0.1 Introduction

This is a personal introduction to Geant4. https://www.ge.infn.it/geant4/events/nss2003/geant4lectures.html

Event biasing (variance reduction) techniques:

- Primary event biasing
  Biasing primary events/particles in terms of type of event, momentum distribution.

- Leading particle biasing
  Taking only the most energetic (or most important) secondary.
  Simulating of a full shower is an expensive calculation. Instread of generating a full shower, trace only the most energetic secondary. Other secondary particles are immediately killed befoer being stacked. In this way, it is convenient to roughly estimate, e.g. the thickness of a shielf. Of course, physical quantities such as energy are not conserved for each event.

- Physics based biasing
  Biasing secondary production in terms of particle type, momentum distribution, cross-section, etc.

- Geometry based biasing
  Importance weighting for volume/region.
  Duplication of sudden death of tracks.
  Define importance for each geometrical region, duplicate a track with relative weight if it goes toward more important region. Russian-roulette in another direction. Scoring particle flux with weights.

- Forced interaction
  Force a particular interaction, e.g. within a volume.

0.2 Basic

Basic ideas and concepts in Geant4.

0.2.1 Example

1. General
   (a) Configure the **Run**
   (b) Configure the **Event** Loop
2. Experimental set-up
   (a) geometrical set-up.
   (b) the coordinates of impact of tracks in the layers of the tracker, energy release in the strips of the tracker.
   (c) energy deposited in calorimeter
   (d) energy deposited in anticoincidence(?)
   (e) Digitise the hits, setting a threshold for the energy deposit in the tracker
   (f) Generate a trigger signal combining signals from different detectors.
3. Physics
   (a) Primary events
   (b) Electromagnetic processes appropriate to the energy range of the experiment
   (c) Hadronic processes
4. Analysis
   (a) x-y distribution of impact of the track
   (b) histograms during the simulation execution
   (c) store significant quantities in a ntuple
   (d) Plot energy distribution in the calorimeter
5. Visualisation
   (a) Visualize the experimental set-up
   (b) trakcs
   (c) hits
6. UI
   (a) Configure the tracker, by modifying the number of active planes, the pitch of the strips, the area of silicon tiles, the material of the converter.

(b) Configure the calorimeter, by modifying the number of active elements, the number of layers.

(c) Sources

(d) Digisation by modifying threshold

(e) Histograms

7. Persistency

   (a) Produce an intermediate output of the simulation at the level of hits in the tracker

   (b) Store significant results in FITS format

   (c) Read in an intermediate output for further elaboration.

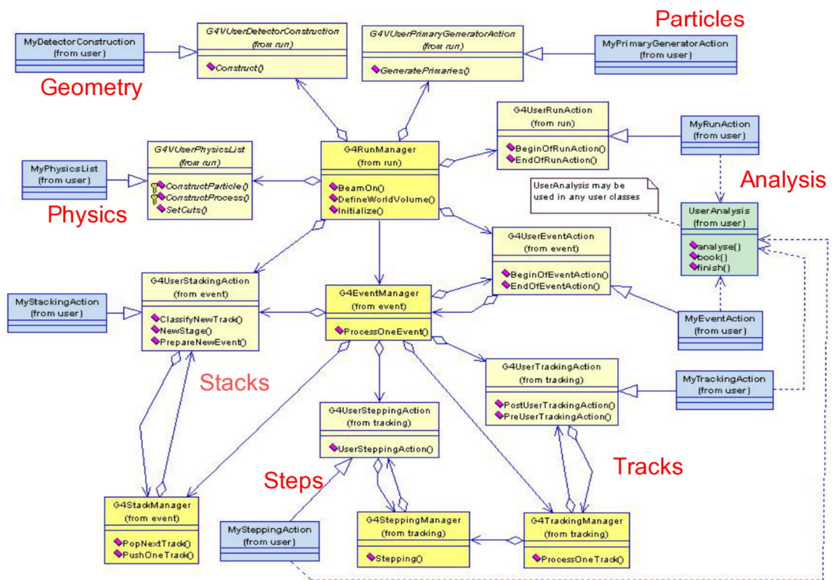
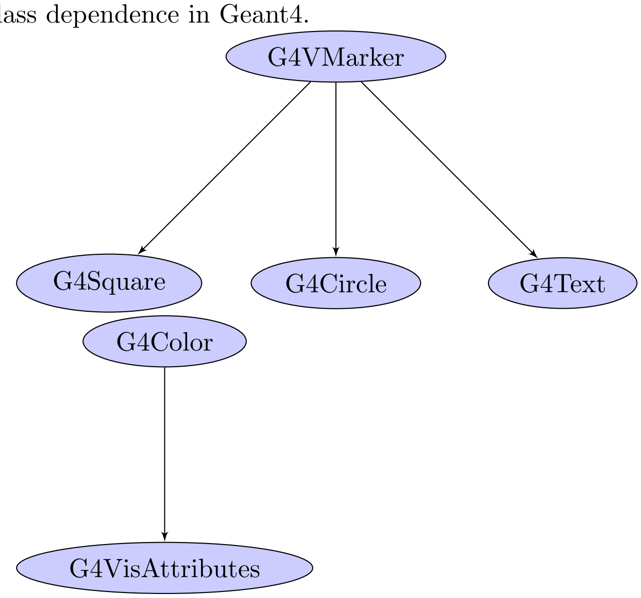


Figure 1: Geant4 running example

0.3 Class

Class dependence in Geant4.



0.4 Visulization

Visulization can be performed either with commands or by C++ codes of user-action classes.

0.4.1 Visualisable Objects

- Detector components
- Physical volumes, logical volumes, solid
- trajectory, tracking step
- hits in detector components
- polyline, marker, texts

0.4.2 Visulization Attributes

A set of visulization attributes is held by the class G4VisAttributes. A G4VisAttributes object is assigned to a visualisable object with its mtheod **SetVisAttributes()**:

```
volume->SetVisAttributes(G4VisAttributes::Invisible);
```

Class G4LogicalVolume holds a pointer of G4VisAttributes.

- Color, forced-wireframe style(What's this ??)

0.4.3 Commands

Scene A set of visualizable 3D data

Scene hadler computer graphics data modeler, which uses raw data in a scen

Viewer Image generator

Each scene handler is assigned to a scene and each viewer is assigned to a scene handler. "Visualisation driver" = "scene handler" + "viewer".

Examples:

```
# Invoke the OGLIX driver:
# Create a scene handler and a viewer
/vis/open OGLIX
# Set the camera and drawing style
/vis/viewer/reset
/vis/viewer/viewpointThetaPhi 70 20
/vis/viewer/set/style wireframe
# Visualizw the whole detector geometry
# The "/vis/drawVolume" create as cene, add the world volume to it, and let
# viewer execute visualisation
```

```
/vis/drawVolume
# the end of visualisation
/vis/viewer/update


# Visualizing Events
# Store particle trajactories for visualisation
/tracking/storeTrajactory 1
# DAWN driver, scene handler and viewer:
/vis/open DAWN
# 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 visualise the scene, and declare the end of visualisation
/run/beamOn 10

# List available driver
help /vis/open
help /vis/sceneHandler/create
```

0.4.4 C++

To perform visualisation in C++ code, use the **ApplyCommand()** method of the US amnager, as for any other command: `pUI->ApplyCommand("/vis/...");` Or use Draw() methods of visualizable classes.

0.4.5 (G)UI

G4UIterminal C-shell like character terminal

G4UItcsh tcsh term, with command completion, history

G4UIGAG Java based GUI

G4UIXm Motif-based GUI, command completion.

0.4.6 DAVID

DAWN-based Visual Volume Intersection Debugger Automatically detects and highlights overlapping volumes. It also generates log files describing detailed info on the detected overlap.

0.4.7 DTREE

DTREE is the function to visualisa detector-geometry tree. How to display a tree: `/vis/drawTree ! XXXTree` XXX = Atree(ASCII), GAGTree(GAG), XMLTree(XML), etc. Detail level: `/vis/XXXTree/verbose n`