Geant4 internal Classes and Objects introduction to CaTS

20 July 2012

Hans Wenzel Fermilab

User Action Classes

Contents

- Internal Objects
  - Run and Event
  - Track and Step
  - StepPoint
  - Dynamic Particle

- UserAction classes
  - Run and Event
  - Track and Step

- UserInformation classes
  - G4VUserEventInformation
  - G4VUserTrackInformation
  - G4VUserPrimaryVertexInformation
  - G4VUserPrimaryParticleInformation
  - G4VUserRegionInformation

Introduction (1)
Extract information from G4 internal objects

Simulation is successively split into	Corresponding / related Objects
Run consists of	G4RunManager, G4Run
Event(s), consists of	G4Event
Particle(s) transported in	G4Track, G4DynamicParticle
Steps through detector setup,	G4Step, G4StepPoint
depositing energy (ionization),	G4Trajectory
and creating secondaries	G4Stack

Introduction (2)

- User at each moment has possibility to take control or access information via UserAction classes

G4UserRunAction	Actions for each Run
G4UserEventAction	Actions for each Event
G4UserTrackingAction	Actions for each Track
G4UserSteppingAction	Actions for each Step
G4UserStackingAction	Tracks Stack management

Introduction (3)

- User can replace Geant4 classes by providing his own classes derived from the base classes:
  - G4Run
  - G4Trajectory
  - G4VTrajectoryPoint
- User can attach optional User Information classes to
  - G4Event
  - G4Track
  - G4PrimaryVertex
  - G4Region

RunManager in Geant4

- G4RunManager class manages processing a run
  - Must be created by user
  - May be user derived class
  - Must be singleton
- User must register in RunManager using
  - SetUserInitialization() method
    - Geometry
    - Physics
  - SetUserAction() method
    - Event generator
  - Optional UserAction objects

The Minimum Geant 4 Program

From CaTS.cc (main()):

```
runManager->SetUserInitialization(phys);
runManager->SetUserInitialization(new DetectorConstruction(argv[1]));
runManager->SetUserAction(new PrimaryGeneratorAction);

runManager->SetUserAction(new RunAction());
runManager->SetUserAction(new EventAction());
runManager->SetUserAction(new StackingAction());
runManager->Initialize();
```

Run in Geant4

- Run is a collection of events
  - A run consists of one event loop
  - Starts with a /run/beamOn command.
- Within a run, conditions do not change, i.e. the user cannot change
  - detector setup
  - settings of physics processes
- At the beginning of a run, geometry is optimized for navigation and cross-section tables are calculated according to materials appear in the geometry and the cut-off values defined.
- Run is represented by G4Run class or a user-defined class derived from G4Run.
  - A run class may have a summary results of the run.
- G4RunManager is the manager class
- G4UserRunAction is the optional user hook.

Optional User **Run** Action Class

- G4UserRunAction
  - G4Run* GenerateRun()
    - Instantiate user-customized run object
    - e.g. Output File
  - void BeginOfRunAction(const G4Run*)
    - Define histograms
  - void EndOfRunAction(const G4Run*)
    - Analyze the run
    - Store histograms
    - Close Output file

Event in Geant4

- An event is the basic unit of simulation in Geant4.
- At beginning of processing, primary tracks are generated. These primary tracks are pushed into a stack.
- A track is popped up from the stack one by one and “tracked”. Resulting secondary tracks are pushed into the stack.
  - This “tracking” lasts as long as the stack has a track.
- When the stack becomes empty, processing of one event is over.
- G4Event class represents an event. It has following objects at the end of its (successful) processing.
  - List of primary vertices and particles (as input)
  - Hits and Trajectory collections (as output)
- G4EventManager class manages processing an event.
- G4UserEventAction is the optional user hook.

Optional User **Event** Action Class

- G4UserEventAction
  - void BeginOfEventAction(const G4Event*)
    - Event selection
      - Using information from event generator, vertices, primary particles
      - Rest Event counters etc.
    - Optionally attach G4VUserEventInformation object
  - void EndOfEventAction(const G4Event*)
    - Output event information
    - Analyse event
      - Access to hits collection via G4Event::GetHCofThisEvent()
      - Acces digitisation collection via G4Event:: GetDCofThisEvent()
    - Fill histograms (with Event specific Info)

11

Track in Geant4

- Track is a snapshot of a particle.
  - It has physical quantities of current instance only. It does not record previous quantities.
  - Step is a “delta” information to a track. Track is not a collection of steps. Instead, a track is being updated by steps.
- Track object is deleted when
  - it goes out of the world volume,
  - it disappears (by e.g. decay, inelastic scattering),
  - it goes down to zero kinetic energy and no “AtRest” additional process is required, or
  - the user decides to kill it artificially.
- No track object persists at the end of event.
  - For the record of tracks, use trajectory class objects.
- G4TrackingManager manages processing a track, a track is represented by G4Track class.
- G4UserTrackingAction is the optional user hook.

Track status

- At the end of each step, according to the processes involved, the state of a track may be changed.
  - The user can also change the status in UserSteppingAction.
  - Statuses shown in blue are for users only, i.e. Geant4 kernel won't set them.
- **fAlive**
  - Continue the tracking.
- **fStopButAlive**
  - The track has come to zero kinetic energy, but still AtRest process to occur.
- **fStopAndKill**
  - The track no longer exists --it has decayed, interacted or gone out of the world boundary.
  - Secondaries will be pushed to the stack.
- **fKillTrackAndSecondaries**
  - Kill the current track and also associated secondaries.
- **fSuspend**
  - Suspend processing of the current track and push it and its secondaries to the stack.
- **fPostponeToNextEvent**
  - Postpone processing of the current track to the next event.
  - Secondaries are still being processed within the current event.

Tracking User Action Classes

- G4UserTrackingAction
  - void PreUserTrackingAction(const G4Track*)
    - Decide if trajectory should be stored or not
    - Create user-defined trajectory
  - void PostUserTrackingAction(const G4Track*)
    - Delete unnecessary trajectory

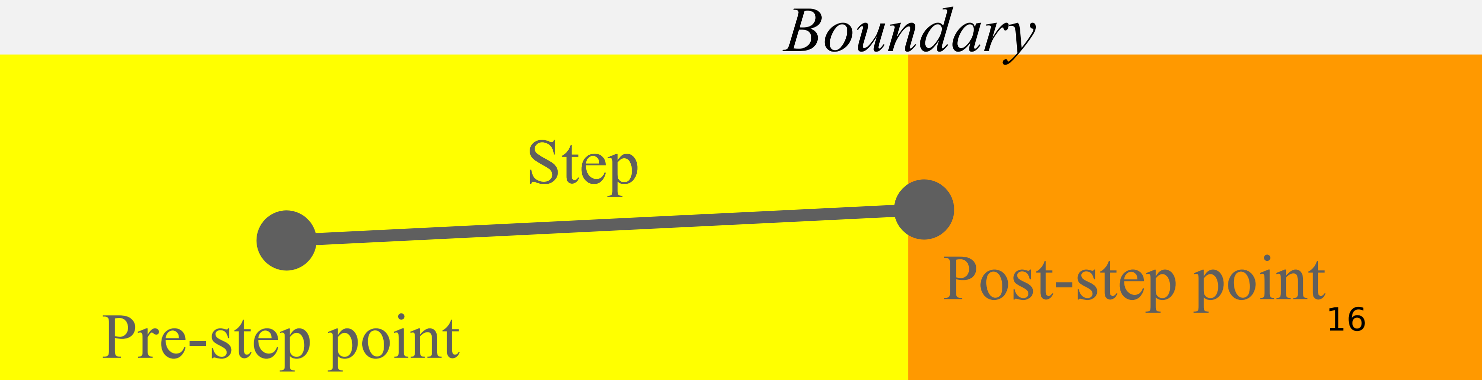
Stacking User Action Class

- G4UserStackingAction
  - G4ClassificationOfNewTrack ClassifyNewTrack(const G4Track*)
    - Invoked every time a new track is created, ie. Pushed to the stack
    - (CaTS e.g. to count pi0, neutrons, beta of charged tracks)
    - Classify a new track -- priority control
      - Urgent, Waiting, PostponeToNextEvent, Kill
  - Manipulate track stack,
    - void PrepareNewEvent()
      - Reset priority control
    - void NewStage()
      - Invoked when the Urgent stack becomes empty
      - Change the classification criteria
      - Event filtering (Event abortion)

15

Step in Geant4

- Step has two points and also “delta” information of a particle (energy loss on the step, time-of-flight spent by the step, etc.).
  - Point is represented by G4StepPoint class
- Each point knows the volume (and material). In case a step is limited by a volume boundary, the end point physically stands on the boundary, and it logically belongs to the next volume.
  - Because one step knows materials of two volumes, boundary processes such as transition radiation or refraction could be simulated.
- G4SteppingManager class manages processing a step, a step is represented by G4Step class.
- G4UserSteppingAction is the optional user hook.



Stepping User Action Class

- G4UserSteppingAction
  - void UserSteppingAction(const G4Step*)
    - Change status of track
      - Kill / suspend / postpone the track
    - Draw the step (for a track not to be stored as a trajectory)

17

Trajectory and trajectory point (1)

- Track does not keep its trace. No track object persists at the end of event.
- G4Trajectory is the class which copies some of G4Track information.
- G4TrajectoryPoint is the class which copies some of G4Step information.
  - G4Trajectory has a vector of G4TrajectoryPoint objects.
  - At the end of event processing, G4Event has a collection of G4Trajectory objects.
    - /tracking/storeTrajectory must be set to 1.
- G4Trajectory and G4TrajectoryPoint objects persist till the end of an event
  - Be careful not to store too many trajectories, memory growth.
    - E.g. avoid for high energy EM shower tracks.

Trajectory and trajectory point (2)

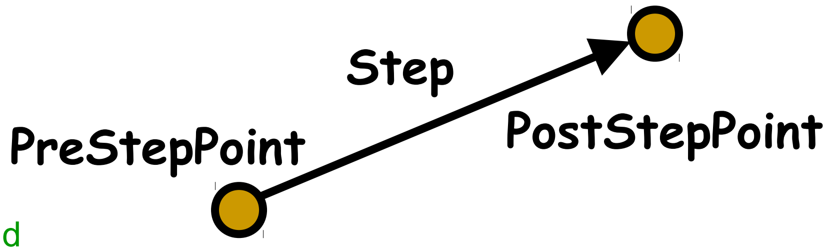
- Keep in mind the distinct classes conceptually corresponding
  - G4Track ← → G4Trajectory
  - G4Step ← → G4TrajectoryPoint
- G4Trajectory and G4TrajectoryPoint as provided by Geant4 store only the minimum information.
  - You can create your own trajectory / trajectory point classes to store information you need.
  - User classes must be derived from G4VTrajectory and G4VTrajectoryPoint base classes.
    - Do not use G4Trajectory nor G4TrajectoryPoint concrete class as base classes unless you are sure to never ever add any additional data member.

StepPoint in Geant4

- Two step point objects attached to step
  - Pre-step point and post-step point
- G4StepPoint has information of track representing a particle at this point
  - Time (global event time, local, proper time since creation of particle
  - Position, kinetic energy, momentum
  - Material
  - …

Step status

- Step status is attached to G4StepPoint to indicate why that particular step was determined.
  - Use “PostStepPoint” to get the status of this step.
  - “PreStepPoint” has the status of the previous step.



- **fWorldBoundary**
  - Step reached the world boundary
- **fGeomBoundary**
  - Step is limited by a volume boundary except the world
- **fAtRestDoItProc**, **fAlongStepDoItProc**, **fPostStepDoItProc**
  - Step is limited by a AtRest, AlongStep or PostStep process
- **fUserDefinedLimit**
  - Step is limited by the user Step limit
- **fExclusivelyForcedProc**
  - Step is limited by an exclusively forced (e.g. shower parameterization) process
- **fUndefined**
  - Step not defined yet

- If you want to identify the first step in a volume, pick fGeomBoudary status in PreStepPoint.
- If you want to identify a step getting out of a volume, pick fGeomBoundary status in PostStepPoint

Recap – User action classes

- All needed UserAction classes
  - must be constructed in main()
  - must be provided to the RunManager using SetUserAction() method
- One mandatory User Action class
  - Event generator must be provided
  - Event generator class must be derived from G4VUserPrimaryGeneratorAction
- List of optional User Action classes
  - G4UserRunAction
  - G4UserEventAction
  - G4UserTrackingAction
  - G4UserSteppingAction
  - G4UserStackingAction

CaTS: Calorimeter and Tracker Simulation

CaTS is a flexible and extend-able framework (based on geant4 and ROOT) for the simulation of calorimeter and tracking detectors. To be able to simulate Dual Read out calorimeters it provides special sensitive detectors and Hit classes that register both the energy deposit and the number of Cerenkov photons produced by particles above the Cerenkov threshold.

CaTS also allows the detailed study of single Calorimeter cells by enabling the tracing of optical photons, providing sensitive detectors that register optical photons and the gdml detector description allows to provide all relevant optical properties (refraction Index, Absorption length, Scintillation Yield,

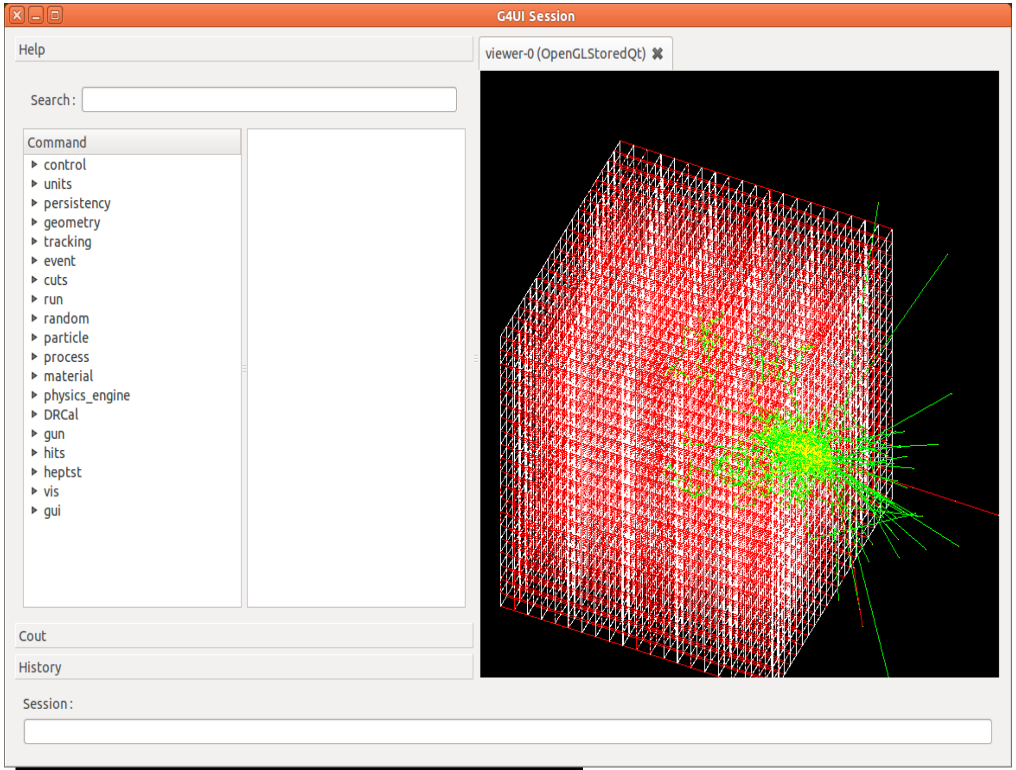
Rayleigh scattering length, Surface properties (e.g. Reflectivity)....)

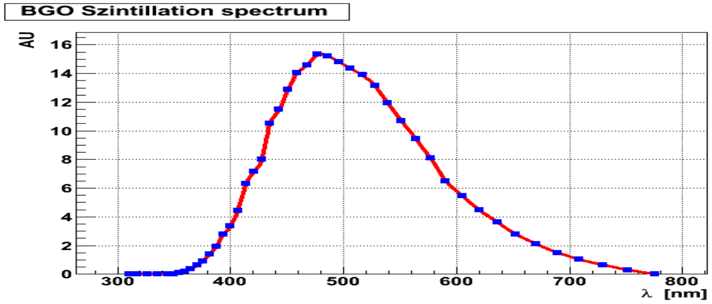
Elements of CaTS

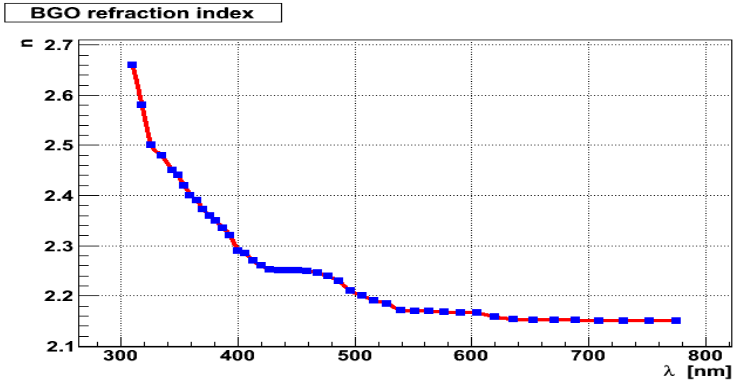
Detector Description:	Xml based gdml input file (e.g. crystalcal.gdml) (Geometry, Materials, optical properties, sensitive detector), we provide working examples
Persistency	uses Root reflexion (gccxml) to automatically, create dictionaries for all Hit classes
Input modules:	GPS, Particle Gun, HEPMC (Pythia)
Physics Lists:	choice of all Reference Physics Lists which can be extended to include optical physics processes (Cerenkov, Rayleigh, Scintillation etc.)
Sensitive Detectors and Hits:	TrackerSD, CalorimeterSD, DRCalorimeterSD (also registers Cerenkov photons), StoppingCalorimeterSD, PhotonSD: sensitive detector that registers optical photons.
User Actions:	examples of user actions (EventAction, RunAction, StackingAction,SteppingAction...) are provided
CVS Code repository & Instructions:	http://cdcvs.fnal.gov/cgi-bin/public-cvs/cvsweb-public.cgi/?hidenonreadable=1&f=h&logsort=date&sortby=file&hideattic=1&cvsroot=ilcdet http://home.fnal.gov/~wenzel/cvs.html#Optical

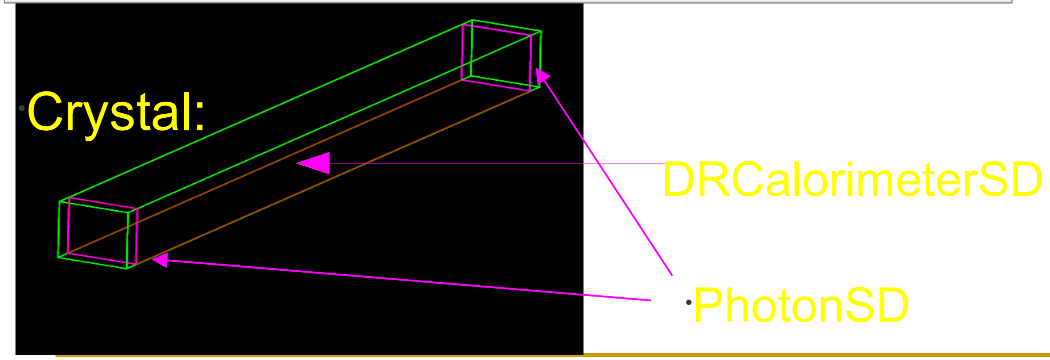
CaTS in Action

Calorimeter and Tracker Simulation









- Crystal:

DRCalorimeterSD

- PhotonSD

How do we control CaTS?

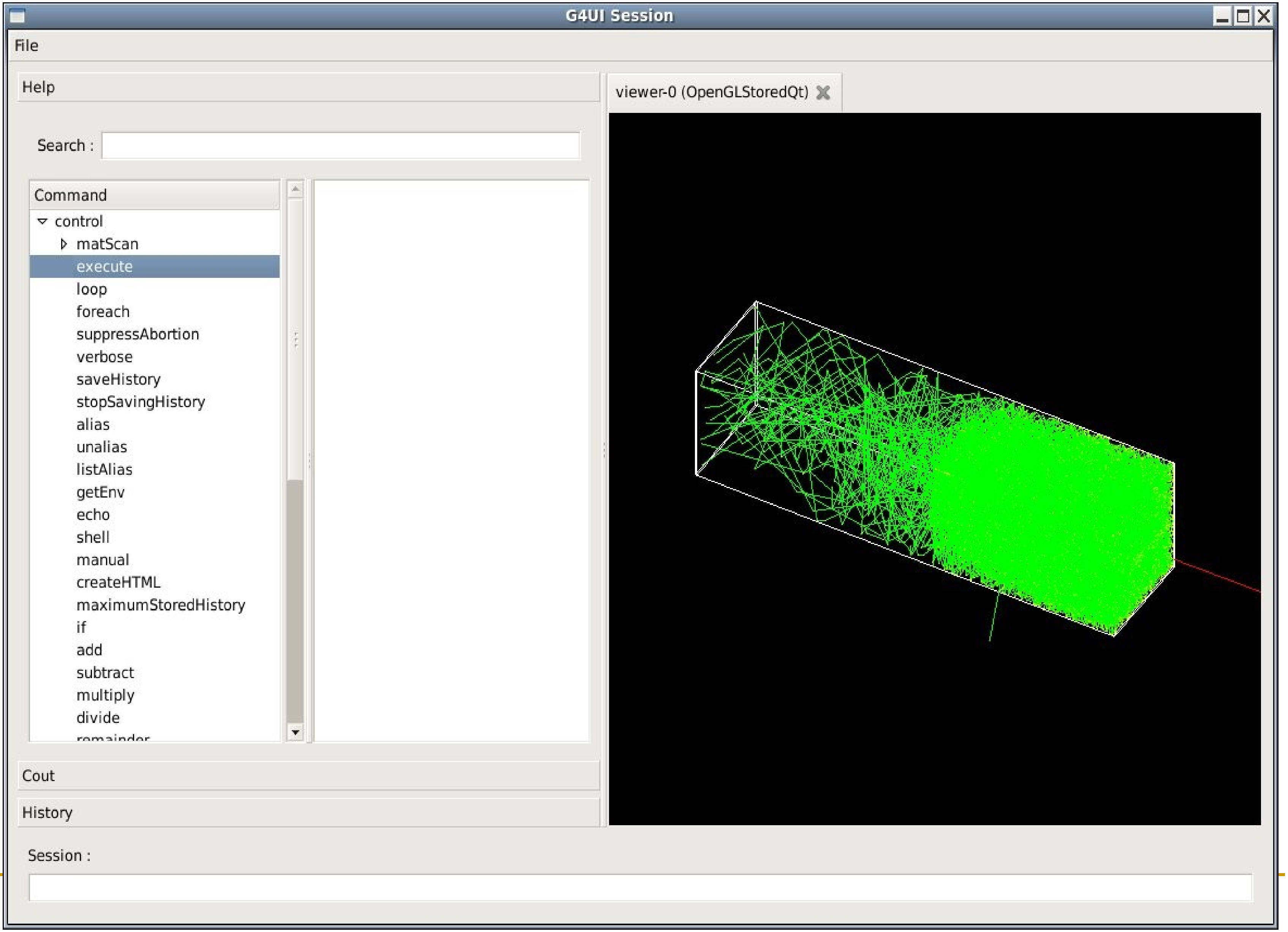
```
./CaTS ../CaTS/gdml/crystalcal.gdml (electron.mac)
```

Most of the control via messenger classes → just type help on the command line to see what is available. But:

For historical reasons some control still via environmental variables,

Will create messanger classes for that when I get the chance

```
export PHYSLIST=QGSP_BERT    (Default FTFP_BERT)
export ENABLEOPTICAL=1         (Default OFF)
export ENABLESCINTILLATION=1  (Default Off)
```


G4UI Session
File
Help
Search :
Command
control
matScan
execute
loop
foreach
suppressAbortion
verbose
saveHistory
stopSavingHistory
alias
unalias
listAlias
getEnv
echo
shell
manual
createHTML
maximumStoredHistory
if
add
subtract
multiply
divide
viewer-0 (OpenGLStoredQt)
Cout
History
Session :