Geant 4

October 2006, Geant4 v8.1p01

Geant4 Physics Based Event Biasing

Jane Tinslay, SLAC

Stanford Linear Accelerator Center

Outline

- Introduction
- Variance reduction
- Built in biasing options
- G4WrapperProcess
- Primary particle biasing
- Radioactive decay biasing
- Leading particle biasing
- Cross section biasing
- Bremsstrahlung splitting example
- Summary & future plans

Introduction

- Event biasing(variance reduction) techniques are important for many applications
- Geant4 is a toolkit
  - ➔ Users are free to implement their own biasing techniques
- Geant4 provides the following features to support event biasing
  - ➔ Some built in biasing techniques of general use with related examples
  - ➔ A utility class, G4WrapperProcess, to support user defined biasing

Variance Reduction

- Variance reduction techniques are used to reduce computing time taken to calculate a result with a given variance
  - ➔ Want to increase efficiency of the Monte Carlo
  - ➔ Measure of efficiency given by

$$\varepsilon = \frac{1}{s^2 T}$$

- ✓ s = variance on calculated quantity
- ✓ T = computing time

- When using a variance reduction technique, generally want to apply our own probability distribution, p’(x) in place of the natural one, p(x)
  - ➔ p’(x) enhances the production of whatever it is that were interested in
- Basically bypassing the full, slow, analogue simulation
- To get meaningful results, must apply a weight correction to correct for the fact that we’re not using the natural distribution:

$$w = \frac{p(x)}{p'(x)}$$

  - ➔ Preserves natural energy, angular distributions etc
- In general, all x values in the p(x) distribution should be possible in the p’(x) distribution

Built in Biasing Options

- Primary particle biasing ✔ Since v3.0
- Radioactive decay biasing ✔ Since v3.0
- Leading particle biasing - Hadronic
  - ➔ Partial MARS migration n, p, π, K (<5 GeV) ✔ Since v4.0
  - ➔ General lead particle biasing ✔ Since v4.3
- Cross section biasing - Hadronic ✔ Since v4.3
- Geometry based biasing (see talk by Alex Howard)
  - ➔ Importance sampling ✔ Since v5.0
  - ➔ Weight cutoff and weight window ✔ Since v5.2

G4WrapperProcess

- G4WrapperProcess can be used to implement user defined event biasing
  - ➔ Is a process itself, i.e inherits from G4VProcess
  - ➔ Wraps an existing process - by default, function calls are forwarded to existing process
  - ➔ Non-invasive way to modify behaviour of an existing process

- To use:
  - ➔ Subclass G4WrapperProcess and override appropriate methods, eg PostStepDoit
  - ➔ Register subclass with process manager in place of existing process
  - ➔ Register existing process with G4WrapperProcess

- G4WrapperProcess structure

```
class G4WrapperProcess  : public G4VProcess {

    G4VProcess* pRegProcess;
…
inline
void G4WrapperProcess::RegisterProcess(G4VProcess* process)
{
  pRegProcess=process;
…
}
…
inline G4VParticleChange*
G4WrapperProcess::PostStepDoIt(const G4Track& track,
                               const G4Step& stepData)
{
   return pRegProcess->PostStepDoIt(track, stepData);
}
```

- Example:

```
class MyWrapperProcess  : public G4WrapperProcess {
…
 G4VParticleChange* PostStepDoIt(const G4Track& track,
                                 const G4Step& step) {
     // Do something interesting
  }
}
```

```
void MyPhysicsList::ConstructProcess() {
   …
   G4LowEnergyBremsstrahlung* bremProcess =
      new G4LowEnergyBremsstrahlung();

   MyWrapperProcess* wrapper = new MyWrapperProcess();
   wrapper->RegisterProcess(bremProcess);

   processManager->AddProcess(wrapper, -1, -1, 3);
}
```

Primary Particle Biasing

- Increase number of primary particles generated in a particular phase space region of interest
  - ➔ Weight of primary particle is appropriately modified

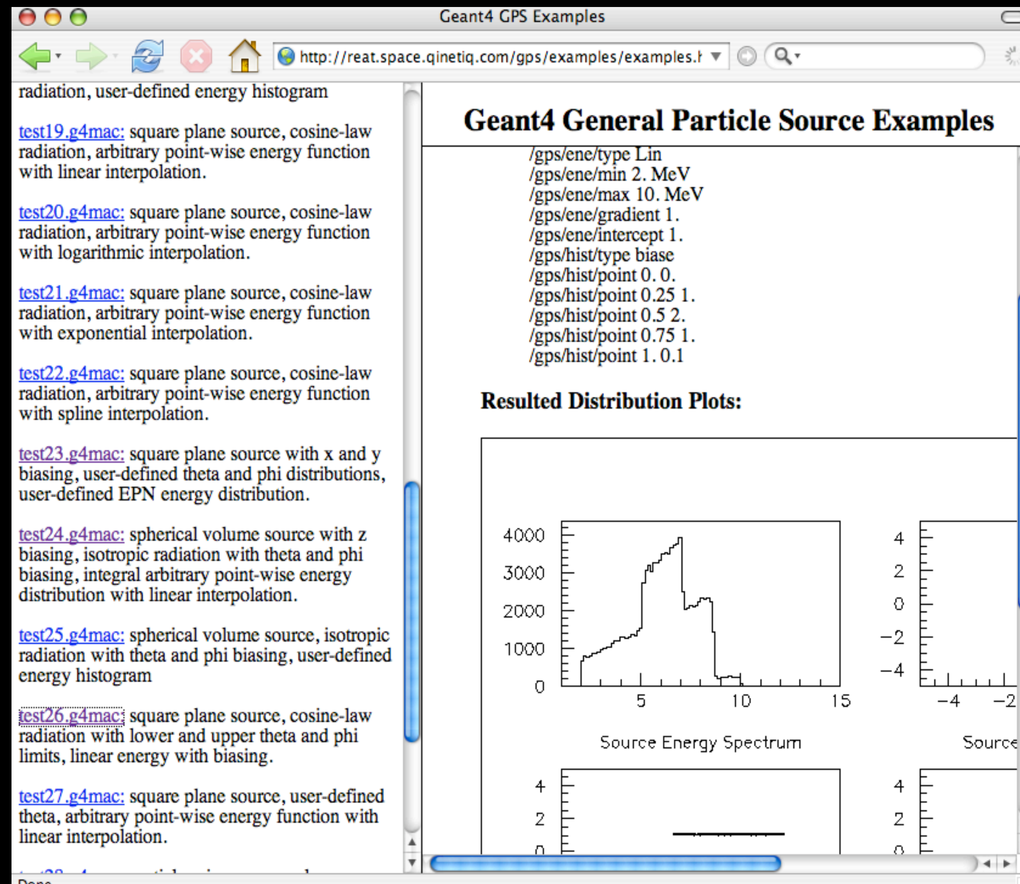
- Use case:
  - ➔ Increase number of high energy particles in cosmic ray spectrum

- General implementation provided by G4GeneralParticleSource class
  - ➔ Bias position, angular and energy distributions

- G4GeneralParticleSource is a concrete implementation of G4VPrimaryGenerator
  - ➔ Instantiate G4GeneralParticleSource in your G4VUserPrimaryGeneratorAction class
  - ➔ Configure biasing to be applied to sampling distributions through interactive commands

```
MyPrimaryGeneratorAction::MyPrimaryGeneratorAction() {
   generator = new G4GeneralParticleSource;
}
void
MyPrimaryGeneratorAction::GeneratePrimaries(G4Event*anEvent){
   generator->GeneratePrimaryVertex(anEvent);
}
```

- Extensive documentation at
  - → http://reat.space.qinetiq.com/gps/



- Online manual
- Detailed examples online
- Examples also distributed with Geant4
  - → examples/extended/eventgenerator/exgps

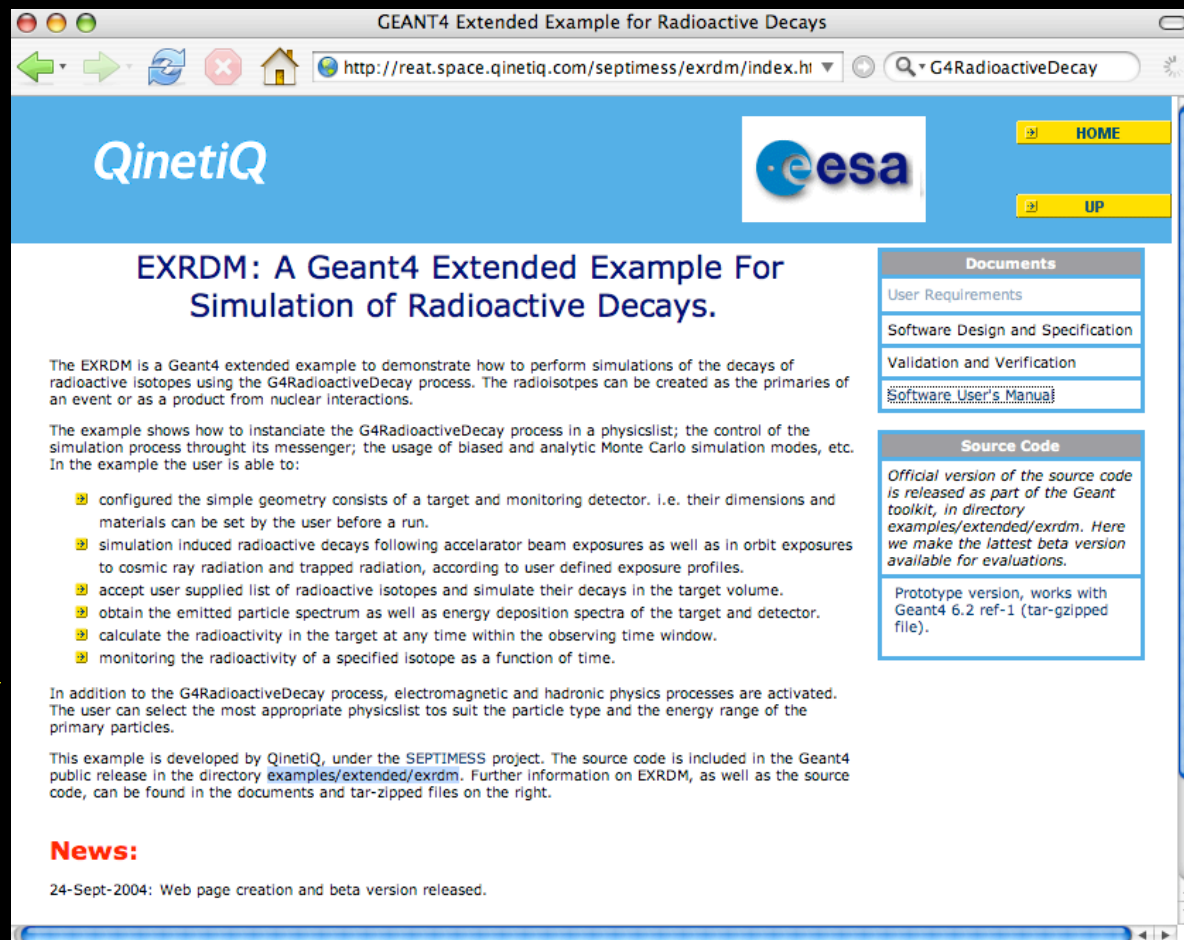
Radioactive Decay Biasing

- G4RadioactiveDecay simulates decay of radioactive nuclei
- Implements the following biasing methods
  - ➔ Increase sampling rate of radionuclides within observation times
    - ✓ User defined probability distribution function
  - ➔ Nuclear splitting
    - ✓ Parent nuclide is split into user defined number of nuclides
  - ➔ Branching ratio biasing
    - ✓ For a particular decay mode, sample branching ratios with equal probability

- G4RadioactiveDecay is a process
  - ➔ Register with process manager
  - ➔ Biasing can be controlled in compiled code or through interactive commands

```
void MyPhysicsList::ConstructProcess()
{
   …
   G4RadioactiveDecay* theRadioactiveDecay =
      new   G4RadioactiveDecay();

   G4ProcessManager* pmanager = …
   pmanager ->AddProcess(theRadioactiveDecay);
…
}
```

- Extensive documentation at
  - ➔ http://reat.space.qinetiq.com/septimess/exrdm/
  - ➔ http://www.space.qinetiq.com/geant4/rdm.html



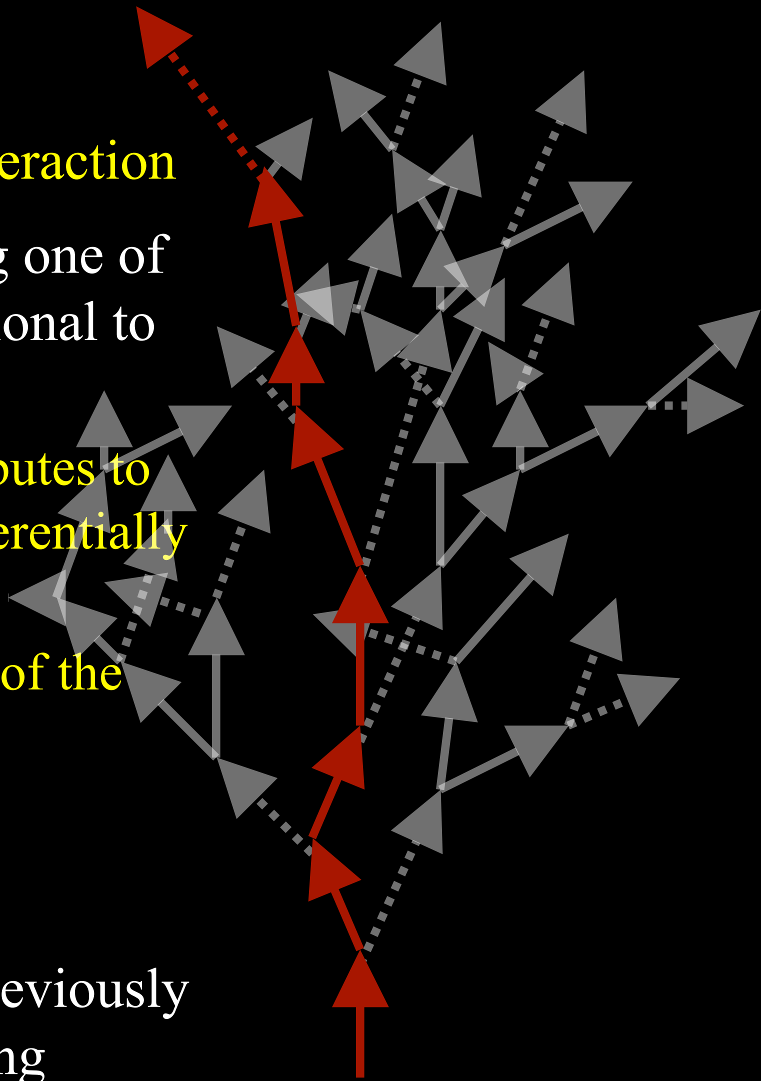
- Example at

examples/extended/
radioactivedecay/exrdm

Leading Particle Biasing - EM

- In analogue approach to electromagnetic shower simulation, each shower followed to completion
- Applications where high energy particles initiate electromagnetic showers may spend a significant amount of time in shower simulation
  - ➔ Computing time increases linearly with energy
- Leading particle biasing may significantly reduce computing time for suitable applications. Useful for:
  - ➔ Estimating shower punch through
  - ➔ Reducing time taken to simulate showers resulting from π^0s in hadronic cascades for example

- Most important processes contributing to EM shower development at high energies are bremsstrahlung and pair production
  - ➔ Two secondaries produced in each interaction
- Leading particle biasing involves selecting one of the secondaries with a probability proportional to secondary energy
  - ➔ Highest energy secondary which contributes to most to the total energy deposition preferentially selected
  - ➔ Lower energy secondary selected some of the time
  - ➔ Remaining secondary killed
  - ➔ Weight surviving secondary
- Use G4WrapperProcess class described previously useful for to implement user defined leading particle biasing



Leading Particle biasing - Hadronic

- Useful for punch through studies
- G4Mars5Gev
  - ➔ Inclusive event generator for hadron(photon) interactions with nuclei
  - ➔ Translated from Mars13(98) version of MARS code system
    - ➔ MARS is a particle simulation Monte Carlo
    - ➔ More details on MARS at http://www-ap.fnal.gov/MARS
  - ➔ Generates fixed number of particles at each vertex with appropriate weights assigned
  - ➔ Valid with energies E< 5 GeV with the following particle types
    - ➔ π+, π-, K+, K-, K0L, K0S, proton, neutron, anti-proton, gamma

- To use, create a G4Mars5GeV object and register with an appropriate inelastic process:

```
void MyPhysicsList::ConstructProcess() {
   …
   G4Mars5Gev* leadModel = new G4Mars5GeV();

   G4ProtonInelasticProcess* inelProcess =
      new G4ProtonInelasticProcess();
   inelProcess->RegisterMe(leadModel);

   processManager->AdddiscreteProcess(inelProcess);
}
```

- More examples provided in the LHEP_LEAD, LHEP_LEAD_HP, QGSC_LEAD, QGSC_LEAD_HP physics lists
- Documentation:
  - ➔ http://geant4.web.cern.ch/geant4/support/proc_mod_catalog/models/hadronic/LeadParticleBias.html

- G4HadLeadBias
  - Built in utility for hadronic processes
    - ✓ disabled by default
  - Keep only the most important part of the event and representative tracks of given particle type
    - ✓ Keep track with highest energy, I.e, the leading particle
    - ✓ Of the remaining tracks, select one from each of the following types if they exist: Baryons, π^0's, mesons, leptons
    - ✓ Apply appropriate weight
  - Set SwitchLeadBiasOn environmental variable to activate

Cross Section Biasing

- Artificially enhance/reduce cross section of a process
- Useful for studying
  - ➔ Thin layer interactions
  - ➔ Thick layer shielding
- Built in cross section biasing in hadronics for PhotoInelastic, ElectronNuclear and PositronNuclear processes
- User can implement cross section biasing for other processes through G4WrapperProcess
  - ➔ Documentation at http://www.triumf.ca/geant4-03/talks/03-Wednesday-AM-1/05-F.Lei/

- Built in hadronic cross section biasing controlled through BiasCrossSectionByFactor method in G4HadronicProcess

```
void MyPhysicsList::ConstructProcess() {
   …
   G4ElectroNuclearReaction * theElectroReaction =
      new G4ElectroNuclearReaction;

   G4ElectronNuclearProcess theElectronNuclearProcess;
   theElectronNuclearProcess.RegisterMe(theElectroReaction);

   theElectronNuclearProcess.BiasCrossSectionByFactor(100);
   pManager->AddDiscreteProcess(&theElectronNuclearProcess);
   …
}
```

- More details at
   - ➔ http://www.triumf.ca/geant4-03/talks/03-Wednesday-AM-1/03-J.Wellisch/biasing.hadronics.pdf

Uniform Bremsstrahlung Splitting

- Example of biasing through enhancing production of secondaries
- Aim to increase Monte Carlo efficiency by reducing computing time spent tracking electrons
  - ➔ In this case only interested in scoring photons
- Enhance photon production by applying splitting when a bremsstrahlung interaction occurs
  - ➔ Instead of sampling photon energy & angular distributions just once, sample them N times
  - ➔ Creates N unique secondaries
  - ➔ Different splitting method compared to importance sampling where N identical copies are created

- Electron energy is reduced by energy of just one photon
  - ➔ Energy is not conserved per event, although is conserved on average
- As usual, remove bias introduced by generating multiple secondaries by assigning a statistical weight to each secondary

$$weight = \frac{Parent\,weight}{N}$$

  - ➔ N = number of secondary photons
  - ➔ Preserves correct photon energy and angular distributions
- No default bremsstrahlung splitting in Geant4 toolkit
- User can implement bremsstrahlung splitting through G4WrapperProcess

Example Implementation

- Create BremSplittingProcess class
  - Inherit from G4WrapperProcess
  - Override PostStepDoIt method of G4WrapperProcess
  - Introduce splitting configuration parameters

```
class BremSplittingProcess : public G4WrapperProcess {
  // Override PostStepDoIt  method
  G4VParticleChange*
  PostStepDoIt(const G4Track& track, const G4Step& step);

  static void SetNSplit(G4int);
  static void SetIsActive(G4bool);
…
  // Data members
  static G4int fNSplit;
  static G4bool fActive;
};
```

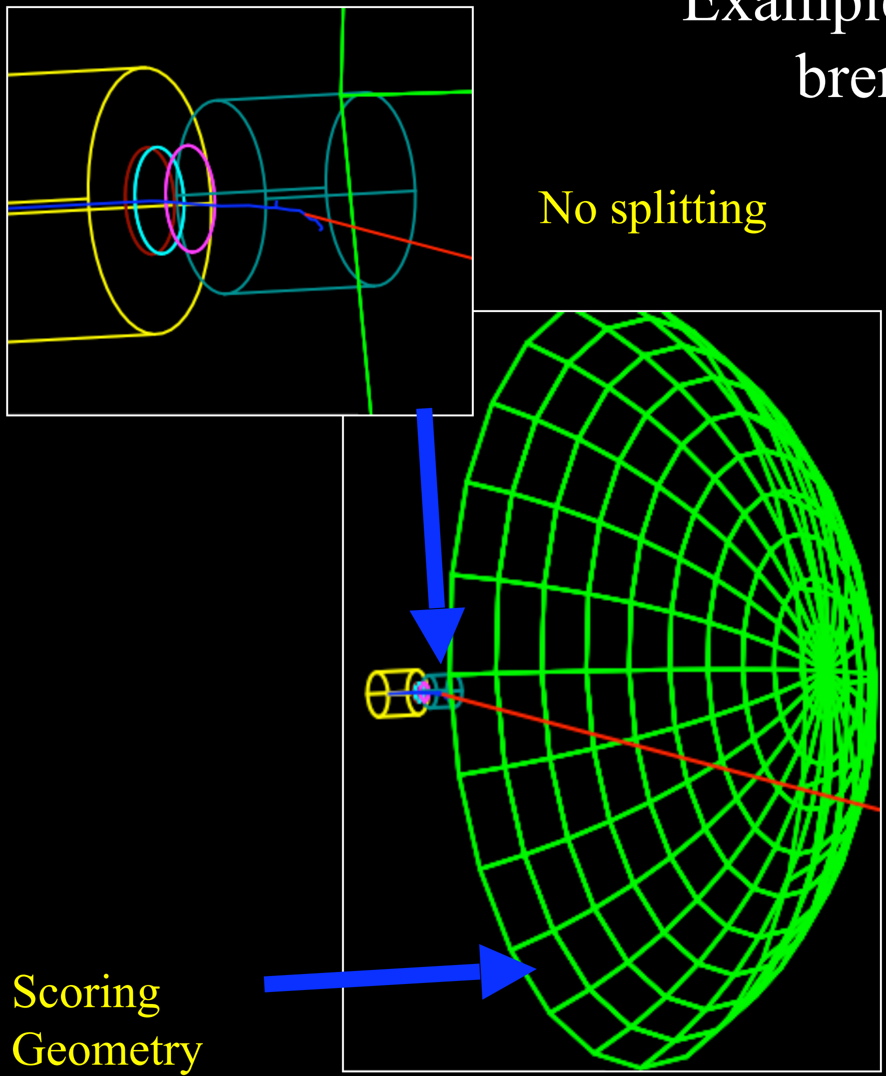
```
G4VParticleChange*
BremSplittingProcess::PostStepDoIt(const G4Track& track, const G4Step& step)
{
…
 G4double weight = track.GetWeight()/fNSplit;
 std::vector<G4Track*> secondaries; // Secondary store

  // Loop over PostStepDoIt method to generate multiple secondaries.
  for (i=0; i<fNSplit; i++) {
    particleChange = pRegProcess->PostStepDoIt(track, step);
    assert (0 != particleChange);
    G4int j(0);

    for (j=0; j<particleChange->GetNumberOfSecondaries(); j++) {
      secondaries.push_back(new G4Track(*(particleChange->GetSecondary(j))));
    }
  }
  particleChange->SetNumberOfSecondaries(secondaries.size());
  particleChange->SetSecondaryWeightByProcess(true);

  std::vector<G4Track*>::iterator iter = secondaries.begin(); // Add all secondaries

  while (iter != secondaries.end()) {
    G4Track* myTrack = *iter;
    myTrack->SetWeight(weight);
    particleChange->AddSecondary(myTrack);
    iter++;
  }
…
  return particleChange;
}
```

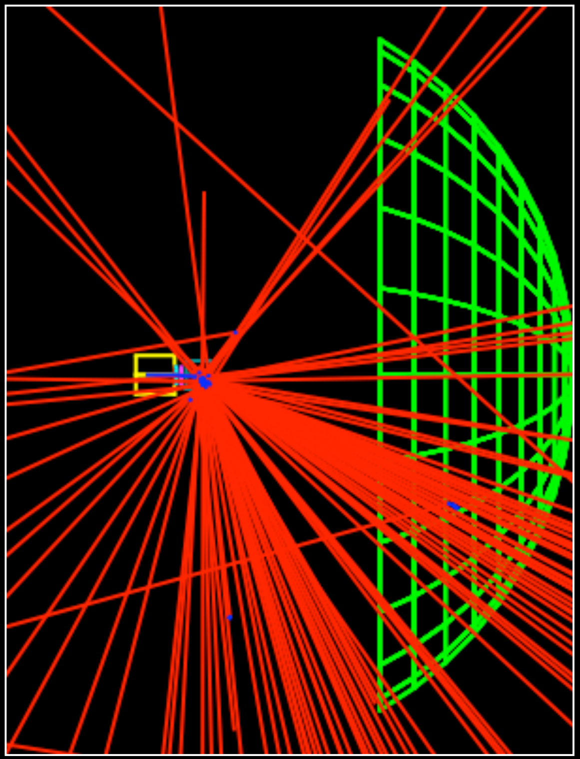
- Finally, register BremSplittingProcess with electron process manager

```
void MyPhysicsList::ConstructProcess() {
…
    G4LowEnergyBremsstrahlung* bremProcess =
       new G4LowEnergyBremsstrahlung();

    BremSplittingProcess* bremSplitting =
       new BremSplittingProcess();

    bremSplitting->RegisterProcess(bremProcess);

    pmanager->AddProcess(bremSplitting,-1,-1, 3);
…
}
```

- Use same procedure to implement Russian Roulette + bremsstrahlung splitting

Example demonstrating uniform bremsstrahlung splitting





Summary & Future Plans

- Presented a number of physics based event biasing techniques
  - ➔ Some biasing options are implemented in Geant4 for general use
  - ➔ Others need to be implemented by user

- Develop examples to demonstrate use
- See Alex Howard's talk for information on geometrical based biasing