Visualization in GEANT4

John Allison  
*University of Manchester, UK*

Abstract

The Visualization System of GEANT4 is an interface between the simulation kernel and the graphics system. In itself it defines only the functionality required by the kernel and the GEANT4 user. It contains an abstract interface to which an installer may fit his or her favourite graphics system; concrete interfaces to some graphics systems come ready-to-use with the GEANT4 toolkit. Commands to the graphics system(s) are channelled through or initiated by the Visualization Manager. There exist basic facilities for debugging, development and publicity which hopefully can evolve as effort accumulates and graphics systems become more powerful. This document describes those facilities and is intended to act as a GEANT4 Visualization Primer.

Keywords: GEANT4, Detector Simulation, Scientific Visualization

1 Introduction

Visualization is an essential part of a detector simulation program. In order to gain confidence in the geometrical model, it is essential to produce graphical representations of the geometrical hierarchy and to draw views and sections of the detector. To gain confidence in the physics processes that are being simulated, it is essential to superimpose the “steps” and the “hits” which occur in the simulation. The Visualization System also helps to evolve detector design. Not least, visualization is a communication and publicity tool.

Past versions of GEANT, notably GEANT3, currently existing in version 3.21 (which will probably be the last), offered facilities for the user to check his or her work by drawing individual detector components, sub-detector assemblies and whole detectors. It was possible to eliminate hidden lines and shade surfaces, but the predominant style was wireframe and sections (cuts). For example, it was common to superimpose projections of steps on a previously drawn section, as a way of confirming the correct behaviour of the program and appreciating the processes which were going on.

The above basic facilities are also provided in GEANT4. In addition, more sophisticated facilities are provided, for example, interactive cutaways, which rely on the greater computer system performance available today. We hope the Visualization Interface is flexible enough to accommodate new facilities and new graphics systems as they become available.

In designing the Visualization System, our guide has been the original User Requirements Document; the next section outlines the visualization items and describes how well we have satisfied, or surpassed, them. Subsequent sections outline the design of the Visualization System and the way of communicating with it to obtain drawings on the computer screen or on hard copy. Details are left to the appendices.

2 User Requirements

The specific user requirements for visualization as set out in the requirements document are (to quote):

**UR 6-1.** The detector setup can be visualized in its entirety or in subsections or by individual component.

**UR 6-2.** The individual geometrical entities representing a detector can be visualized and their parameters displayed.

**UR 6-3.** Particle trajectories can be visualized and each step of the particle tracking can be visualized.

**UR 6-4.** The detector response in sensitive elements of the detector can be visualized.
The visualization system is capable of navigating the genealogy relationships of information in an event, e.g., highlighting all hits from a given track, all tracks from a given parent track, etc.

The visualization is capable of showing the geometry structure, i.e., the genealogy relationships of the geometry.

There is a built-in visualizer of reasonable sophistication. However, the toolkit can provide a straightforward way of connecting to other visualization systems so that the provider can build a visualizer of his/her choice into the framework. Some specific interfaces are provided.

To these, the visualization working group has added:

1. Unless the user decides otherwise (for example, for performance reasons) the drawings are faithful graphics representation, i.e., there are no unnecessary approximations (e.g., polygons for circles) worse than the device resolution.
2. The user can view several different scenes simultaneously and a given scene from several viewpoints.
3. The Visualization Interface can support more than one graphics system concurrently.
4. ‘Transient’ objects such as steps and tracks can be drawn on an existing view of “permanent” objects (such as detectors) in a transient way (drawn and erased) or accumulated.
5. Remote graphics can be supported, i.e., the graphics renderer can be spawned and run on a computer different to the one running the GEANT4 kernel.
6. Graphics actions (e.g., mouse movement to produce camera movements) should be possible in parallel with other GEANT4 actions or commands.
7. The user can draw sections or cutaways of the detector.

For the alpha release, not all the above requirements will be implemented. There will no “picking” or anything which requires feedback from the graphics system to the GEANT4 Visualization Manager or kernel. On the other hand, the alpha release will offer multiple view manipulation, putting it ahead of GEANT3 in that respect.

We have a basic framework which supports several graphics systems. They are:

- The Fukui Renderer (DAWN) [1], for high quality graphics and hardcopy (PostScript).
- OPACS [2], an X-Windows-based system in the public domain.
- OpenGL.
- Open Inventor.

The last two require commercial licences, for which CERN has a special deal [3]. Ray tracing, using GEANT4’s own tracking algorithms, will be added later.

3 Philosophy of User Interaction

GEANT4 is a “toolkit”. The user or installer at each site decides which features he requires, and compiles them with the kernel. In particular, he decides which graphics systems to use. Then the user compiles his geometry and user action functions and links to make a GEANT4 executable. For a particular GEANT4 executable, through the general GEANT4 User Interface (G4UI), which may be command-line-based or graphical, the user issues commands (Appendix A). For example, he may draw a detector component. He may also issue commands directly from C++ code, for example, from a user action function to draw steps and hits (Appendix B).

4 Features of GEANT4 Visualization

In GEANT4 Visualization, there are some particular possibilities, features or concepts which the user needs to appreciate.
4.1 Active View

The user may open several views of several scenes with several graphics systems. Thus he might have more than one windows open. So the Visualization Manager needs to have the concept of the “active view”, i.e., the one that the drawing commands will affect.

4.2 Scene and Scene Data

A “scene” is defined by a set of long-life GEANT4 objects which the user wishes to view. Together with their associated characteristics, they comprise the “scene data”. The GEANT4 objects which qualify are: physical volumes, trajectories (if they are stored), hits and digitisations, i.e., anything that is guaranteed to be available in memory at the end of simulating an event (that is what is meant by “long-life”). Steps and non-stored trajectories, so-called transient objects, are not part of the scene, but can be superimposed on individual views of it (Section 4.5).

Each scene is represented by a C++ object derived from G4Scene. Each such C++ object keeps its own scene data encapsulated in a G4SceneData object.

4.3 View and View Parameters

A “view” is a particular graphical rendering of a scene. Normally this is a window on the computer screen, but it may be a piece of paper (hardcopy). The view is defined by the scene and additional “view parameters”, such as viewpoint direction and lighting. The view parameters also include drawing style, for example, wireframe or solid, but this may be overridden for individual GEANT4 scene objects.

Each view is represented by a C++ object derived from G4View. Each such C++ object keeps its own view parameters encapsulated in a G4ViewParameters object.

4.4 Current Scene Data/View Parameters

The Visualization Manager keeps a memory of the scene data and view parameters — the so-called “current scene data” and “current view parameters” — which usually correspond to those of the active view.

However, the user can create a new view, or select a different existing view. The memory in the Visualization Manager means that if he now issues a draw command, the picture in the new window will be a view of the same scene with the same view parameters, rendered by the same or different graphics system. (Transient objects, such as steps, are not memorised in the scene data, so are excluded from this procedure.)

The user may also request the current scene data and view parameters to be “updated” to those of the active view. A subsequent change of view, followed by a draw, will therefore copy the view. In these ways, a user can have several views of the same or different scenes and copy them (except transient objects) from view to view, and graphics system to graphics system. In particular, he may experiment to find a suitable view using wireframe for speed, then copy, change drawing style to solid, to get a different view, or copy to the high quality renderer.

4.5 Transient Objects

“Transient” objects are those which exist temporarily during event simulation, such as G4Step objects, which the user might wish to visualize. He can superimpose them on the active view by writing C++ code in the appropriate user action routine. An example is given in Appendix B. He cannot copy them from view to view. They might or might not disappear if he attempts, for example, to change the viewpoint direction, depending on whether the graphics system keeps its own memory of drawn objects, for example, in a display list; that is, the behaviour is graphics-system dependent.
5 Conclusion

Some basic visualization facilities are available in the alpha release of GEANT4. Details are given in the Appendices. Try them, enjoy them or be frustrated by them — either way, let us know what you think.

A Commands Available through the GEANT4 User Interface

This is a transcript of a terminal session:

G4vis> ls
Command directory path : /vis/
Command /vis/
Guidance :
Visualization
Sub-directories :
   /vis/set/  Set UI parameters
   /vis/show/  Show UI parameters
Commands :
draw * Draws scene data
reset_view *
pan * Takes 2 parameters: right, up
dolly * Takes 1 parameter: move in
zoom * Takes 1 parameter: zoom factor
draw_spiral *
draw_spirals * Takes 1 parameter: no. of random spirals
spin * Takes 2 parameters: frames, degrees per frame
trig * Specify id, p, x, y, z, dx, dy, dz
G4vis> cd set
G4vis> ls
Command directory path : /vis/set/
Command /vis/set/
Guidance :
Set UI parameters
Sub-directories :
Commands :
.verbose * Takes 1 parameter: level [0-10]
.graphics_system *
.physical_volume *
.drawing_style *
.rep_style *
.projection_style *
.transients * Allows drawing of steps, etc., during tracking.
.no_transients * Disables drawing of steps, etc., during tracking.
.viewpoint * Takes 2 parameters: theta, phi (in degrees)
.view *

B User Action Functions

An example from G4UserSteppingAction.cc:

   G4Polyline pl;
   ...
   G4VisAttributes va(G4Colour(colR,colG,colB));
   pl.SetVisAttributes(&va);
   pl.append(fpSteppingManager->get_fpStep() ->getPrePosition());
   pl.append(fpSteppingManager->get_fpStep() ->getPostPosition());
C The C++ Abstract Interface of the Visualization Manager

G4VisManager is actually derived from G4VVisManager, and it is the latter that defines what is available to the user in user action routines.

```cpp
class G4VVisManager {
  public:
    static G4VVisManager* GetConcreteInstance();
    virtual void Draw(); // Draw current scene in current view.
    virtual void Draw(const G4Polyline&); // Draw "transient" primitives.
    virtual void Draw(const G4Polyhedron&); // Timing messages.
    virtual void EventBegins(); // All stacks are clear, event processing is about to start.
    virtual void EventEnded(G4Event *pEvent); // Event processing has ended, all event-persistent data is available.
};
```

D The C++ Abstract Interface of the Scene

G4VScene is itself derived from G4VGraphicsScene, and it is the latter that defines what is available to the user in user action routines.

```cpp
class G4VGraphicsScene {
  public:
    virtual void AddThis(const G4Box& box); // Functions for adding raw GEANT4 objects, if the graphics system can understand them (optional on the part of the graphics system).
    virtual void AddThis(const G4Cons& cons); // For solids not above...
    virtual void AddThis(const G4Trd& trd); // Timing messages.
    virtual void AddThis(const G4Tubs& tubs); // Event processing has ended, all event-persistent data is available.
};
```

References

3. Contact the CERN User Consultancy Office for details. At the time of writing, CERN has purchased a number of licences for the CERN site. Negotiations for off-site licenses for HEP are in progress.