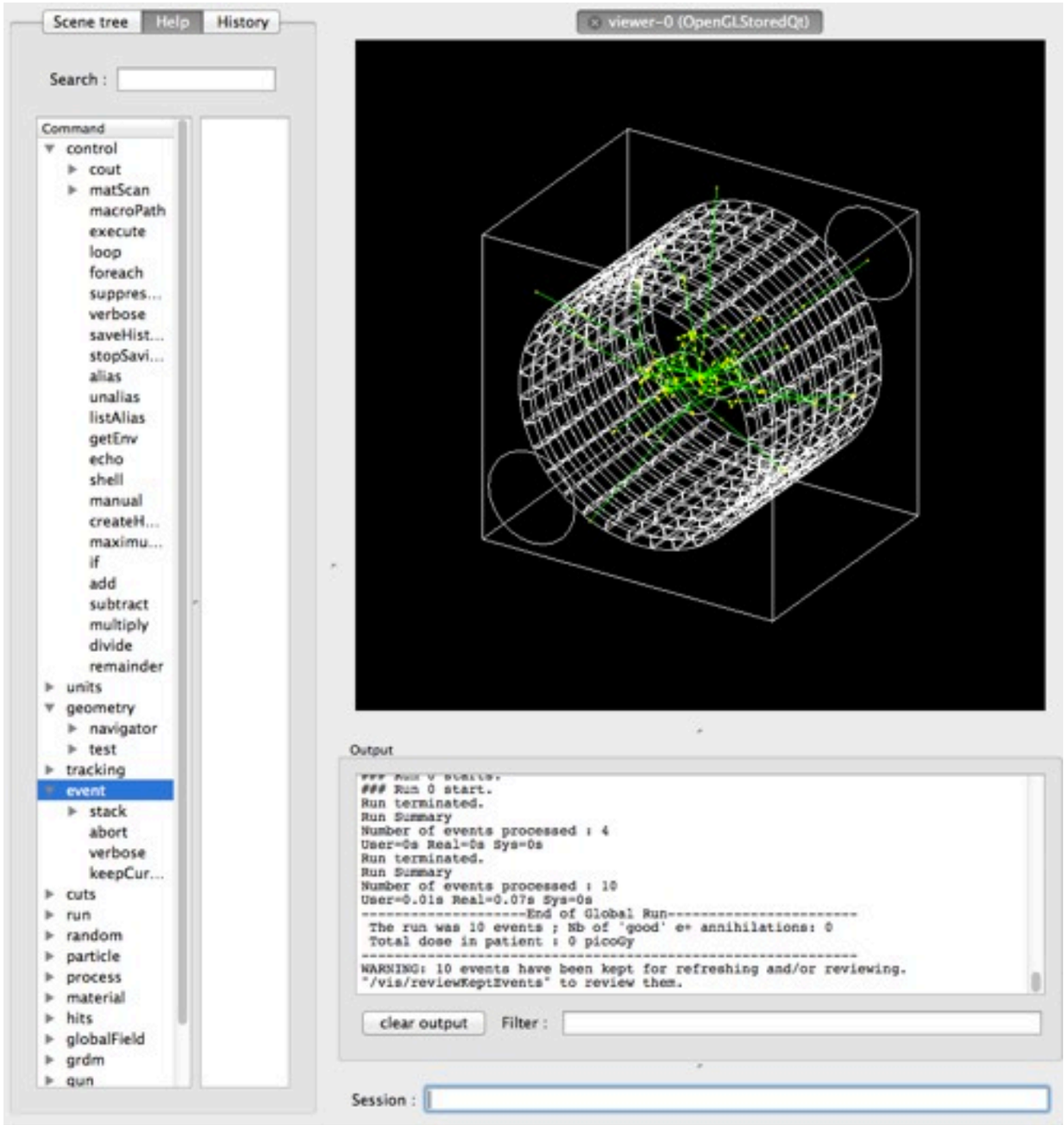
- ➡View directly from Geant4
- ➡Requires additional GL libraries (already included on most Linux and Windows systems)
- ➡Rendered, photorealistic image with some interactive features zoom, rotate, translate
- ➡Fast response
- ➡Print to vector or pixel graphics
- ➡Movies





Qt Libraries

- ➡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

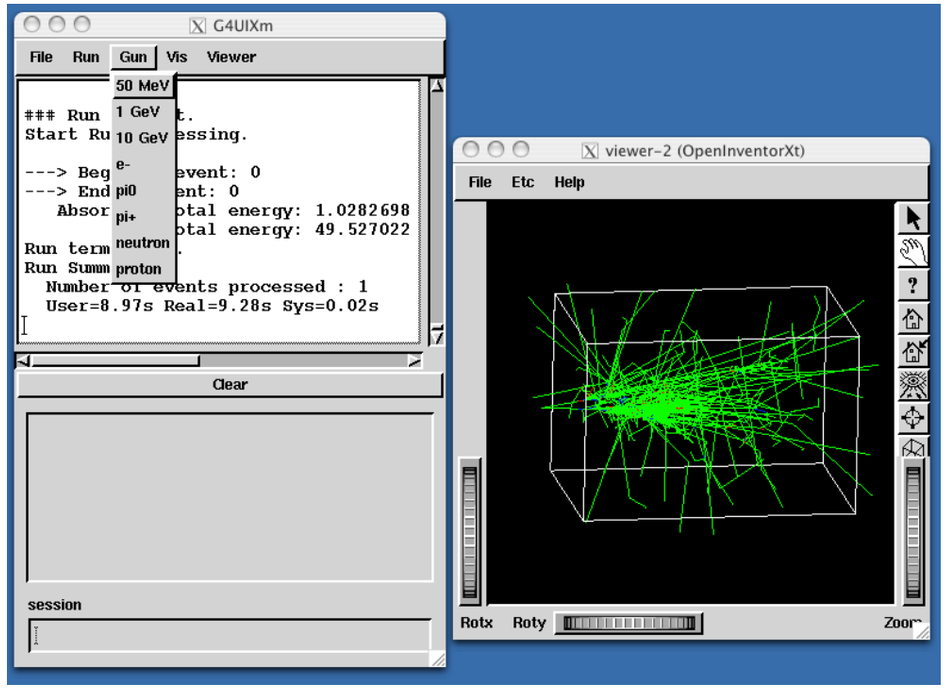


OpenInventor

- Control from the OpenInventor GUI (view direclty from Geant4)

- Requires addition of OpenInventor libs (freely available for most Linux and Windows systems)

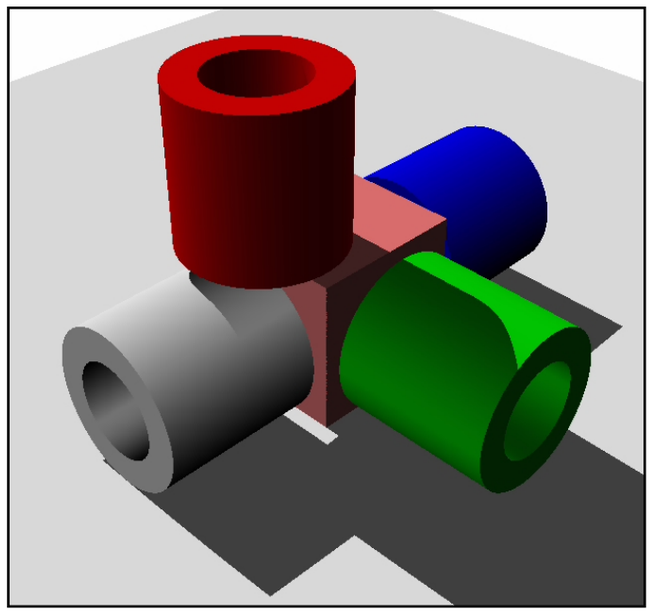
- Rendered, photorealistic image
- Many interactive features



RayTracer

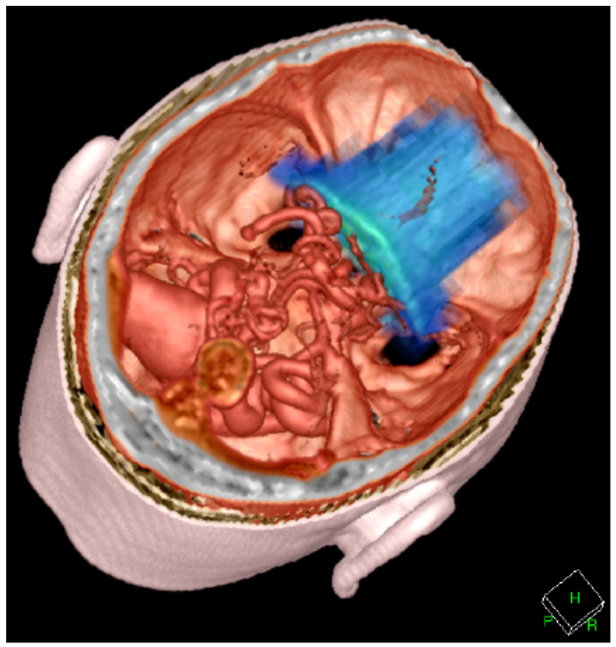
- Create a jpeg file (and with RayTracerX option, also draws to x window)

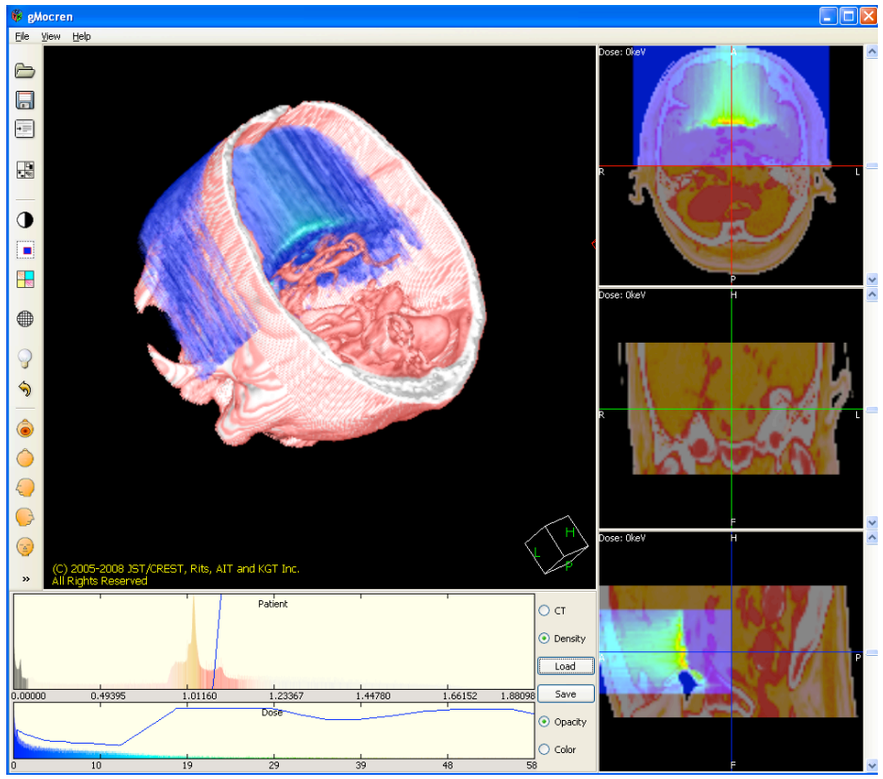
- **Can show geometry but not trajectories**
- **Can render any geometry that Geant4 can handle** (such as Boolean solids)
  no other Vis driver can handle every case
- Supports shadows, transparency and mirrored surfaces



gMocren

- ✓Create a file to be viewed in the gMocren browser.
- ✓can show volume data such as Geant4 dose distrubutions overlaid with scoring grids, trajectories and detector geometry
- ✓Can overlay patient scan data (from DICOM) with Geant4 geometry, trajectories and dose





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
```

Visualization commands

```
/vis/ogl/              G4OpenGLViewer 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/touchable/        Operations on touchables.
/vis/viewer/           Operations on Geant4 viewers.
```

```
/vis/viewer/set/viewpointThetaPhi 70 20
  /vis/viewer/zoom <scale_factor>
   /vis/viewer/set/style wireframe
```

`/vis/drawVolume` → **and registers it**

`/vis/specify logicLAr` → **set specific logical volume for visualization**

`/vis/viewer/flush` → **close visualization**

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 :

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

Thanks for your attention