GEANT4: STFC Summer School

Contents

Introduction

This is intended as a brief resource containing many of the useful functions in GEANT4.

1 Geometries

G4Tubs defines a cylinder or cylindrical section:

```
G4Tubs(const G4String& pName, // Name of your geometry
          G4double  pRMin, // Inner radius of the cylinder
          G4double  pRMax, // Outer radius of the cylinder
          G4double  pDz, // Half-length of the cylinder
          G4double  pSPhi, // Minimum phi (= 0 for full cylinder)
          G4double  pDPhi // Maximum phi (= 360 * degree for full)
          )
```

G4Box defines a cuboid:

```
G4Box(const G4String& pName, // Name of geometry
         G4double  pX, // Half-length in x
         G4double  pY, // Half-length in y
         G4double  pZ // Half-length in z
         )
```

G4Cons defines a cone or conical section:

```
G4Cons(const G4String& pName, // Name of geometry
          G4double  pRmin1, // Inner radius at one end
          G4double  pRmax1, // Outer radius at one end
          G4double  pRmin2, // Inner radius at other end
          G4double  pRmax2, // Outer radius at other end
          G4double  pDz, // Half-length of the cone
          G4double  pSPhi, // Minimum phi (= 0 for full cone)
          G4double  pDPhi // Maximum phi (= 360 * degree for full)
          )
```

Once you have defined your geometry, you need to create a Logical Volume, which defines the material properties of the geometry. Note that you can create multiple Logical Volumes from the same geometric shape.

```
G4LogicalVolume( G4VSolid*              pSolid,
         // The geometric object you defined
         G4Material*           pMaterial,
         // The G4Material to make it out of
         const G4String&       Name,
         // The name of the Logical Volume
         )
```

Finally, you need to place your Logical Volume in space. To do this, you create a third volume, the Physical Volume, and place it:

```
G4PVPlacement(   G4RotationMatrix*  pRot,
         // Defines any rotation you want to apply
         const G4ThreeVector&     tlate,
         // Defines any translation you want to apply
         G4LogicalVolume*   pCurrentLogical,
         // The logical volume you wish to place
         const G4String&    pName,
         // The name of the Physical Volume
         G4LogicalVolume*   pMotherLogical,
         // The logical volume that this will be placed inside
         G4bool             pMany,
         // Not relevant, set to false
         G4int              pCopyNo,
         // Identifies the placement
         G4bool             pSurfChk=false
         // check for overlaps with existing volumes
         )
```

Other geometry definitions can be found here.
It is also convenient here to define what are called "scoring volumes", which refer to geometric volumes for which you wish to record data later. These are simply G4LogicalVolume instances that can be compared against later.

2 Materials

GEANT4 already contains a huge number of pre-defined materials. These can be accessed from NIST. A list can be found here. To access these:

```
G4NistManager *nist = G4NistManager::Instance();
G4Material *material = nist->FindOrBuildMaterial("G4_Air");
// loads Air, as defined by NIST
```

It is possible to define your own materials, based on elemental fractions:

```
G4Element  *el = nist->FindOrBuildElement(
            G4int Z,  // Element Z
            G4bool isotopes=true //
            );
G4Material *material = new G4Material(
            const G4String& mName, // Material name
            G4double mDen, // Density of material
            G4int nEl // Number of elements in the material
```

```
            );
material->AddElement(
            G4Element* el, // Element
            G4int natoms // Number of atoms of that element
            )
```

You can also define a new element with different isotopic abundances - not included here.

3 Source definitions

There are two typical sources of radiation in GEANT4, the G4ParticleGun (which is the generic option) and the G4GeneralParticleSource. The G4GeneralParticleSource is a useful option for defining complicated sources of radiation in macro form. The various options can be found here.

The G4ParticleGun provides the standard route to creating radiation, and also has a lot of flexibility. This is handled in the G4VUserPrimaryGeneratorAction class, an example of which (from here) is duplicated below:

```
//////////////////////////////////
// PrimaryGeneratorAction.hh
//////////////////////////////////

#ifndef PrimaryGeneratorAction_h
#define PrimaryGeneratorAction_h 1

#include "G4VUserPrimaryGeneratorAction.hh"
#include "G4ThreeVector.hh"
#include "globals.hh"

class G4ParticleGun;
class G4Event;

class PrimaryGeneratorAction : public G4VUserPrimaryGeneratorAction
{
  public:
    PrimaryGeneratorAction(
      const G4String& particleName = "geantino",
      G4double energy = 1.*MeV,
      G4ThreeVector position= G4ThreeVector(0,0,0),
      G4ThreeVector momentumDirection = G4ThreeVector(0,0,1));
    ~PrimaryGeneratorAction();

    // methods
    virtual void GeneratePrimaries(G4Event*);

  private:
    // data members
    G4ParticleGun*  fParticleGun; //pointer a to G4 service class
};

#endif

//////////////////////////////////
// PrimaryGeneratorAction.cc
//////////////////////////////////
```

```
#include "PrimaryGeneratorAction.hh"

#include "G4Event.hh"
#include "G4ParticleGun.hh"
#include "G4ParticleTable.hh"
#include "G4ParticleDefinition.hh"

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo
    ......

PrimaryGeneratorAction::PrimaryGeneratorAction(
                                const G4String& particleName,
                                G4double energy,
                                G4ThreeVector position,
                                G4ThreeVector momentumDirection)
  : G4VUserPrimaryGeneratorAction(),
    fParticleGun(0)
{
  G4int nofParticles = 1;
  fParticleGun  = new G4ParticleGun(nofParticles);

  // default particle kinematic
  G4ParticleTable* particleTable = G4ParticleTable::GetParticleTable();
  G4ParticleDefinition* particle
    = particleTable->FindParticle(particleName);
  fParticleGun->SetParticleDefinition(particle);
  fParticleGun->SetParticleEnergy(energy);
  fParticleGun->SetParticlePosition(position);
  fParticleGun->SetParticleMomentumDirection(momentumDirection);
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo
    ......

PrimaryGeneratorAction::~PrimaryGeneratorAction()
{
  delete fParticleGun;
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo
    ......

void PrimaryGeneratorAction::GeneratePrimaries(G4Event* anEvent)
{
  // this function is called at the beginning of event

  fParticleGun->GeneratePrimaryVertex(anEvent);
}

//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo
    ......
```

4 Recording information

Typically, you will want to record some kind of information in GEANT4, such as energies, times, positions, etc. To do this, it is useful to know the structure of operations. Typically, the "lowest" element you will interact with is the Stepping, typically controlled through SteppingAction.cc and SteppingAction.hh. The Stepping relates to individual steps within the MC process, and is usually where you will begin the process of recording energies, times, etc, and passing them up the chain.

The typical element you care about above the Stepping action is the Event. This is defined for every "beam" event (**not** every particle, in the case of multiple particles being created). Typically, this will be controlled through EventAction.cc and EventAction.hh. Here, you can bring together the information from the individual Steps into a single record for an Event.

Finally, at the upper end is a GEANT4 Run, controlled by a RunManager (typically RunManager.cc), containing all of the events. An instance of the run is created for every thread, and then merged later. This is where you will pull all of your individual event information together for printing and/or writing to file.

The best way to handle outputting data is to use the G4AnalysisManager and G4Accumulable classes, which provide a thread-safe method for handling data retention and saving.

Examples of how to handle these can be found in the circulated examples.

5 Macros

In general, we don't want to recompile every time we change some small detail of the source, for example. Instead, GEANT4 uses a macro system. These files contain information for controlling the simulation through "Messenger" classes. As you develop your simulations, you can write your own Messenger class to control things like detector geometries and positions, without having to recompile.

A subset of macro commands to control the G4GeneralParticleSource is contained below. Duplicated from the full list here.

```
/gps/particle name # Defines the particle type [default geantino], using
   Geant4 naming convention.

/gps/direction Px Py Pz #Set the momentum direction [default (1,0,0)] of
   generated particles using

/gps/energy E unit #Sets the energy [default 1 MeV] for mono-energetic
   sources. The units can be eV, keV, MeV, GeV, TeV or PeV

/gps/position X Y Z unit #Sets the centre co-ordinates (X,Y,Z) of the
   source [default (0,0,0) cm]. The units can be micron, mm, cm, m or km.
   (NB: it is recommended to use /gps/pos/centre instead.)

/gps/ion Z A Q E #After /gps/particle ion, sets the properties (atomic
   number Z, atomic mass A, ionic charge Q, excitation energy E in keV) of
    the ion.

/gps/number N #Sets the number of particles [default 1] to simulate on
   each event.

/gps/pos/type dist #Sets the source positional distribution type: Point [
   default], Plane, Beam, Surface, Volume.
```

```
/gps/pos/shape shape #Sets the source shape type, after /gps/pos/type has
   been used. For a Plane this can be Circle, Annulus, Ellipse, Square,
   Rectangle. For both Surface or Volume sources this can be Sphere,
   Ellipsoid, Cylinder, Para (parallelepiped).

/gps/pos/centre X Y Z unit #Sets the centre co-ordinates (X,Y,Z) of the
   source [default (0,0,0) cm]. The units can only be micron, mm, cm, m or
    km.

/gps/ang/type AngDis #Sets the angular distribution type (iso [default],
   cos, planar, beam1d, beam2d, focused, user) to either isotropic, cosine
   -law or user-defined.

/gps/ene/type EnergyDis #Sets the energy distribution type to one of (see
   Table Table 1): Mono (mono-energetic, default), Lin (linear), Pow (
   power-law), Exp (exponential), Gauss (Gaussian), Brem (bremsstrahlung),
    Bbody (black-body), Cdg (cosmic diffuse gamma-ray), User (user-defined
    histogram), Arb (point-wise spectrum), Epn (energy-per-nucleon
   histogram)

/gps/ene/min Emin unit #Sets the minimum [default 0 keV] for the energy
   distribution. The units can be eV, keV, MeV, GeV, TeV or PeV.

/gps/ene/max Emax unit #Sets the maximum [default 0 keV] for the energy
   distribution. The units can be eV, keV, MeV, GeV, TeV or PeV.
```