Geant 4
Visualization and (G)UI

Authors: Hajime Yoshida and Satoshi Tanaka
Part 1: How to perform visualization
- Introduction
- Visualizable Objects
- Visualization Attributes
- Polyline and Marker
- Visualization Drivers
- Main() Function
- Visualization Commands
- How to Visualize from C++ Codes
- Exercises
- Information
Contents (2)

- Part 2: Geant4 GUI
  - Select (G)UI
  - Environmental variables
  - Useful GUI Tools Released by Geant4 Developers

- Part 3: Advanced Topics
  - Volume-Overlapping Detection with DAVID
Visualization and GUI

Part 1:

Geant4

Visualization
Geant4 Visualization must respond to varieties of user requirements. For example,

- 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 visualized objects
- Etc
2. Visualizable Objects (1)

- You can visualize simulation data such as:
  - 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

- Visualization is performed either with commands or by writing C++ source codes of user-action classes
2. Visualizable Objects (2)

- You can also visualize other user defined objects such as:
  - A polyline, that is, a set of successive line segments for, e.g., coordinate axes
  - A marker which marks an arbitrary 3D position, for, e.g., eye guides
  - Texts, i.e., 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 visualization attributes is held by class `G4VisAttributes`

- A `G4VisAttributes` object is assigned to a visualizable object with its `SetVisAttributes()` method, e.g.,
  - `experimentalHall_logical`  
    -> `SetVisAttributes(G4VisAttributes::Invisible)`
3.1 Constructors of 
G4VisAttributes

- The following constructors are supported by class G4VisAttributes:
  - G4VisAttributes G4VisAttributes ()
  - G4VisAttributes (G4bool visibility)
  - G4VisAttributes (const G4Colour& colour)
  - G4VisAttributes (G4bool visibility, const G4Colour& colour)
3.2 Visibility

- A boolean flag ($G4bool$) to control the visibility of objects

- Access function
  - $G4VisAttributes::SetVisibility (G4bool visibility)$
  - If you give false to the 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**
  - An equivalent class name, **G4Color**, is also available
- Class **G4Colour** is instantiated by giving RGB components to its constructor:
  - **G4Colour::G4Colour** ( **G4double r = 1.0,**
    **G4double g = 1.0,**
    **G4double b = 1.0** )
  - 0≤r, g, b ≤ 1.0
  - The default arguments define “white” color
3.3 Colour (2)

- Access functions of `G4VisAttributes` to set `G4Colour`
  - `SetColour (const G4Colour& colour)`
  - `SetColour (G4double r, G4double g, G4double b)`
3.4 Assigning G4VisAttributes a logical volume

- Class G4LogicalVolume holds a pointer of G4VisAttributes
- Access functions of G4LogicalVolume
  - SetVisAttributes (const G4VisAttributes* pva)
- Sample C++ codes
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 \textit{G4Polyline}
- Used to visualize tracking steps, particle trajectories, coordinate axes, etc
- \textit{G4Polyline} is defined as a list of \textit{G4Point3D} objects. Elements of the list define vertex positions of a polyline.

- \textbf{Sample C++ codes}
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, etc), color, and special properties of
  - (a) always facing the camera and
  - (b) having the possibility of its size (diameter) defined either in real 3D or 2D screen units (pixels)
4.2 Marker (2)

- **Kinds of markers**
  - Square : \textit{G4Square}
  - Circle : \textit{G4Circle}
  - Text : \textit{G4Text}

- **Constructors**
  - \textit{G4Circle} (const \textit{G4Point3D}\& pos )
  - \textit{G4Square} (const \textit{G4Point3D}\& pos)
  - \textit{G4Text} (const \textit{G4String}\& text, const \textit{G4Point3D}\& pos)
4.2 Marker (3)

- Each marker class inherits class G4VMarker
- All access functions of G4VMarker are available. For example,
  - SetPosition( const G4Point3D& )
  - SetWorldSize( G4double real_3d_size)
  - SetScreenSize( G4double 2d_size_pixel)
  - Others
- Sample [C++ codes](#) to define a small red circle as a marker
5. Visualization Drivers

- Visualization drivers are interfaces 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

- By default, Geant4 provides visualization drivers, i.e. interfaces for:
  - DAWN: Technical High-quality PostScript output
  - OPACS: Interactivity, unified GUI
  - OpenGL: Quick and flexible visualization
  - OpenInventor: Interactivity, virtual reality, etc
  - RayTracer: Photo-realistic rendering
  - VRML: Interactivity, 3D graphics on Web
5.2 Available Visualization Drivers and its Driver Name

- DAWNFILE → Fukui Renderer DAWN
- OPENGLX → OpenGL with Xlib
- OPACS → OPACS (o packages)
- OIX → OpenInventor with Xlib
- RAYTRACER → JPEG files
- VRMLFILE → VRML 1.0/2.0 viewer and file
- etc
5.3 How to Use Visualization Drivers

- Users can select/use visualization driver(s) by setting environmental variables before compilation:
  - `setenv G4VIS_USE_DRIVERNAME 1`

- **Example** (DAWNFILE, OpenGLXlib, and VRMLFILE drivers):
  - `setenv G4VIS_USE_DAWNFILE 1`
  - `setenv G4VIS_USE_OPENGLX 1`
  - `setenv G4VIS_USE_VRMLFILE 1`

- Note that Geant4 library should be installed with setting the corresponding environmental variables `G4VIS_BUILD_DRIVERNAME_DRIVER` to “1” beforehand, e.g.,
  - `setenv G4VIS_BUILD_DAWNFILE_DRIVER 1`

- **Sample “.cshrc” file**
6. main() Function (1)

- Derive your own concrete class from `G4VisManager` according to your computer environments.
- Describe the followings in your main():
  - Include the header file of your visualization manager.
  - Instantiate and initialize your visualization manager. The `Initialize()` method do the initialization.
  - Delete your visualization manager at the end.
- You can use the C macro “G4VIS_USE”, which is automatically set if you incorporate a visualization driver in compilation.
A typical form of main() function:

    // Include the header file of your visualization manager
    #ifdef G4VIS_USE
    #include "ExN03VisManager.hh"
    #endif

    // Instantiate and initialize the visualization manager
    #ifdef G4VIS_USE
    G4VisManager* visManager = new ExN03VisManager;
    visManager->Initialize();
    #endif

    // Delete the visualization manager
    #ifdef G4VIS_USE
    delete visManager;
    #endif
7. Visualization Commands

- Here, we introduce some frequently-used built-in visualization commands
- For simplicity, we assume that the Geant4 executable is compiled, incorporating DAWNFILE, OPENGLX, and VRMLFILE drivers
  - setenv G4VIS_USE_DAWNFILE 1
  - setenv G4VIS_USE_OPENGLX 1
  - setenv G4VIS_USE_VRMLFILE 1
7.1 Scene, Scene Handler, and Viewer

- In order to use visualization commands, you should understand ideas of “scene”, “scene handler”, and “viewer”.
- Scene: A set of visualizable 3D data
- Scene handler: CG-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
- “visualization driver” = “scene_handler” + “viewer”
7.2 Steps of Visualization

- **Step 1**: Create a scene handler and a viewer
- **Step 2**: Create an empty scene
- **Step 3**: Add 3D data to the created scene
- **Step 4**: Attach the current scene handler to the current scene
- **Step 5**: Set camera parameters, drawing style (wireframe/surface), etc
- **Step 6**: Make the viewer execute visualization
- **Step 7**: Declare the end of visualization
7.3 An Example of Visualizing Detector

- # Invoke the OGLIX driver: Create a scene handler and a viewer.
  /vis/open OGLIX

  # Set camera and drawing style
  /vis/camera/reset
  /vis/camera/viewpoint 70 20
  /vis/viewer/set/style wireframe

  # Visualize of the whole detector geometry
  # The “/vis/drawVolume” create a scene, add the world volume
  # to it, and let viewer execute visualization..
  # The “/vis/viewer/update” declare the end of visualization.
  /vis/drawVolume
  /vis/viewer/update
7.4 An Example of Visualizing Events

- # Store particle trajectories for visualization
  /tracking/storeTrajectory

  # Invoke the DAWNFILE driver: Create a scene handler and a viewer.
  /vis/open DAWNFILE

  ….. Camera setting, and drawing style selection, if necessary …..

  # Create a new empty scene
  /vis/scene/create

  # Add the world volume and trajectories to the current scene
  /vis/scene/add/volume
  /vis/scene/add/trajectories

  # Let the viewer visualize the scene, and declare end of visualization
  /run/beamOn 10
7.5 /vis/open Command

- Command
  - Idle> /vis/open <driver_tag_name>
  - The “driver_tag_name” is a name which shows “driver name” + “mode”

- Action: Create a visualization driver
  - In other words, create a scene hander and a viewer

- 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
7.6 /vis/camera/… Commands

- Commands
  - Viewpoint setting:
    Idle> /vis/camera/viewpoint <theta_deg> <phi_deg>
  - Zooming
    Idle> /vis/camera/zoom <scale_factor>
  - Initialization of camera parameters:
    Idle> /vis/camera/reset
7.7 /vis/viewer/set/style Command

- Command
  - Idle> /vis/viewer/set/style <style_name>
  - The “style_name” is “wireframe”, “surface”, etc
7.8 /vis/drawVolume and /vis/viewer/update Commands

- Commands:
  - Idle> /vis/drawVolume <physical-volume-name>
    (Default: world)
  - Idle> /vis/viewer/update
  - Note that /vis/viewer/update should be executed to declare end of visualization.

- You can add visualization commands of, say, coordinate axes between the two commands. For example,
  - Idle> /vis/drawVolume
  - Idle> /vis/draw/axes <Ox> <Oy> <Oz> <length>
  - Idle> /vis/viewer/update
  - Note: “/vis/draw/axes” will be renamed to “/vis/scene/add/axes”
7.9 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 (2)
Sample Visualization (3)
8. Visualization from C++ codes

- It is also possible to perform visualization from C++ codes.
- You can describe the visualization commands in C++ codes via the `ApplyCommand()` method of the UI manager:
  - `pUI->ApplyCommand("/vis/…");`
- Or you can use `Draw()` methods of visualizable classes.
9. Exercises

- Read and execute sample visualization macros for examples/novice/N03
  - The macro files are “exN03VisX.mac”, where $X=0,1,2,…$
  - Explanation of macros is all described in the macro files as comment lines.
10. Information

- Geant4 User Guide (and source codes)
- README file:
  - geant4/source/visualization/README
- Link collection
- Information on Geant4 visualization on Linux
  - [http://geant4.kek.jp/~tanaka/GEANT4/g4vis_on_linux.html](http://geant4.kek.jp/~tanaka/GEANT4/g4vis_on_linux.html)
Part 2: Geant4 GUI
1. Select (G)UI (1)

- In your `main()`, according to your computer environments, construct a G4UIsession concrete class provided by Geant4 and invoke its `sessionStart()` method.

- Example:
  ```cpp
  G4UIsession* session=0;
  if (argc==1) // Define UI session for interactive mode.
  {
    // G4UIterminal is a (dumb) terminal
    session = new G4UIterminal;
  }
  ```
1. Select (G)UI (2)

- Geant4 provides the following interfaces for various (G)UI:
  - G4UItterminal: C-shell like character terminal
  - G4UItcsh: tcsh-like character terminal with command completion, history, etc
  - G4UIGAG: Java based GUI
  - G4UIOPACS: OPACS-based GUI, command completion, etc
  - G4UIBatch: Batch job with macro file
  - G4UIXm: Motif-based GUI, command completion, etc

- Note for G4UItcsh:
  - Use G4UItterminal with argument G4UItcsh*:
    
    session = new G4UItterminal (new G4UItcsh) ;
2. Environmental Variables

- Users can select and plug in (G)UI by setting environmental variables before compilation
  - `setenv G4UI_USE_GUINAME`

- Example ("G4UIterminal", "GAG", and Motif)
  - `setenv G4UI_USE_TERMINAL 1`
  - `setenv G4UI_USE_GAG 1`
  - `setenv G4UI_USE_XM 1`

- Note that Geant4 library should be installed with setting the corresponding environmental variable `G4VIS_BUILD_GUINAME_SESSION` to “1” beforehand
3. Useful GUI Tools Released by Geant4 Developers

- **GGE**: Geometry editor based on Java GUI
  - [http://erpc1.naruto-u.ac.jp/~geant4](http://erpc1.naruto-u.ac.jp/~geant4)

- **GPE**: Physics editor based on Java GUI
  - [http://erpc1.naruto-u.ac.jp/~geant4](http://erpc1.naruto-u.ac.jp/~geant4)

- **OpenScientist, OPACS**: Flexible analysis environments
  - [http://www.lal.in2p3.fr/OPACS](http://www.lal.in2p3.fr/OPACS)
Visualization and GUI

Part 3: Advanced Topics
1. Volume-Overlapping Detection with DAVID (1)

- DAVID
  - DAWN-based Visual Volume Intersection Debugger
  - DAVID automatically detects and highlights overlapping of volumes
    - Precise visualization with DAWN
    - Interactive visualization with VRML
  - DAVID also generates log files describing detailed information on the detected overlapping
1. Volume-Overlapping Detection with DAVID (2)

- **Usage of DAVID**
  - Switch the viewer of the DAWNFILE driver from renderer DAWN (default) to DAVID.
    - `setenv G4DAWNFILE_VIEWER david`
  - Then visualize volumes as usual with the DAWNFILE driver
  - Then volume overlapping is, if any, visualized
    - The view is also stored in files g4david.prim (DAWN format) and g4david.eps (PostScript format)
    - Log file: g4david.log
1. Volume-Overlapping Detection with DAVID (3)

- Sample visualization with overlapped volumes highlighted
1. Volume-Overlapping Detection with DAVID (4)

- Log file format
  - PhysVolName.CopyNo Shape line_num
  - The “line_num” is the line number of the overlapping volume in the DAWN-format file “g4.prim file” generated by Geant4

- Sample log file:

```
.....
!!! INTERSECTED VOLUMES !!!
caloPhys.0: Tubs: line 17
caloPhys.1: Tubs: line 25
.....
```
1. Volume-Overlapping Dection with DAVID (5)

- If no overlapping is detected, DAVID displays the following message:

```
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
!!! Number of intersected volumes : 0 !!!
!!! Congratulations ! (^o^)/ !!!
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```