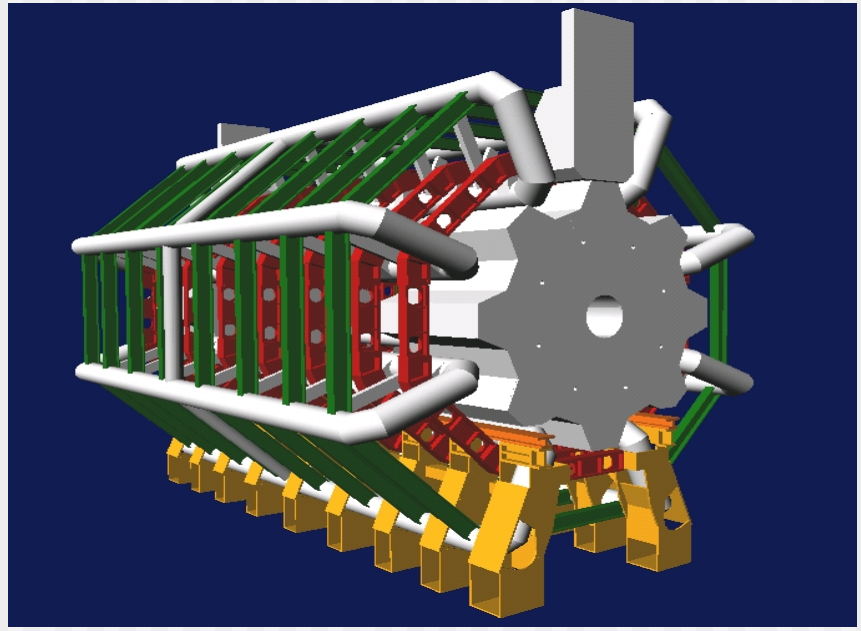
Geant 4

Visualisation and (G)UI

http://geant4.cern.ch

PART I

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 you may like to see:
  - 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 (macro or interactive) or by writing C++ source codes of user-action classes

2. Visualisable Objects (2)

- You 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 ...

3. Visualization Attributes

- Necessary for visualization, but not included in geometrical information
  - 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 (e.g. a logical volume) with its method `SetVisAttributes()`:

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

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, visualization is skipped for objects for which this set of visualization attributes is assigned.
    The default value of visibility is **`true`**.

3.3 Colour (1)

- Class **G4VisAttributes** holds its colour entry as an instance of class **G4Colour**
- **G4Colour** 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);
```

3.3 Colour (2)

- A colour can be set in a **`G4VisAtttributes`** object via the functions of **`G4VisAttributes`**
  - `SetColour(const G4Colour& colour)`
  - `SetColour( G4double r ,`
    `G4double g ,`
    `G4double b)`

3.4 Assigning **G4VisAttributes** to a logical volume

- Class **G4LogicalVolume** holds a pointer of **G4VisAttributes**
- Access functions of **G4LogicalVolume**
  - **SetVisAttributes** (const G4VisAttributes* pva)
- For instance:

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

logical volume

4. Polyline and Marker

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

4.1 Polyline

- A set of successive line segments
- Defined with a class **`G4Polyline`**
- Used to visualize tracking steps, particle trajectories, coordinate axes, etc
- **`G4Polyline`** is defined as a list of **`G4Point3D`** objects. Elements of the list define vertex positions of a polyline.

Example C++ code for polyline:

```
//-- 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 );
  G4VisAttributes att ( red );
  x_axis.SetVisAttributes( att );

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

Create a polyline

Define the points of the polyline

Set the visualization attributes of the polyline

4.2 Marker (1)

- Set a mark to an arbitrary 3D position
- Usually used to visualize hits of particles
- Designed as a 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 )`

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)`

Example C++ code for marker:

```
G4Point3D position(0,0,0);
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.);
G4VisAttributes attribs(colour);
 // Define a red visualization attribute
circle.SetVisAttributes(attribs);
 // Assign the red end of C++ source code
```

Create a circle in a given position

Set diameter and style

Set colour and vis attributes

5. Visualisation Drivers

- **Visualization drivers** are interfaces of Geant4 to 3D graphics software
- You can select your favorite one(s) depending on your purposes such as
  - Demo
  - Preparing precise figures for journal papers
  - Publication of results on Web
  - Debugging geometry
  - Etc.

5.1 Available Graphics Software

- Geant4 provides several visualization drivers tailored to different purposes:
  - **DAWN** : Technical High-quality PostScript output
  - **OPACS**: Interactivity, unified GUI
  - **OpenGL**: Quick and flexible visualisation
  - **OpenInventor**: Interactivity, virtual reality, etc
  - **RayTracer** : Photo-realistic rendering
  - **VRML**: Interactivity, 3D graphics on Web
  - ...

5.2 Available Visualisation Drivers

- DAWN → Fukui Renderer DAWN
- OPENGLX → OpenGL with Xlib
- HepRep → HepRep graphics
- OIX → OpenInventor with Xlib
- RayTracer → JPEG files
- VRML → VRML 1.0/2.0
- etc

5.3 How to Use Visualization Drivers

- Visualization should be switched on using the variable G4VIS_USE
- You can select/use visualisation driver(s) by setting environmental variables before compilation, according to what is installed on your computer:
  - `setenv G4VIS_USE_DRIVERNAME 1`
- Example (DAWN, OpenGLXlib, and VRML drivers):
  - `setenv G4VIS_USE_DAWN 1`
  - `setenv G4VIS_USE_OPENGLX 1`
  - `setenv G4VIS_USE_VRML 1`

6. **main()** Function (1)

To have a Geant4 executable able to **handle visualization**, you have two choices:

- Instantiate and initialize your own Visualization Manager in the `main()`. It must inherit by `G4VisManager` and implement the `void RegisterGraphicSystem()` method
- (Easiest) To use the ready-for-the-use **`G4VisExecutive`** class available in Geant4. It must be instantiated and initialized in the **`main()`** program (→ see next slide)

6. **main()** Function (2)

```
//----- C++ source codes: Instantiation and
   initialization of G4VisManager in main()

#include "G4VisExecutive.hh“
```

Includes the G4VisExecutive class

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

Instantiate and initialize the Visualization Manager if G4VIS_USE is “true”

```
#ifdef G4VIS_USE
delete visManager;
#endif
```

Don’t forget to **delete** the pointer to G4VisExecutive at the end of main()

7. Visualisation commands

- There are some frequently-used built-in visualization commands in Geant4, that you may like to try
- Geant4 executable in this tutorial is compiled incorporating DAWN, OpenGL and VRML drivers
  - `setenv G4VIS_USE_DAWN    1`
  - `setenv G4VIS_USE_OPENGLX 1`
  - `setenv G4VIS_USE_VRML    1`

7.1 An example of commands to visualize a detector

```
/vis/open OGLIX
# or /vis/open DAWNFILE
```
create a scene handler and a viewer

```
/vis/viewer/reset
/vis/viewer/viewpointThetaPhi 70 20
/vis/viewer/set/style wireframe
```
set vis options

`/vis/drawVolume` set the detector geometry as object to visualize, and registers it

`/vis/viewer/flush` close visualization

These commands can be given interactively or executed via macro (e.g. vis1.mac of N03)

7.2 An Example of Visualizing Events

`/tracking/storeTrajectory` } Store particle trajectories for visualization

`/vis/open DAWNFILE` } Scene handler and viewer for DAWN

… } Optional settings (viewpoint, axes, etc.)

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

`/vis/scene/add/volume`
`/vis/scene/add/trajectories` } Adds world volumes and trajectories to the scene

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

Again, commands can executed via macro (e.g. vis2.mac of example N03) or interactively

7.3 /vis/open command

- Command
  - Idle> /vis/open <driver_tag_name>
  - The “driver_tag_name” is the driver’s name
- 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

7.4 /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
    ```

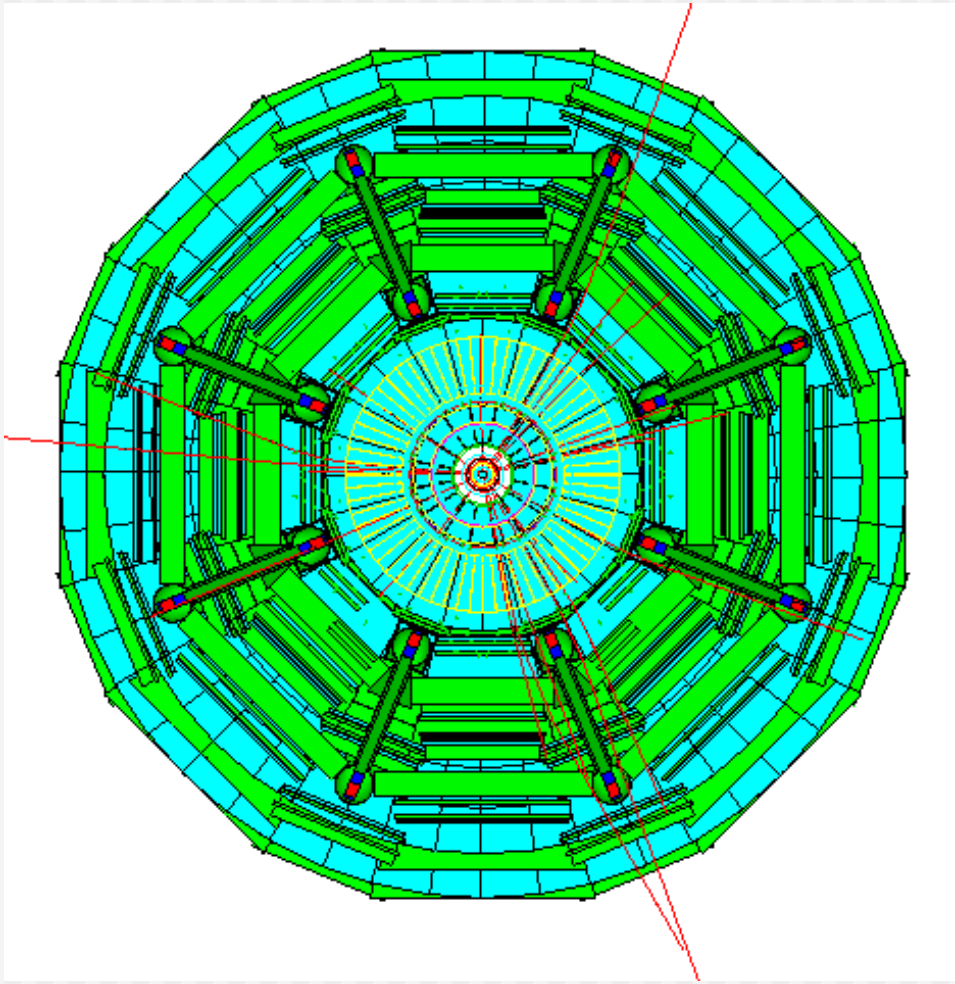
7.5 /vis/drawVolume and /vis/viewer/flush commands

- Commands:
  - `Idle> /vis/drawVolume <physical-volume-name>`
    `(Default: world)`
    `Idle> /vis/viewer/flush`
  - Note that `/vis/viewer/flush` should be executed to declare end of visualisation.
  - You can draw a specific volume (rather than the full geometry)
- You can add visualization commands of, say, coordinate axes. For example,
  `Idle> /vis/scene/add/axes <Ox> <Oy> <Oz> <length> <unit>`

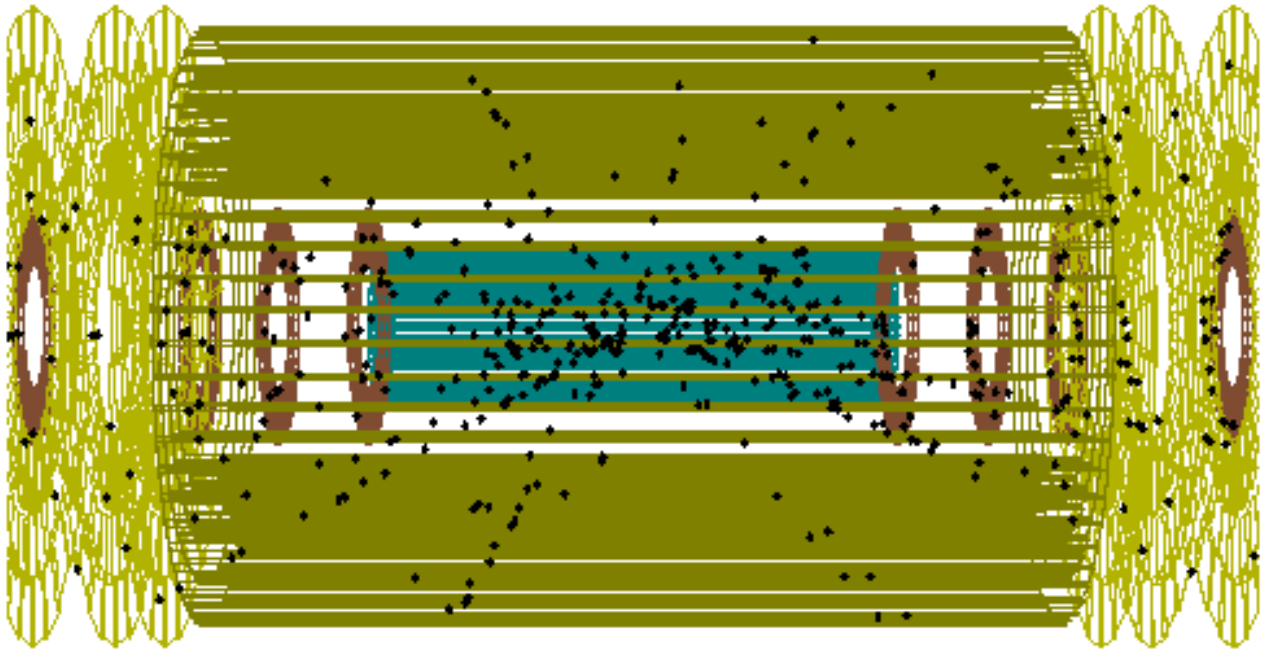
7.6 Commands to Visualize Events

- Commands
  - `Idle> /tracking/storeTrajectory 1`
    `Idle> /vis/scene/add/trajectories`
    `Idle> /run/beamOn <number_of_events>`
- Action:
  - Automatic visualization of events

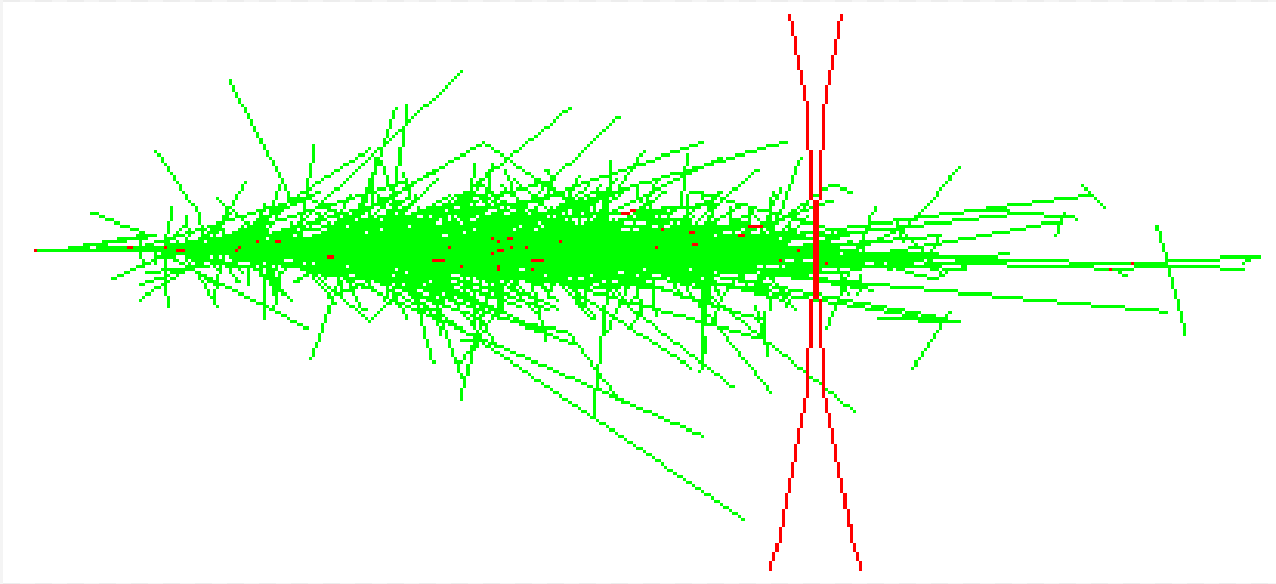
Sample Visualization (1)



Sample Visualization (2)



Sample Visualization (3)



8. Visualisation from C++ code

- It is also possible to hard-code visualization commands in the C++ code (personally, I'd discourage it)
- You can describe the visualization commands in C++ codes via the **`ApplyCommand()`** method of the UI manager, as for any other command:

```
G4UImanager::GetUIpointer()
->ApplyCommand("/vis/…");
```

- Alternatively, you can use **`Draw()`** methods of visualizable classes

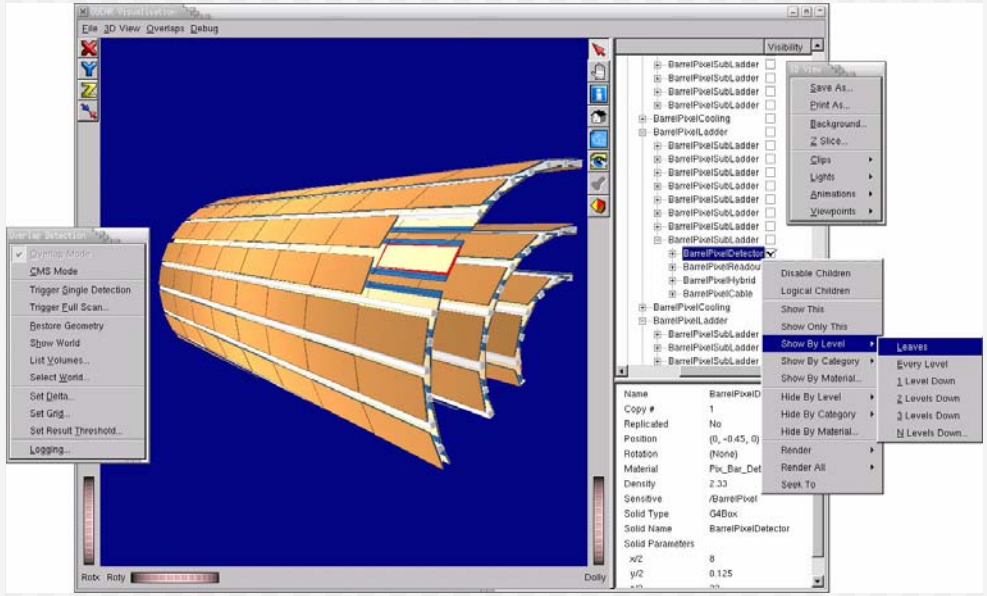
```
myPolyline->Draw();
```

9. Information

- Geant4 User Guide (and source codes)
- README file:
  - **`geant4/source/visualisation/README`**
- On-line documentation on Geant4 visualisation
  - http://cern.ch/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/Visualization

PART II

Geant4 (G)UI



Steering the simulation (1)

- A Geant4 simulation can be **steered in three 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!)
  - batch session (via a ASCII macro)
  - commands captured from an interactive session

Steering the simulation (2)

- Setting up batch mode (namely, read commands from a macro file) in the main()

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

takes the first argument after the executable as the macro name and runs it

- Your executable can be run as

```
myExecutable mymacro.mac
```

- To execute a macro interactively:

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

Steering the simulation (3)

- Setting up **interactive mode** is also easy – but there are many choices of interface
  - All of them must be derived from the abstract class **G4UIsession**
  - Geant4 provides several implementations
- In the **main()**, according to the computer environments, construct a **G4UIsession** concrete class provided by Geant4 and invoke its **SessionStart()** method

An exemple of interactive session

- For instance: in the `main()`

```
G4UIsession* session=0;
```

Create a (null) pointer to the base session class

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

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

Define the session as a dumb terminal, and starts it

Don't forget to delete it

Select (G)UI

- Geant4 provides several interfaces for various (G)UI:
  - **G4UIterminal**: C-shell like character terminal
  - **G4UItcsh**: tcsh-like character terminal with command completion, history, etc
  - **G4UIGAG**: Java based graphic UI (GUI)
  - **G4UIXm**: Motif-based GUI, command completion
- Define and invoke them like G4UIterminal

```
session = new G4UIGAG();
session->StartSession();
```

- Note for **G4UItcsh**, it must be defined as

```
session = new G4UIterminal (new G4UItcsh);
```

Environmental variables

- Users can select and plug in (G)UI by setting *environmental variables* before compilation, similar to what seen for visualization drivers
  - `setenv G4UI_USE_GUINAME`
- Example:
  - `setenv G4UI_USE_TERMINAL  1 (default)`
  - `setenv G4UI_USE_GAG       1`
  - `setenv G4UI_USE_XM        1`

User interface choices

- `G4UIterminal` – C-shell-like character terminal
  - runs on all Geant4-supported platforms
- `G4UItcsh` – tcsh-like character terminal with command completion, history, etc.
  - runs only on Solaris and Linux
- `G4UIXm`, `G4UIXaw`, `G4UIXWin32` – G4UIterminal implemented over Motif, Athena and WIN32 libraries
  - runs on Unix/linux and Windows, respectively
- `G4UIGAG` – Java-based GUI
  - runs on all Geant4 platforms

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 for analysis
  - http://www.lal.in2p3.fr/OpenScientist

Build-it user commands

- Geant4 provides a number of general-purpose user interface commands which can be used:
  - interactively via a (G)UI
    ```
    Idle> /run/setCut [value] [unit]
    ```
  - 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 (1)

- If built-in commands are not enough, you can make your own (e.g. change at run-time parameters of primary generator, etc.)
- Geant4 provides several command classes, all derived from **`G4UIcommand`**, according to the type of argument they take
  - **`G4UIcmdWithoutParameter`**
  - **`G4UIcmdWithABool`**
  - **`G4UIcmdWithADouble`**
  - **`G4UIcmdWithADoubleAndUnit`**
  - **`...`**

User-defined commands (2)

- Commands have to be defined in messenger classes, that **inherit from G4UImessenger**
- Define the command in the constructor:

```
G4UIcmdWithADoubleAndUnit* fThetaCmd =
  new G4UIcmdWithADoubleAndUnit
   ("/prim/angle",this);
```

Command taking as argument a double and a unit, called /prim/angle

```
fThetaCmd->SetGuidance(“Opening angle of the source");
fThetaCmd->SetDefaultUnit("deg");
fThetaCmd->SetUnitCandidates(“deg rad”);
```

Sets guidance, default unit, etc.

- Delete the command in the destructor

User-defined commands (3)

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

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

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

Retrieve a G4double value from the **(string)** argument given to the command

Use the value in the way it is needed (e.g. pass it to other classes: opening angle for primary generator)

PART III

Summary

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
- Interactive sessions where user can give commands by keyboard can be used (from dumb terminals to graphic interfaces)
- A number of general-purpose commands are provided by Geant4, but **users can define more**, according to their needs → flexibility!

PART IV

Backup

7.1 Scene, Scene Hander, Viewer

- In order to use visualization commands, let's define "scene", "scene handler", and "viewer":
- **Scene**: *A set of visualizable 3D data (e.g. a geometry, a polyline, etc.)*
- **Scene handler**: *Computer Graphics data modeler, which uses raw data in a scene*
- **Viewer**: *Image generator*
- Each scene handler is assigned to a scene
- Each viewer is assigned to a scene handler
- "visualisation driver" = "scene_handler" + "viewer"

7.2 Logical steps of Visualization

- Step 1: Create a scene handler and a viewer
- Step 2: Create an empty scene
- Step 3: Set camera parameters, drawing style, etc. [Optional]
- Step 4: Add 3D data to the created scene
- Step 5: Attach the current scene handler to the current scene
- Step 6: Make the viewer execute visualisation
- Step 7: Declare the end of visualization

You can do all that **interactively** or **using a macro** (see next slide)

7.5 /vis/viewer/set/style command

- Command
  - Idle> /vis/viewer/set/style
    <style_name>
  - The “style_name” can be “wireframe” or “surface”