UI and Visualisation

Updates and reports

G4 Workshop Catania 2024

Overview

- Geant4 11.3 adapted to Qt6
  - Qt6 will power our “flagship” GUI
    - Qt5 is still functional
  - Development by Guy Barrand – see Guy’s report
- Vtk comes of age
  - Development by Stewart Boogert – see Stewart’s report
- Some consolidated features
  - A review – this report
  - The future – a discussion

Contributors

- Main contributors listed (with approximate start year) – but there is much cross-fertilization, including from the Geometry Category
- **UI systems**
  - Core framework: Hajime Yoshida, Koichi Murakami, and Guy Barrand (1998)
  - Xm GUI: Guy Barrand (1998)
  - Qt GUI: Laurent Garnier (2007), Guy Barrand (2007) and John Allison (2023)
  - Win32 GUI: O. Pena-Rodrigue and Gabriele Cosmo (2021)
- **Graphics Reps** (low-level library of graphics primitives, etc)
  - Core primitives and vis manager virtual interface: John Allison (1994)
  - Polyhedron and Boolean processors: Evgueni Tchernaev (1998)
- **Modeling**
  - Core models: John Allison (1994)
  - Trajectory models and filters: Jane Tinslay (2005)
- **Graphics**
  - Core vis manager: John Allison (1994)
  - OpenGL drivers
    - Core scene handling: Andy Walkden and John Allison (1996)
    - Qt extension: Laurent Garnier (2007)
  - Open Inventor drivers: Joe Boudreau (1996), Guy Barrand (2004), Fred Jones (2012)
  - Ray Tracer: Makoto Asai (2000)
  - DAWN and VRML: Satoshi Tanaka (1996)
  - HepRepFile: Joseph Perl (2001)
  - Vtk drivers: Stewart Boogert (2021)
  - ToolsSG drivers: Guy Barrand (2021)
  - ASCII Tree: John Allison (2001)
  - Others...

Available User Interfaces and Graphics Drivers

- User Interfaces
  - Plain terminal
    - csh & tcsh
  - Graphical
    - Xm (motif)
      - In-window graphics, interactive help, and output
    - Win32 (Windows)
      - As above
    - Qt (cross-platform UI system)
      - As above, plus an interactive scene tree
        - Select visibility and colour of geometry objects...and more (described later)
- Graphics Drivers
  - Graphical
    - OpenGL (some advanced features, also export)
    - OpenInventor (some advanced features, VRLM export)
    - ToolsSG (GLES/Qt/ZB, export and plotting)
    - Qt3D (prototype)
    - Vtk (many interactive options and export formats)
    - RayTracerX (export jpeg)
    - ...
  - File-writing (always registered):
    - DAWNFILE (browse with DAWN)
    - HepRepFile (browse with HepRApp)
    - VRML2FILE (browse with any web browser)
    - RayTracer (export jpeg)
    - ToolsSGOffscreen (numerous picture formats)
  - Diagnostic (always registered):
    - ASCIITree (always registered, dumps to G4cout)

Installing libraries

- Installing X11
  - MacOS: Xquartz
  - Linux: native
  - Windows: X11 not available, but special drivers use the native windowing system.
- Installing Qt
  - MacOS: **brew install qt**
    - For Qt5: **brew unlink qt** (to avoid linking to Qt6)
  - Linux: **apt install qtbase5-dev**
  - Windows: Binaries from Qt web site (qt.io)
- Installing Vtk
  - MacOS: **brew install vtk**
  - Linux: **apt**
  - Windows: **conda**?
- Installing Coin3D (Open Inventor)
  - Coind3D (Open Inventor): See https://www.coin3d.org

UI and vis build options

- From the Installation Guide, Build Options:
  - GEANT4_USE_INVENTOR (DEFAULT : OFF)
  - GEANT4_USE_INVENTOR_QT (DEFAULT : OFF)
  - GEANT4_USE_OPENGL_WIN32 (DEFAULT : OFF, Windows Only)
  - GEANT4_USE_OPENGL_X11 (DEFAULT : OFF, Unix Only)
  - GEANT4_USE_QT (DEFAULT : OFF)
  - GEANT4_USE_QT_QT6 (DEFAULT : OFF)
  - GEANT4_USE_RAYTRACER_X11 (DEFAULT : OFF, Unix only)
  - GEANT4_USE_VTK (DEFAULT : OFF)
  - GEANT4_USE_XM (DEFAULT : OFF, Unix Only)
- Set required options ON, e.g:
  - GEANT4_USE_QT=ON
- Don't mix X11 and Qt (unless you know what you're doing!!)
- GEANT4_USE_QT brings in:
  - OpenGLQt
  - ToolsSGQt
  - Qt3D (under development)

```
My cmake - suppress messages:
---------
cmake -Wno-dev --log-level=ERROR \
-DCMAKE_INSTALL_PREFIX=`pwd` \
-DGEANT4_USE_GDML=ON \
-DGEANT4_USE_QT=ON \
-DGEANT4_USE_QT_QT6=ON \
-DCMAKE_PREFIX_PATH=\
"$(brew --prefix qt@5)" \
-DGEANT4_ENABLE_TESTING=ON \
-DGEANT4_USE_FREETYPE=ON \
-G Xcode \
~/Geant4/geant4-dev
```

A minimal starter

- A minimal **main()**
- A simple **vis.mac**
- And **gps.mac**
  - Uses **G4GeneralParticleSource** – see next slide

```
int main(int argc,char** argv) {
  auto ui = new G4UIExecutive(argc,argv);
  auto runMan = G4RunManagerFactory::CreateRunManager();
  runMan->SetUserInitialization(new DetectorConstruction);
  auto physList = G4PhysListFactory().ReferencePhysList();
  runMan->SetUserInitialization(physList);
  runMan->SetUserInitialization(new ActionInitialization);
  auto visMan = new G4VisExecutive;
  visMan->Initialise();
  auto UImanager = G4UImanager::GetUIpointer();
  UImanager->ApplyCommand("/control/execute vis.mac");
  ui->SessionStart();
  delete ui;
  delete visMan;
  delete runMan;
}
```

```
gps.mac
-------
/gps/particle proton
/gps/position 0 0 -20 cm
/gps/direction 0 0 1
/gps/energy 70 MeV
```

```
vis.mac
-------
/control/verbose 2
/run/initialize
/vis/open
/vis/drawVolume
/vis/viewer/set/viewpointThetaPhi 110 170 deg
/vis/scene/add/axes
/vis/scene/add/trajectories
/vis/scene/endOfEventAction accumulate
/control/execute gps.mac
```

A PrimaryGeneratorAction

- Uses **G4GeneralParticleSource**
  - Instantiated in **PrimaryGeneratorAction**
    - Itself instantiated in **ActionInitialization**

```
#ifndef PrimaryGeneratorAction_hh
#define PrimaryGeneratorAction_hh 1
#include "G4VUserPrimaryGeneratorAction.hh"
class G4GeneralParticleSource;
class PrimaryGeneratorAction :
  public G4VUserPrimaryGeneratorAction {
public:
  PrimaryGeneratorAction();
  ~PrimaryGeneratorAction();
  void GeneratePrimaries(G4Event*);
private:
  G4GeneralParticleSource* fpParticleGun;
};
#endif
```

```
#include "PrimaryGeneratorAction.hh"
#include "G4GeneralParticleSource.hh"
PrimaryGeneratorAction::PrimaryGeneratorAction()
{fpParticleGun  = new G4GeneralParticleSource;}
PrimaryGeneratorAction::~PrimaryGeneratorAction()
{delete fpParticleGun;}
void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{fpParticleGun->GeneratePrimaryVertex(anEvent);}
```

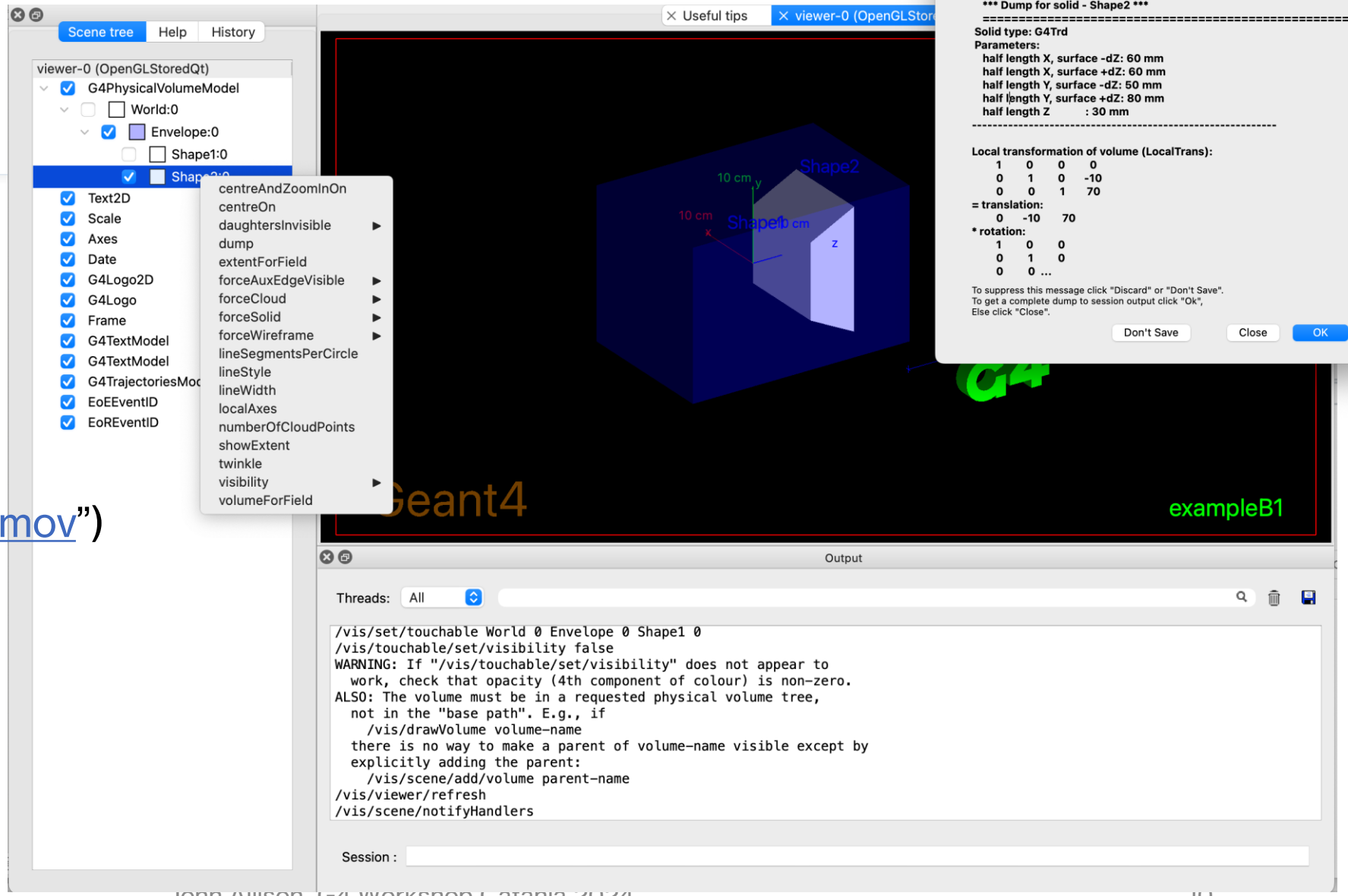
How to choose your UI and Vis driver at run time

- If you do nothing before start-up, the "best" UI and vis are chosen according to your original build flags
- Choosing with environment variables
  - **UI**: G4UI_USE_TCSH=1, G4UI_USE_WIN32=1, G4UI_USE_XM=1, or G4UI_USE_QT=1
  - **Vis**: G4VIS_DEFAULT_DRIVER=TSG (or any from the list of registered drivers)
- Using **~/.g4session**
  - This is a dot (.) file you keep in your home directory (~/)
  - The format is:
    - First line: chosen default UI session: Qt, Xm, Win32, tcsh or csh
    - Subsequent lines: **<app> <session> [<vis>] [<window-size>]**
- Note: Environment takes precedence.
- After all the above you can still change driver with **/vis/open [driver]**

```
Qt  # Default session
#exampleB1 tcsh
#exampleB1 Qt TSG 1000x1000+0-0
#exampleB1 Qt Qt3D
#exampleB2a tcsh
exampleB2b Qt Qt3D
#exampleB2b tcsh
#exampleB3 tcsh
#mvexampleB4a Xm
exampleB5 Qt TSG
#test202 tcsh
#test202 Qt Vtk
#userVisAction tcsh
```

The Qt GUI

- Scene tree
  - Click to make (in)visible, or change colour
  - Right-click to get other actions (see video "Scene tree centreAndZoomInOn.mov")



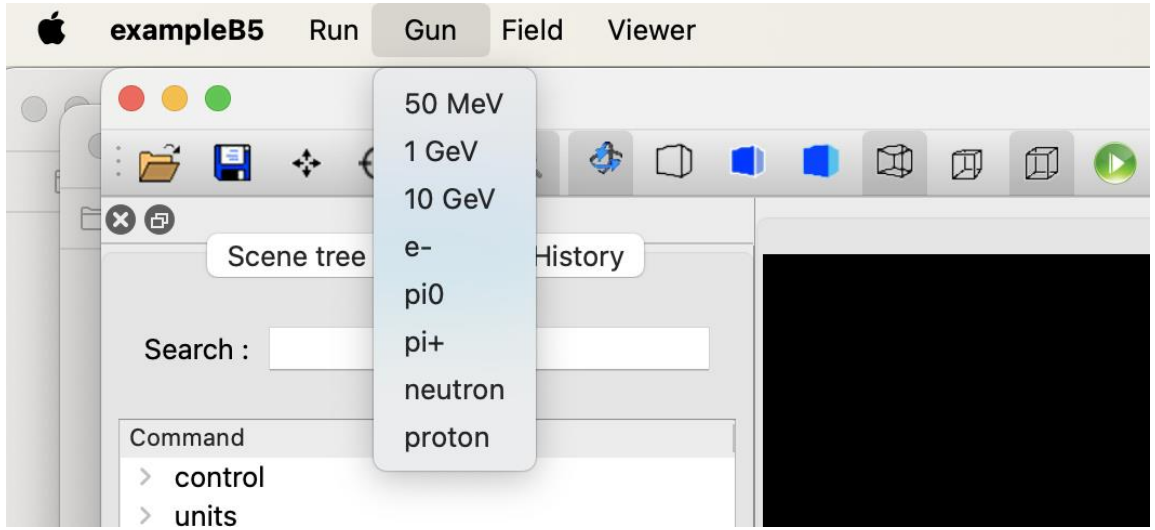
Adding your own action buttons or icons

- The Qt GUI has several built-in buttons

  - You can remove, then add back what you want
    - **/gui/defaultIcons false**
    - **/gui/addIcon "Move" move**
    - See **examples/basic/B5/icons.mac**
- You can add your own **gui.mac** (applies also to the the Xm GUI)
  - See basic examples B2, B4 and B5

```
UImanager->ApplyCommand("/control/execute init_vis.mac");
if (ui->IsGUI()) {
  UImanager->ApplyCommand("/control/execute gui.mac");
}
```

```
/gui/addMenu gun Gun
/gui/addButton gun "50 MeV"  "/gun/energy 50 MeV"
/gui/addButton gun "1 GeV"   "/gun/energy 1 GeV"
/gui/addButton gun "10 GeV"  "/gun/energy 10 GeV"
```



Command-based visualization – basics

- To echo commands: **/control/verbose 2**
- Adjust verbosity:
  /vis/verbose [verbosity]
  - E.g., **/vis/verbose confirmations**

Simple graded message scheme - digit or string (1st character defines):
0) quiet, // Nothing is printed.
1) startup, // Startup and endup messages are printed...
2) errors, // ...and errors...
3) warnings, // ...and warnings...
4) confirmations, // ...and confirming messages...
5) parameters, // ...and parameters of scenes and views...
6) all // ...and everything available.

- **/vis/disable**
  - Good to turn off trajectory storing as well: **/tracking/storeTrajectory 0**
- Useful commands: **/vis/list**, **/vis/viewer/list**, **/vis/scene/list**
- **/vis/open [<driver>]**
  - There is a default driver (OGL at present)
    - The default can be changed by environment or **~/.g4session**
  - **/vis/open** is actually **/vis/sceneHandler/create** + **/vis/viewer/create**
  - You can open multiple drivers of the same or different type, and multiple viewers of each driver
- **/vis/drawVolume [<physical-volume-name>]**
  - Actually **/vis/scene/create** + **/vis/scene/add/volume** + **/vis/sceneHandler/attach**
- **/vis/scene/add/trajectories [rich] [smooth]**
  - Adjust presentation with **/vis/modeling/...**, selection with **/vis/filtering/**
- **/vis/scene/add/axes**, et (see **examples/basic/B1/vis.mac**)

Scene editing

- Adding specific volumes (the extents are accumulated so always seen in standard view)
    - **/vis/scene/create**
    - **/vis/scene/add/volume A**
    - **/vis/scene/add/volume B**
    - **…**
    - **/vis/sceneHandler/attach**
- Adding trajectories (also hits or digis, if **Draw** methods implemented)
    - **/vis/scene/add/trajectories [rich] [smooth]**
- Event display and keeping behaviour (similarly for run)
    - **/vis/scene/endOfEventAction <refresh|accumulate> [number-of-events-to-be-kept]**
- Activate and de-activate models with **/vis/scene/activateModel**

Scene editing (contd)

- Changing a vis attribute with **/vis/geometry**
  - E.g., **/vis/geometry/set/colour <logical-vol-name> <colour>**
  - Changes the vis attributes in the actual logical volume
    - So changes them for all touchables that have that logical volume
  - Restore original vis attributes with **/vis/geometry/restore**
- To change the vis attributes of a specific touchable, use **/vis/set/touchable** and **/vis/touchable/set**
  - see later: **View Control**

Trajectory modeling

- Models and filters come with a fertile and versatile set of commands
  - Well documented for models and filters, but in-app guidance somewhat cryptic
- Creating a model creates its specific commands
  - E.g., **/vis/modeling/trajectories/drawByCharge-0/set 0 pink**
  - The commands are created “on the fly” for each model or filter
- Also, every model has a set of “defaults”
  - This creates a corresponding set of commands for customisation
    - Use **help** and guidance to see them
    - Of particular interest is **setTimeSliceInterval**
      - Needs rich trajectories


```
/vis/scene/add/trajectories rich
/vis/modeling/trajectories/drawByCharge-0/default/setTimeSliceInterval 0.01 ns
```

Trajectory filtering

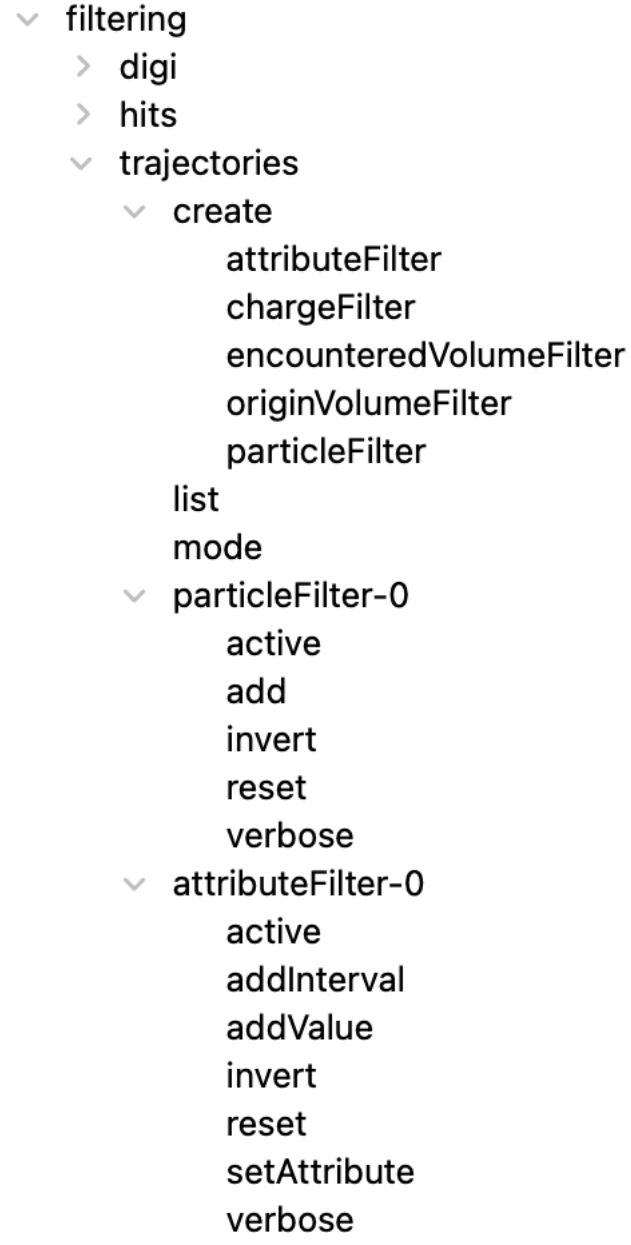
- Similarly, create a filter

```
# To draw only gammas:
/vis/filtering/trajectories/create/particleFilter
/vis/filtering/trajectories/particleFilter-0/add gamma
```

- Or its inverse

```
# To draw all except gammas
/vis/filtering/trajectories/particleFilter-0/invert true
```

- Filters can be chained



Event keeping

- By default, the vis manager keeps 100 events
  - The default can be changed with **/vis/scene/endOfEventAction**
  - They can be viewed at end of run
    - One by one: **/vis/reviewKeptEvents**
- If you want to keep your own events
  - For example, select a rare event in end-of-event action
  - **/vis/drawOnlyToBeKeptEvents**
    - This turns off event keeping by the vis manager
      - You can do this manually with **/vis/scene/endOfEventAction accumulate 0**

```
auto evMan = G4EventManager::GetEventManager();
if (<selection-success>) {
  evMan->KeepTheCurrentEvent();
}
```

- Actually, the events are kept by the run manager
  - They are deleted at the start of the next run

Saving, replaying and interpolating views

- **/vis/viewer/save [filename.g4view]**
  - By default, saves to **g4_00.g4view**, **g4_01.g4view**,…
- Then, to get the view again
  - **/control/execute g4_00.g4view**
- Or interpolate to get an animation
  - **/vis/viewer/interpolate**
  - Can be used to file and make a movie
- This extends to time-slicing – next slide…

Time-slicing
See examples/extended/visualization/movies

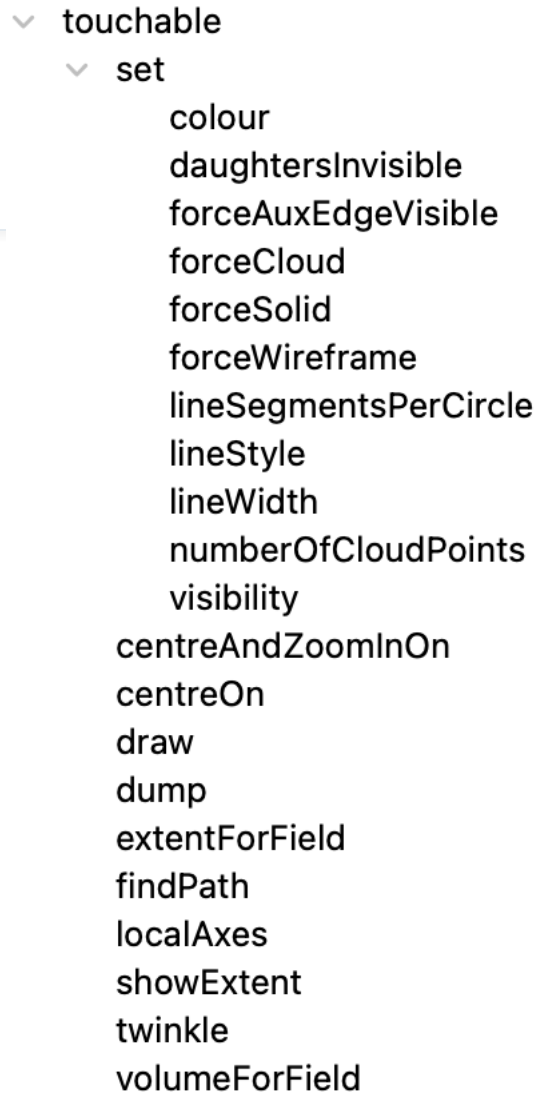
```
/vis/scene/add/trajectories rich
/vis/modeling/trajectories/drawByCharge-0/default/setTimeSliceInterval 0.01 ns
# Optionally add features (see guidance on /vis/viewer/set/timeWindow/)
/vis/viewer/set/timeWindow/displayLightFront true 0 0 -20 cm -0.01 ns
/vis/viewer/set/timeWindow/displayHeadTime true
/vis/viewer/set/timeWindow/fadeFactor 1
/run/beamOn
# Then set a time window and save
/vis/viewer/set/timeWindow/startTime 0 ns .1 ns
/vis/viewer/save
# Then zoom, pan etc to a view of interest
# Then set the next time window and save
/vis/viewer/set/timeWindow/startTime .5 ns .1 ns
/vis/viewer/save
# Then zoom, pan etc to a view of interest
# Then set the next time window and save
/vis/viewer/set/timeWindow/startTime 1 ns .1 ns
/vis/viewer/save
# Then another view, the next time window, and a save...
# ...repeat a few more times
# Then try
/vis/viewer/interpolate
```

Making movies

- On MacOS, Quicktime has a “Screen recording” feature
  - Just run your app
    - To record sound: on the small control panel: Options/Microphone and select
    - To stop recording, there’s a button on the top bar
- Alternatively, export images (OpenGL only, I think)
  - **/vis/viewer/interpolate ! ! ! ! export**
    - This produces lots of files
  - You can change export format with (for example)
    - **/vis/ogl/set/exportFormat jpg**
  - Then import them into your favourite movie maker – see Making a movie

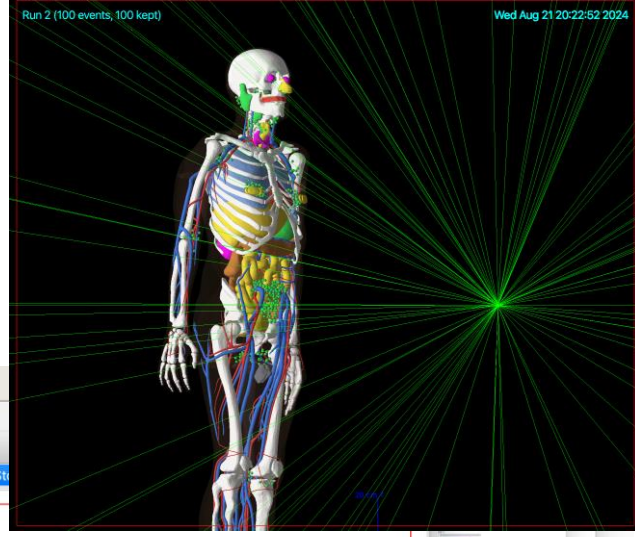
Touchables

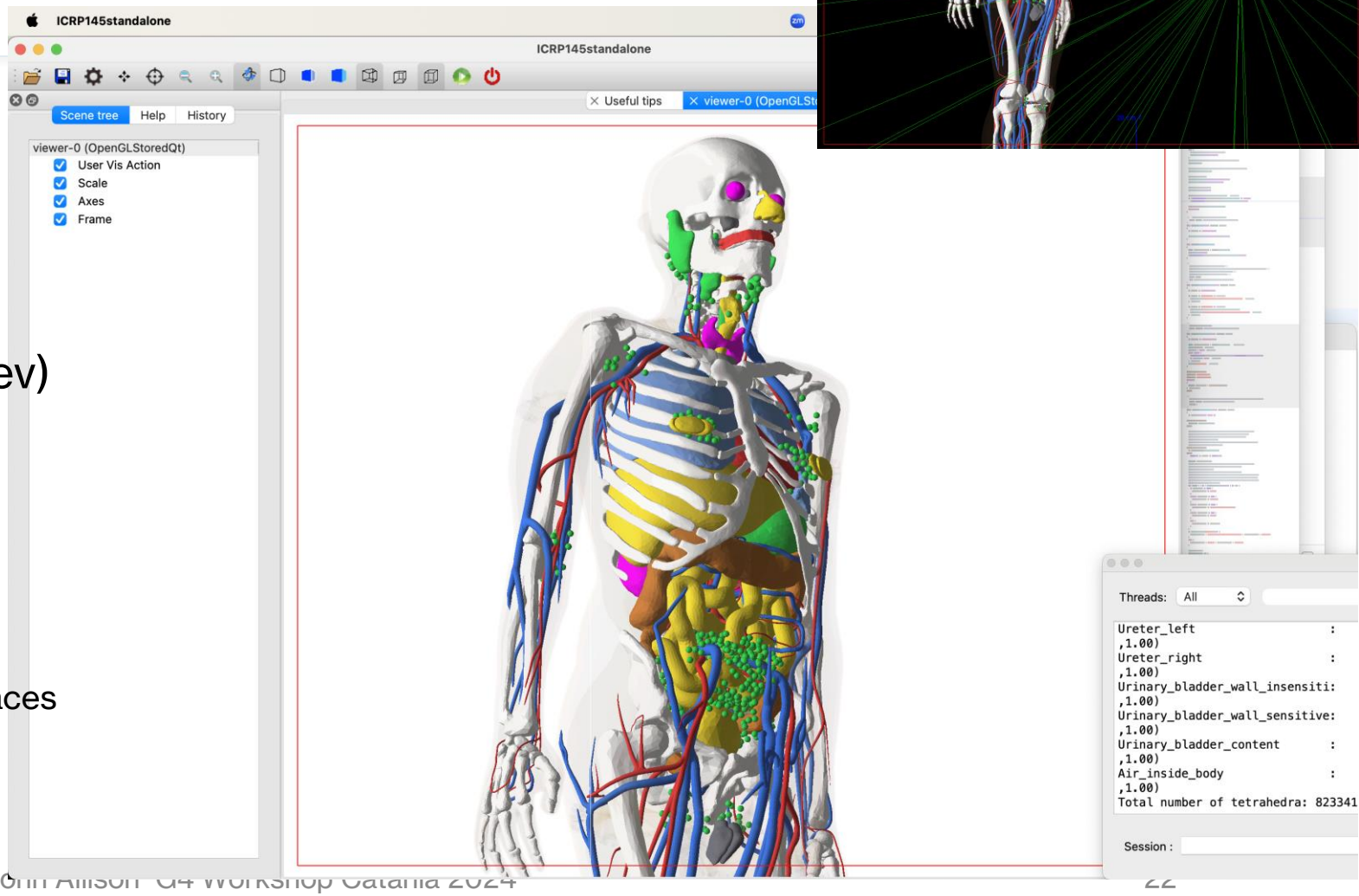
- **Touchables** are the leaves of the geometry hierarchy tree
- To print the tree: **/vis/drawTree**
  - Various verbosity options: **/vis/ASCIITree/verbose**
- To find the path to a touchable
  - **/vis/touchable/findPath <physical-vol-name>**
- Then, with information supplied, **/vis/set/touchable**
- Then **/vis/touchable/set/colour** (other options!!)
- Also **/vis/touchable/twinkle** (and other options!!)
- This is all achieved using **Vis Attribute Modifiers (VAMs)**
  - **VAMs** belong to the viewer, and may be copied with **/vis/viewer/clone** or **/vis/open** + **/vis/viewer/copyViewFrom**
    - So they can be different for different views of the same scene



Meshes (G4VNestedParameterisation)

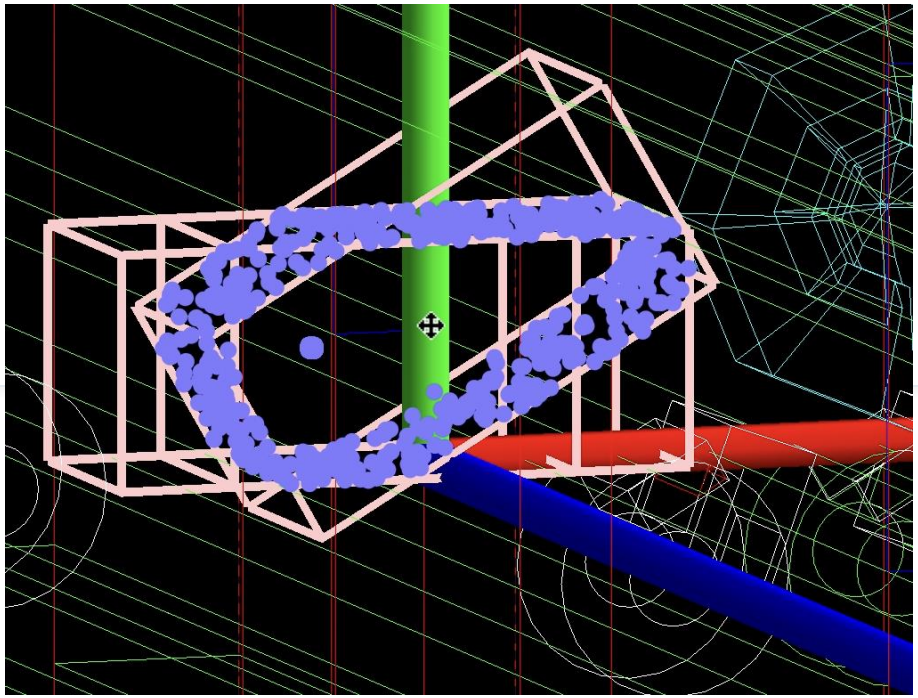
- Typically used for representing regions of variable material
  - E.g., human phantom, **examples/advanced/ ICRP145_HumanPhantoms**
- Only rectangular and tetrahedral at present
  - Clever algorithm to eliminate shared surfaces (E. Tcherniaev)
- Dots or surfaces representation





Draw geometry:
/vis/viewer/set/specialMeshRendering
/vis/viewer/set/specialMeshRenderingOption surfaces
/vis/drawVolume

Geometry overlaps and other useful things



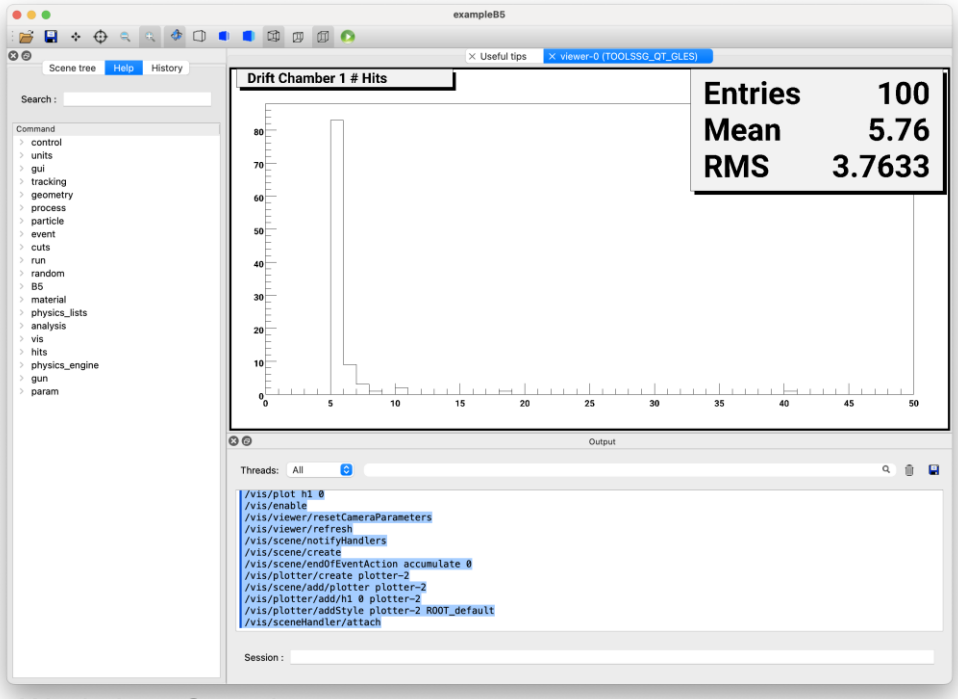
- /vis/drawLogicalVolume <log-vol-name>
  - Boolean components
  - Geometry voxels
  - Local axes
  - Geometry overlaps
  - Demo

```
Command /vis/drawLogicalVolume
Guidance :
Draws logical volume with additional components.
Synonymous with "/vis/specify".
Creates a scene consisting of this logical volume and asks the
  current viewer to draw it. The scene becomes current.
Adds a logical volume to the current scene,
Shows boolean components (if any), voxels (if any), readout geometry
  (if any), local axes and overlaps (if any), under control of the
  appropriate flag.
  Note: voxels are not constructed until start of run -
 "/run/beamOn".  (For voxels without a run, "/run/beamOn 0".)
```

Plotting

- If you have histograms registered with the Analysis Manager you can plot them at end of run
  - Only simple 1D and 2D with boxes for now
- At the end of run, the run manager prints
- Only with ToolsSG (TSG) viewer
  - For best results, build with GEANT4_USE_FREETYPE=ON
    - Choice of styles – see guidance
    - From Geant4 11.3 (December 2024)
- Try basic example B5
  - /run/beam 100
  - /vis/plot h1 0

```
100 events have been kept for refreshing and/or reviewing.
  "/vis/reviewKeptEvents" to review one by one.
  To see accumulated, "/vis/enable", then
  "/vis/viewer/flush" or "/vis/viewer/rebuild".
There are histograms that can be viewed with visualization:
  2 h1 histograms(s)
  2 h2 histograms(s)
  List them with "/analysis/list".
  View them immediately with "/vis/plot" or "/vis/reviewPlots".
```



User drawing (from C++ code)

- The **Draw** methods of the vis manager
  - Restrictions and limitations
- User vis actions
  - Treated like a model – refreshed as required
- Draw method of trajectories
- Draw methods of hits and digis

- See separate presentation

The future

- “Prediction is difficult, especially of the future” – unattributed quote!
- The Qt GUI will continue to be our “flagship” UI
- So far, OpenGL has been our “flagship” graphics driver
  - Some features, notably time-slicing (drawing by time), is available *only* with OpenGL
    - Time slicing uses display lists, an OpenGL-1 deprecated feature
  - Cutting is somewhat crude (uses clipping planes)
- ToolsSG parallels OpenGL in almost all respects
  - Quite light and zippy
  - As well as OpenGL-ES, it has a ZB driver (with its own z-buffer), including offscreen
    - Memory-based (non-GPU), thus cannot compete for performance, but OK
    - Requires only the ability to draw or dump a pixel map
  - Has plotting
- Vtk: now fully released, and being used
  - Very high performance
  - Interfaces to ParaView (and FreeCAD?)
  - Nice cutters and slicers
  - Needs manpower to develop into a truly useful driver
  - Can it do zoom-in-on and twinkling?
  - Can it do time-slicing?
- These – and other – drivers will continue to be offered as a choice for users

Discussion

- OpenGL
  - OpenGL-1 will probably be continued to be supported because of a huge amount of legacy code, including ours
  - So I recommend continuing our OpenGL drivers
    - To keep the very nice features, such as time-slicing
- ToolsSG
  - Parallels OpenGL in almost all respects
  - Needs no installation (all platforms have OpenGL support)
  - Offer off-screen rendering
  - Let's continue to invest in it
- Vtk
  - Has some really nice features
    - Excellent performace
    - Clipping and cutting
    - Export to ParaView and other nice applications
  - At present, Vtk in Geant4 appears to be somewhat specialist. So my question is: how much do we push it as a vis tool? There is an overhead, of course, namely the installation of Vtk libraries, followed by a rebuild of Geant4. On Mac, at least, that is simple and in my experience error free.
  - Let's continue to invest in it
- Open Inventor
  - Again, somewhat specialist
  - Not so easy to install
  - But worth continuing
- Fukui renderer/DAWN/DAVID
  - Still works
- VRML
  - Still works
- HepRep
  - Still works
  - But the browser, HepRApp, is no longer supported and works only on an old Java version
  - Deprecate?
- Qt3D
  - Deprecated by Qt, so we should do the same
- RayTracer
  - Essential, but needs support
- gMocren
  - Not supported
  - Deprecate?

Thankyou