

Geant 4

March 2006, Geant4 v8.0p01

# Trajectory Visualisation

---

Jane Tinslay

SLAC

Stanford  
Linear  
Accelerator  
Center

# Outline

---

- Visualising trajectories
- Enhanced trajectory drawing
- Models supplied with Geant4
- Model configuration
- Implementing user defined models
- Future development
- Information resources

# Visualising Trajectories

---

- Trajectory are not visualised by default
- Possible to manipulate trajectory drawing through:
  - Interactive commands
    - `/vis/scene/add/trajectories <drawing-mode>`
    - Draws trajectories using enhanced trajectory drawing model functionality
  - Using enhanced trajectory drawing directly in compiled code
  - Adding trajectory drawing in EndOfEventAction method
  - Sub-classing G4VTrajectory and implementing DrawTrajectory method
- Rest of this talk will focus on enhanced trajectory drawing
  - Intended to superceed other two methods mentioned above
  - Hopefully meets most peoples needs

# Enhanced Trajectory Drawing

---

- First released in Geant4 v8.0
- Introduces concept of a trajectory model:
  - Represents a particular style of drawing trajectories
  - Possible to have a whole range of models, for example
    - Colour trajectories according to charge
    - Colour trajectories according to particle type
- Enhanced trajectory drawing supports
  - Use and configuration of models supplied with Geant4 distribution
  - Interactive and compile time model configuration
  - User defined models
- Flexible and extendable

# Models Supplied with Geant4

- Two models are supplied with Geant4 v8.0 release
  - More are being developed for future releases
- G4TrajectoryDrawByParticleID (**drawByParticleID**)
  - Colours trajectories according to particle type
  - All trajectories are coloured grey by default
  - Configure to highlight chosen particle types with chosen colours
  - No limit on the number of particle types that can be highlighted
- G4TrajectoryDrawByCharge (**drawByCharge**)
  - Default model
  - Colours trajectories according to charge

|    |       |
|----|-------|
| +1 | Blue  |
| -1 | Red   |
| 0  | Green |

# Model Configuration

---

- Create and configure multiple models through either
  - Interactive commands
  - Compiled code
- Interactive commands
  - Located in `/vis/modeling/trajectories` directory
  - Possible to have multiple instances of given model type
  - Configuration commands generated dynamically when model created
    - Commands apply directly to that instance
  - List and select instantiated models with commands:
    - `/vis/modeling/trajectories/list`
    - `/vis/modeling/trajectories/select <model-instance-name>`

## Example macro

```
#Create a drawByCharge model named drawCharge-0 by
#default. Subsequent models will be named drawByCharge-1...
/vis/modeling/trajectories/create/drawByCharge

#Create a drawByCharge model named testChargeModel
/vis/modeling/trajectories/create/drawByCharge testChargeModel

#Configure drawByCharge-0 model through colour string arguments
/vis/modeling/trajectories/drawByCharge-0/set 1 red
/vis/modeling/trajectories/drawByCharge-0/set -1 red
/vis/modeling/trajectories/drawByCharge-0/set 0 white

#Configure testCharge model through G4Colour components
/vis/modeling/trajectories/testChargeModel/setRGBA 1 0 1 1 1
/vis/modeling/trajectories/testChargeModel/setRGBA -1 0.5 0.5 0.5 1
/vis/modeling/trajectories/testChargeModel/setRGBA 0 1 1 0 1

#Use drawByCharge-0 model
/vis/modeling/trajectories/select drawByCharge-0
/run/beamOn 1
#use testChargeModel
/vis/modeling/trajectories/select testChargeModel
/run/beamOn 1
```

**Creation**

**Configuration**

**Selection and  
execution**

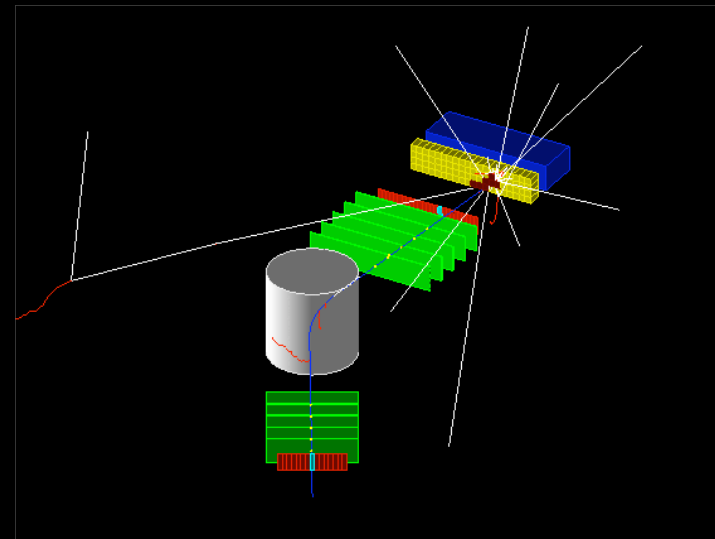
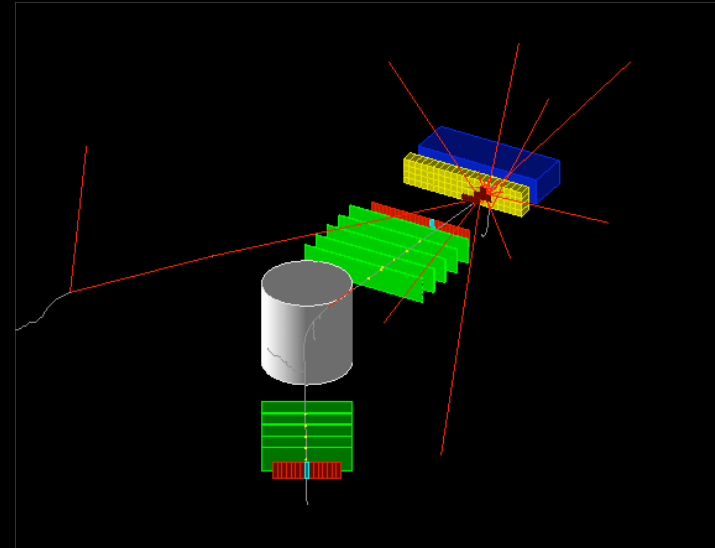
## ■ examples/extended/analysis/A01

### Example macro

```
#Standard setup
/vis/scene/create
/vis/open OGLIX
/vis/scene/add/volume
/vis/scene/add/trajectories
/vis/scene/add/hits
/vis/viewer/set/lightsThetaPhi 90. 0.
/vis/viewer/set/viewpointThetaPhi 150. 90.
/vis/viewer/set/style surface
/vis/viewer/set/hiddenEdge true
#Create drawByParticleID model, highlighting photons
/vis/modeling/trajectories/create/drawByParticleID
/vis/modeling/trajectories/drawByParticleID-0/set gamma red
/run/beamOn 1

...

#Create drawByCharge model, colouring photons white
/vis/modeling/trajectories/create/drawByCharge
/vis/modeling/trajectories/drawByCharge-0/set 1 blue
/vis/modeling/trajectories/drawByCharge-0/set -1 red
/vis/modeling/trajectories/drawByCharge-0/set 0 white
/run/beamOn 1
```



- Compiled code
  - Instantiate model
  - Configure model
  - Register with visualisation manager

#### main.cc

```
// Create and initialise visualisation manager
G4VisManager* visManager = new G4VisExecutive;
visManager->Initialize();

// Create new drawByParticleID model
G4TrajectoryDrawByParticleID* model =
    new G4TrajectoryDrawByParticleID;

// Configure model
model->SetDefault("cyan");
model->Set("gamma", "green");
model->Set("e+", "magenta");
model->Set("e-", G4Colour(0.3, 0.3, 0.3));

//Register model with visualisation manager
visManager->RegisterModel(model);
```

# Implementing User Defined Models

---

- New trajectory models must inherit from `G4VTrajectoryModel` and implement these pure virtual methods:

```
virtual void Draw(const G4VTrajectory&, G4int i_mode = 0) const = 0;  
virtual void Print(std::ostream& ostr) const = 0;
```

- New models can be used directly in compiled code
  - Need to be registered with visualisation manager

```
main.cc
```

```
// Create custom model  
MyCustomTrajectoryModel* myModel = new MyCustomTrajectoryModel("custom");  
  
// Configure it if necessary and then register with G4VisManager  
...  
visManager->RegisterModel(myModel)
```

- Additional classes need to be written if want to be able to create and configure the model interactively
- Messenger classes
  - Messengers to configure the model should inherit from G4VModelCommand. The concrete trajectory model type should be used for the template parameter

#### **G4ModelCommandDrawByParticleIDSet.cc**

```
class G4ModelCommandDrawByParticleIDSet : public
    : G4VModelCommand<G4TrajectoryDrawByParticleID> {
    ...
};
```

- Factory class
  - A factory class responsible for the model and associated messenger creation must also be written. The factory should inherit from G4VModelFactory. The abstract model type should be used for the template parameter, eg:

#### **G4TrajectoryDrawByChargeFactory.cc**

```
class G4TrajectoryDrawByChargeFactory
    : public G4VModelFactory<G4VTrajectoryModel> {
    ...
};
```

- Construct model and associated messengers in Create method

#### **G4TrajectoryDrawByParticleIDFactory.cc**

```
ModelAndMessengers
G4TrajectoryDrawByParticleIDFactory::Create(const G4String& placement, const G4String& name) {
// Create model with given name
G4TrajectoryDrawByParticleID* model = new G4TrajectoryDrawByParticleID(name);

// Create associated messengers with commands in ``placement'' command directory.
Messengers messengers;
messengers.push_back(new G4ModelCommandDrawByParticleIDSet(model, placement));
...
return ModelAndMessengers(model, messengers);
}
```

- Factory must then be registered with visualisation manager

#### **G4VisExecutive.cc**

```
G4VisExecutive::RegisterModelFactories() {...
  RegisterModelFactory(new G4TrajectoryDrawByParticleIDFactory());
}
```

# Future Development

---

- New models planned in near future:
  - **drawByMomentum**
    - Colour indicates particle momentum
  - **drawByOriginLogicalVolume**
    - Colour indicates where particle originated
  - **drawByOriginInteraction**
    - Colour indicates what interaction created particle
  - etc

# Information Resources

---

- For information on commands, interactive guidance is useful
- See Application Developers users guide for information on how to use enhanced trajectory drawing:

<http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/ForApplicationDeveloper/html/index.html>

- See Toolkit Developers guide for information on how to write your own models:

<http://geant4.web.cern.ch/geant4/G4UsersDocuments/UsersGuides/ForToolkitDeveloper/html/index.html>