Geant4 Visualization

Getting Started

Jacek Generowicz
CERN IT/API

Jacek.Generowicz@cern.ch

Geant4 Users' Workshop, CERN, 11 Nov 2002

Requirements

Geant4 visualization needs to be able to satisfy a variety of needs:

- Very quick response to survey successive events
- High-quality outputs for presentation and publication
- Impressive special effects for demonstration
- Flexible camera control for debugging geometry of detector components and physics
- Interactive picking of graphical objects for attribute editing or feedback to the associated data
- Highlighting collisions of physical volumes visually
- Remote visualization via the Internet

Geant4 Visualization is able to respond to all these requirements...
... but difficult to do with a single bulit-in visualizer.

Abstract Visualization Interface

Solution: Geant4 provides an abstract interface to be used by different kinds of graphcis systems.

- DAWNFILE
- DAWN-NetworkFukui
- HepRepFile
- OPACS
- OpenGL-Xlib
- OpenGL-Motif
- OpenGL-Win32
- OpenInventor-X
- OpenInventor-Win32
- RayTracer
- VRMLFILE
- VRML-Network

(The red ones link against external libraries)

In the hands-on examples you will use OpenGL-Xlib, and VRMLFILE.

Compiling-in visualization code

When Geant4 is being compiled, the preprocessor variables G4VIS_BUILD_* are used to determine which of the systems will be made available.

The configure script

```
  .../geant4/Configure -install
```

asks which drivers should be supported, and sets the variables appropriately.

When user code is being compiled, the preprocessor variables G4VIS_USE_* are used to determine which systems shoud be used. The values of these are taken from similarly named environment variables.

NB. The preprocessor variable G4VIS_USE is defined if the environment variable G4VIS_NONE is NOT set.

Clearly, you will not be able to use systems which have not been compiled into your Geant4 libraries.

Visualizing (with OpenGL)

You must ensure that the environment variable OGLHOME points at the directory containing the OpenGL libraries. Either libMesa* or libGL* will do.

You must write your own Visualization Manager.
It must be derived from G4VisManager
It must implement the method
RegisterGraphicsSystems()

Start by using MyVisManager, a sample implementation which can be found in the directory visualization/management/include

Note how the preprocessor variables are used in this class:

```
#ifdef G4VIS_USE_OPENGLX
 RegisterGraphicsSystem (new G4OpenGLImmediateX);
 RegisterGraphicsSystem (new G4OpenGLStoredX);
#endif
```

Instantiating the VisManager

```
int main( ... ) {
  ...
#ifdef G4VIS_USE
  // Your Visualization Manager
  #include "MyVisManager.hh"
#endif
  ...
#ifdef G4VIS_USE
  // Instantiation and initialization of the
  // Visualization Manager
  G4VisManager* visManager = new MyVisManager;
  visManager -> initialize ();
#endif
  ...
#ifdef G4VIS_USE
  delete visManager;
#endif
  ...
}
```

Scenes, Handlers and Viewers

A scene is a set of visualizable objects, such as detector components, hits, trajectories, axes, etc.

A scene handler is a graphics-data modeler, which processes raw data in a scene for later visualization.

A viewer generates output based on data processed by a scene handler.

Visualization steps

The typical steps of performing Geant4 visualization are:

- Create a scene handler and a viewer.
  - /vis/sceneHandler/create OGLIX
  - /vis/viewer/create (default: current scene handler)
- Create an empty scene.
  - /vis/scene/create (this scene becomes current)
- Add raw 3D data to the created scene.
  - /vis/scene/add/volume (default: world)
  - /vis/scene/add/axes
- Attach the current scene handler to the current scene.
  - /vis/sceneHandler/attach (default: current scene)
- Set camera parameters, drawing style (wireframe/surface), etc.
  - /vis/viewer/set/viewpointThetaPhi 30 30
- Make the viewer execute visualization.
- Declare the end of visualization for flushing.
  - /vis/viewer/flush

Full description of all vis commands can be found in /geant4/source/visualization/README.built_in_commands

Miscellaneous

Trajectory storage is switched on with the command

- /tracking/storeTrajectory 1

Be sure to have a look at the macro files (.mac) in the examples directories.

In particular, look at
geant4/examples/novice/N03/visTutor
which contains didactic examples of visualization macro files.

Visualization Attributes

Visualization attributes are data associated with visualizable objects, which are only relevant to visualization. These are data which do not play any part in the simulation itself.

For example:

- colour
- visibility
- line style

Visualization attributes may be set for specific objects ...

... otherwise, default values will be applied.

Attributes are held in an instance of the class G4VisAttributes which is defined in the category graphics_reps.

Setting Attributes

Let’s look at how some attributes are set.

Visibility has only 2 states: visible/invisible.

Set it using the method

```
void G4VisAttributes::SetVisibility(G4bool visibility);
```

Colour has an (almost) infinite number of states.

Use the G4Color class.

Its instances store 4 values repersenting the red, green, blue and alpha components of the colour. Each value should be in the range [0,1].

All values default to 1. (Some drivers ignore alpha.)

Two spellings of the class name are available:

```
  G4Colour red(1.0, 0.0, 0.0);
  G4Color blue(0.0, 0.0, 1.0);
```

Set it using the methods

```
 void G4VisAttributes::SetColor ( const G4Color  color );
 void G4VisAttributes::SetColour( const G4Colour colour );
```

Assigning Attributes

```
// Instantiate a logical volume
myTargetLog = new G4LogicalVolume( myTargetTube, BGO, "TLog", 0, 0, 0 );
// Instantiate vis attributes, make the colour cyan immediately
G4VisAttributes* calTubeVisAtt = new G4VisAttributes( G4Colour(0,1,1) );
// Use wireframe style
calTubeVisAtt -> SetForceWireframe( true );
// Assign the attributes to your volume
myTargetLog -> SetVisAttributes( calTubeVisAtt );
```

The lifetime of the vis attributes must be at least as long as the objects to which they are assigned.

It is the user’s responsibility to ensure this, and to delete them when they are no longer needed.

Quickstart Summary

- Make sure your OpenGL libraries have been compiled in.
- Make sure the appropriate environment variables point to the libraries.
- Make sure to create scenes, handlers and viewers. (Consider using the compound command /vis/open).
- Make sure to flush the viewer.
- Go forth and visualize ...