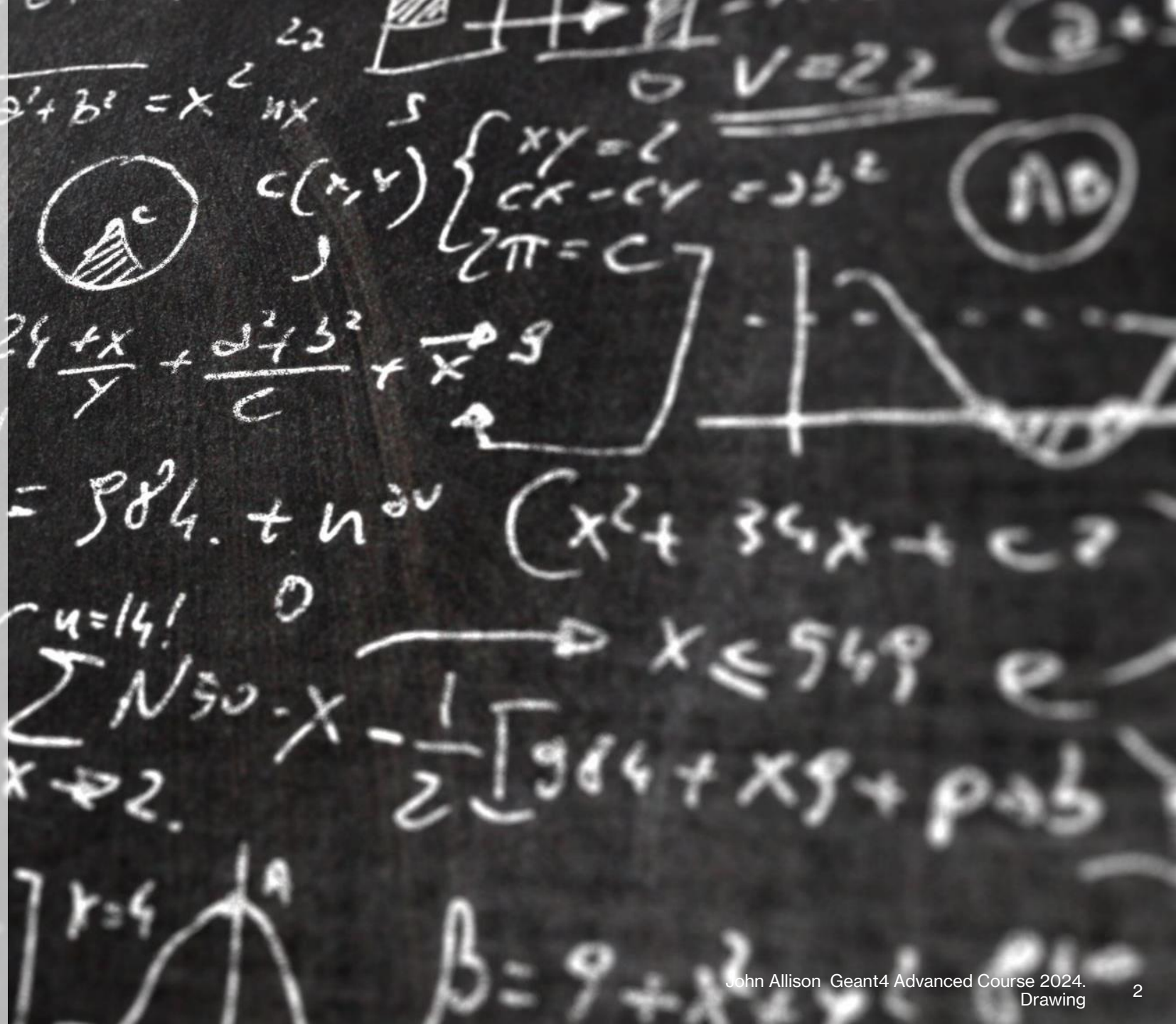

# Geant4 Visualisation: User drawing from C++

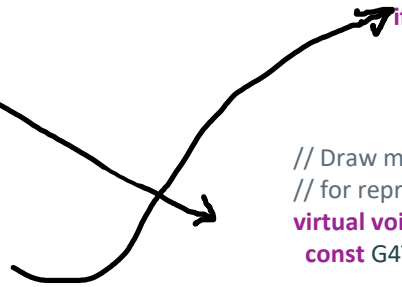
# USER DRAWING (FROM C++ CODE)

- 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



# The Draw methods

- The user interacts with the vis manager through the abstract interface **G4VVismanager** in the low level library **graphics_reps**.
  - It has *many* **Draw** methods – 18 at the last count – for different types
  - Also **DrawGeometry**
  - The header files are copiously commented
- They must be protected by validating the pointer to the vis manager
  - **G4VVisMananger::GetConcreteInstance()**
  - There might be no vis manager
    - But your code will still compile
  - This is also used to switch vis off
- They are inhibited if called from a worker thread in multithreaded mode (e.g., stepping action)
  - So use Serial mode if you want to draw during a run
- But you can add trajectories and hits to the scene, and write user vis actions (next slide), even in Multithreading/Tasking mode, or write your own **Draw** methods of the trajectory or hits classes
  - This is because the vis manager then can control when to draw
- Optional **Begin/EndDraw** methods can improve speed



```
G4VVisManager* pVVisManager = G4VVisManager::GetConcreteInstance();
if (pVVisManager) {
```

```
// Draw methods for Geant4 Visualization Primitives, useful
// for representing hits, digis, et
virtual void Draw (const G4Circle&,
  const G4Transform3D& objectTransformation = G4Transform3D()) = 0;
...
virtual void Draw2D (const G4Polyline&,
  const G4Transform3D& objectTransformation = G4Transform3D()) = 0;
...
// Draw methods for Geant4 Objects as if they were Visualization
// Primitives.  Note that the visualization attributes needed in
// some cases override any visualization attributes that are
// associated with the object itself - thus you can, for example,
// change the colour of a physical volume.
virtual void Draw (const G4VTrajectory&) = 0;
virtual void Draw (const G4VHit&) = 0;
virtual void Draw (const G4VDigi&) = 0;
virtual void Draw (const G4LogicalVolume&, const G4VisAttributes&,
  const G4Transform3D& objectTransformation = G4Transform3D()) = 0;
virtual void Draw (const G4VPhysicalVolume&, const G4VisAttributes&,
  const G4Transform3D& objectTransformation = G4Transform3D()) = 0;
...
   virtual void DrawGeometry
(G4VPhysicalVolume*, const G4Transform3D& t = G4Transform3D());
// Draws a geometry tree starting at the specified physical volume.
```

# User vis actions

- **/examples/extended/visualization/userVisAction**

```
G4VisManager* visManager = new G4VisExecutive;
// Register User Vis Action with optional extent
visManager->RegisterRunDurationUserVisAction(
 "My nice logo", new UVA_VisAction,
 G4VisExtent(-20 * cm, -10 * cm, -25 * cm, -15 * cm, 20 * cm, 40 * cm));
visManager->Initialize();
```

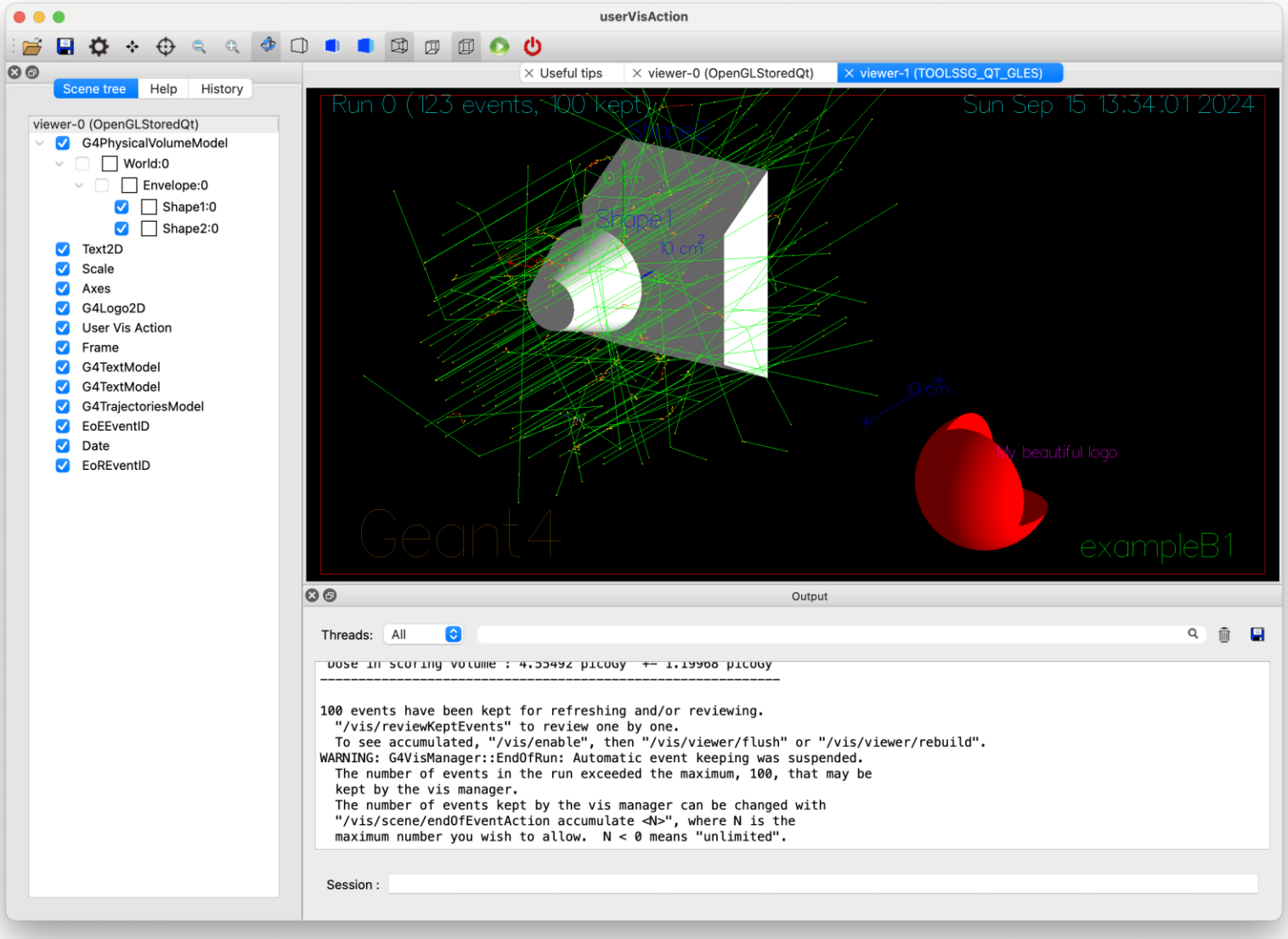
```
#ifndef UVA_VISACTION_HH
#define UVA_VISACTION_HH
#include "G4VUserVisAction.hh"
class UVA_VisAction : public G4VUserVisAction
{
 virtual void Draw();
};
#endif
```

```
void UVA_VisAction::Draw()
{
 G4VVisManager* pVisManager = G4VVisManager::GetConcreteInstance();
 if (pVisManager) {
  // A simple logo...
  G4Orb orb("my_logo_orb", 5 * cm);
  G4Box box("my_cut_box", 5 * cm, 5 * cm, 5 * cm);
  G4SubtractionSolid logo("my_logo", &orb, &box, G4Translate3D(-3 * cm, 3 * cm, 3 * cm));
  G4VisAttributes va1(G4Colour::Red());
  va1.SetForceSolid(true);
  pVisManager->Draw(logo, va1, G4Translate3D(-15 * cm, -20 * cm, 25 * cm));

  G4Text text("My beautiful logo");
  G4VisAttributes va2(G4Colour::Magenta());
  text.SetVisAttributes(va2);
  text.SetScreenSize(12.);
  pVisManager->Draw(text, G4Translate3D(-16 * cm, -18 * cm, 25 * cm));
 }
}
```

# User vis actions (contd)

- Each user vis action gets turned into a model
  - Demo
- User vis actions are better than using direct draw methods
  - They are automatically re-drawn as required
    - They can be refreshed
    - Opened with any driver
      - Including file writing drivers (to get a pdf, etc.)



# User vis actions (contd)

- **DrawGeometry** can be used in a vis action
  - Do as much as you can in the constructor (once and for all)
- Note: this is part of **/examples/extended/visualization/standalone**
  - See next slide

```
DrawGeometryVisAction::DrawGeometryVisAction()
{
 // Get a physical volume from your detector construction
 fDetectorConstruction = new B1::DetectorConstruction();
 fPhysicalVolume = fDetectorConstruction->Construct();
 // Give this an overall transform to avoid clash with other vis
 // action(s) in this case
 fTransform = G4Translate3D(-20 * cm, 20 * cm, 0);
 G4PhysicalVolumeModel pvModel(fPhysicalVolume);
 fExtent = pvModel.GetExtent();
 fExtent.Transform(fTransform);
}

DrawGeometryVisAction::~DrawGeometryVisAction()
{delete fDetectorConstruction;}

void DrawGeometryVisAction::Draw()
{
 G4VVisManager* pVisManager = G4VVisManager::GetConcreteInstance();
 if (pVisManager) {
  pVisManager->DrawGeometry(fPhysicalVolume, fTransform);
 }
}
```

# Standalone

- For debugging geometry you can build an app *without* a run manager and all its required accompaniments (physics lists, etc.)
  - Useful when run initialization (physics tables, etc.) takes a long time
    - E.g., medical examples
      - ICRP examples have a standalone version

```
int main(int argc, char** argv)
{
 G4UIExecutive* ui = new G4UIExecutive(argc, argv);
 G4VisManager* visManager = new G4VisExecutive;
 visManager->Initialize();

 auto standaloneVisAction = new StandaloneVisAction;
 visManager->RegisterRunDurationUserVisAction
 ("A standalone example - 3 boxes, 2 with boolean subtracted cutout",
  standaloneVisAction,
  G4VisExtent(-10 * cm, 10 * cm, -10 * cm, 10 * cm, -10 * cm, 10 * cm));

 auto geometryVisAction = new DrawGeometryVisAction;
 visManager->RegisterRunDurationUserVisAction
   ("A detector geometry", geometryVisAction,
    geometryVisAction->GetVisxtent());

 G4UImanager::GetUIpointer()->ApplyCommand("/control/execute standalone.mac");
 ui->SessionStart();

 delete geometryVisAction;
 delete standaloneVisAction;
 delete visManager;
 delete ui;
}
```

# Trajectory drawing

- Trajectories have a draw method, **DrawTrajectory()**
- The default implementation dispatches the trajectory to the vis manager so that it can use the provided modeling and filtering

```
void G4VTrajectory::DrawTrajectory() const {
  ...
    pVVisManager->DispatchToModel(*this);
```

- But, if you write your own trajectory class, you can override with your own **DrawTrajectory()** – see **examples/extended/runAndEvent/RE01**

# Hit and digi drawing

- **G4VHit::Draw()** and **G4VDigi::Draw()**
  - Empty default functions
    - The user may provide in the concrete derived class
      - The vis manager will draw on request
        - **/vis/scene/add hits**
        - **/vis/scene/add/digis**
  - The user may also provide **GetAttDefs()** and **CreateAttValues()**

# End of Drawing from C++