Utilizzo del toolkit di simulazione Geant4

Laboratori Nazionali del Gran Sasso

10 Novembre 2010

Geant4 Visualization
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
- ...
Visualisable Objects

• 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

• 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 ...
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)

• 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.
Colour

• 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);
    ```

• A **colour** can be **set** in a **G4VisAttributes** object via the functions of **G4VisAttributes**
  
  - `SetColour(const G4Colour& colour)`
  
  - `SetColour(G4double r, G4double g, G4double b)`
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);
  ```
• **Polyline** and **marker** are defined in the `graphics_reps` category
• They are available to model 3D scenes for visualization
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.

```cpp
//-- 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 );
```
Marker

- 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 )`

- **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:

```cpp
G4Point3D position(0,0,0); } Create a circle in a given position
G4Circle circle(position); { Set diameter and style

    // 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.); Set colour and vis attributes
G4VisAttributes attribs(colour);
    // Define a red visualization attribute
    circle.SetVisAttributes(attribs);
    // Assign the red end of C++ source code
```
**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.
Available visualization drivers

- Geant4 provides several visualization drivers tailored to different purposes:
  - OpenGL
  - Qt
  - OpenInventor
  - HepRep
  - DAWN
  - VRML
  - RayTracer
  - ASCIIITree
  - gMocren

A quick overview ...
OpenGL

- Control directly from Geant4
- Uses GL libraries that are already included on most Linux and Windows systems
- Rendered, photorealistic image with some interactive features
- zoom, rotate, translate
- Fast response (can usually exploit full potential of graphics hardware)
- Print as pixel graphics or vector EPS
- Movies
Qt

- View directly from Geant4
- Addition of Qt and GL libraries 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
- Requires addition of OpenInventor libraries (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 (can usually exploit full potential of graphics hardware)
- Expanded printing ability (vector and pixel graphics)
• Create a file to view in the HepRApp HepRep 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 view 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, trajetories 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 m^3
  - Mass of tree to unlimited depth is 22260.5 kg
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`
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)
//----- C++ source codes: Instantiation and initialization of G4VisManager in main()

#include "G4VisExecutive.hh"

// Instantiation and initialization of the Visualization Manager
#include "G4VisExecutive.hh" } Includes the G4VisExecutive class

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

#include "G4VisExecutive.hh" } Includes the G4VisExecutive class

// Instantiation and initialization of the Visualization Manager
#include "G4VisExecutive.hh" } Includes the G4VisExecutive class

#ifndef G4VIS_USE
G4VisManager* visManager = new G4VisExecutive;
visManager -> initialize();
#endif

#ifndef G4VIS_USE
delete visManager;
#endif

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

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

- Sub-directories (from help):
  - 1) /vis/ASCIITree/  Commands for ASCIITree control.
  - 2) /vis/GAGTree/   Commands for GAGTree control.
  - 3) /vis/heprep/    HepRep commands.
  - 4) /vis/rayTracer/ RayTracer commands.
  - 5) /vis/scene/     Operations on Geant4 scenes.
  - 6) /vis/sceneHandler/ Operations on Geant4 scene handlers.
  - 7) /vis/viewer/    Operations on Geant4 viewers.
- Commands:
  - 8) enable * Enables/disables visualization system.
  - 9) disable * Disables visualization system.
  - 10) verbose * Simple graded message scheme -
        digit or string (1st character defines):
  - 11) drawTree * (DTREE) Creates a scene consisting of this physical volume and produces a
        representation of the geometry hierarchy.
  - 12) drawView * Draw view from this angle, etc.
  - 13) drawVolume * Creates a scene consisting of this
        physical volume and asks the current viewer to draw it.
  - 14) open * Creates a scene handler ready for drawing.
  - 15) specify * Draws logical volume with Boolean components, voxels and readout geometry.

Guidance is hierarchical, providing full detail on all commands
Commands to visualize detectors

/vis/open OGLIX  \{ create a scene handler and a viewer \}
# or /vis/open DAWNFILE

/vis/viewer/reset  \{ set vis options \}
/vis/viewer/viewpointThetaPhi 70 20
/vis/viewer/set/style wireframe

/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)
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
/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
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
  ```
/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>
Trajectory Filtering

• Simplest example:
  – `/vis/filtering/trajectories/create/particleFilter`
  – `/vis/filtering/trajectories/particleFilter-0/add e-`
  – will cause everything except electrons to be filtered out

• You can chain multiple filters
  – e.g., filter out gammas
  – and filter out particles with momentum less than 100 MeV

• Two modes of filtering:
  – Important issue when working with those visualization drivers that allow you to turn visibility on and off from the vis application (HepRApp or OpenInventor)
  – One mode has rejected trajectories marked invisible but still sent to vis app
    • user can toggle them back to visible from within the vis app
    • but if there is a very large number of these invisible trajectories, application may be slowed down and files may be very large
  – Other mode has rejected trajectories entirely omitted
    • user cannot toggle them back to visible from within the vis app
    • but application stays fast and files stay small
Filtering example

/vis/modeling/trajectories/drawByAttribute-0/setAttribute IMag
/vis/modeling/trajectories/drawByAttribute-0/addInterval interval1 0.0 keV 2.5 MeV
/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
/vis/modeling/trajectories/drawByAttribute-0/interval1/setLineColourRGBA 0.8 0.8 0.8 1
/vis/modeling/trajectories/drawByAttribute-0/interval2/setLineColourRGBA 0.23 0.41 0.41 1
/vis/modeling/trajectories/drawByAttribute-0/interval3/setLineColourRGBA 0.1 0.1 0.1 1
/vis/modeling/trajectories/drawByAttribute-0/interval4/setLineColourRGBA 0.1 0.1 0.1 1
/vis/modeling/trajectories/drawByAttribute-0/interval5/setLineColourRGBA 0.1 0.3 0.1
/vis/modeling/trajectories/drawByAttribute-0/interval6/setLineColourRGBA 0.1 0.1 0.1

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

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