September 2007

Physics Biasing Framework Prototype

Jane Tinslay, SLAC

Stanford Linear Accelerator Center

Geant 4

User Requirements

- Biasing techniques
  - Would like to be able to implement everything/most of the techniques available in other Monte Carlo codes, including specialisations implemented in each code
    - EGS family, Fluka, MCNP family, Penelope
  - User defined biasing & experimenting

- Specific techniques:
  - Leading particle biasing
  - Implicit capture
  - Bremsstrahlung splitting
  - Cross sectional biasing

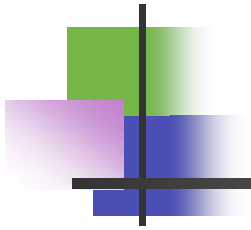
- User Biasing Code
  - Code to a minimal interface
    - Not G4VProcess
      - Most of interface is irrelevant, distracting
      - Cut down on unnecessary function calls
  - Don’t need to modify existing G4VProcess physics process code
  - Don’t need to modify G4VUserPhysicsList
    - Allows use of pre-packaged physics lists
  - Put biasing code in dedicated user hook
    - Also serves as a useful starting point
  - Use a simple tool to do process list manipulation

Physics Biasing Framework

- Aim to provide flexibility through common physics biasing framework
  - Make life a little easier for the users
- Two levels to biasing
  - Processing (G4SteppingManager) level :
    - Manippulating physics & processing lists, taking into account when a physics list/process is active (triggered)
      - Independent of process type
        - Although whatever process grouping constraints currently imposed must still be applied
  - Process level:
    - Actual biasing code
    - Executed in GPIL/DoIt methods
    - Avoid “do it all” interface classes to simplify biasing while allowing access to underlying processes when need to do more complex biasing
      - More complex biasing working on process type level is limited as to what interfaces the actual process provides

Prototype

- Prototype code in CVS under geant4/source/processes/biasing/test/physics_biasing/
  - GPR stands for generalised processing since biasing processes don't need to inherit from anything
- Relevant directories
  - gpr_base : basic general use building blocks
  - gpr_core : more complex structures forming gpr processing
  - gpr_configuration : user interface stuff
  - gpr_geant4_modifications : modifications to geant4
  - gpr_examples : A01 example demonstrating biasing
- It's development code.
  - Lots of debugging print
  - Subset of desired features implemented
  - Probably buggy
  - Just an example of how things could be done

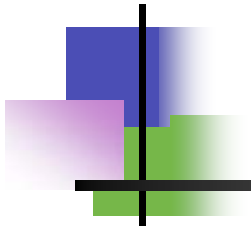


Relevant Technical Stuff (Brief)

- Patching into Geant4
- Implementing biasing code
- Triggers
- User interface

Patching into Geant4

- Modify G4RunManager to use new user hook, G4VUserBiasing
- Processing manager for biasing called G4GPRManager (at the moment).
  - GPR stands for generalised processing
- GPRManager handles:
  - Multiple physics lists
  - Varying process lists
  - Multiple biasing tools
- If a particle is being biased, processing is routed through G4GPRSteppingManager, where G4GPRManager is picked up.
- If a particle is not being biased, regular processing with G4SteppingManager is used
- Switching between stepping managers done in G4TrackingManager
- For efficiency reasons, pointer to G4GPRManager added to G4ParticleDefinition



Implementing Biasing Code

- Biasing code may be a function or an object
- Actual biasing code same as implemented with G4WrapperProcess with these exceptions:
  - Access to the process to be biased is through method interface rather than a data member
  - Incoming process is wrapped
    - Incoming process need not be a G4VProcess
    - Full functionality should allow access to underlying G4VProcess or whatever to allow more complex biasing

```
G4VParticleChange*
BremSplittingProcess::PostStepDoIt(const G4Track& track, const G4Step& step)
{
  unsigned nSplit =  100;

  G4VParticleChange* particleChange = pRegProcess->PostStepDoIt(track, step);
  assert (0 != particleChange);

  unsigned i(0);
  G4double weight = track.GetWeight()/nSplit;
...
```

```
namespace A01BremSplittingFunctions {

  G4VParticleChange* BremSplitting(G4GPRProcessWrappers::G4GPRDiscreteDoIt& original,
                                   const G4Track& track, const G4Step& step)
  {
    unsigned nSplit = 100;

    G4VParticleChange* particleChange = original(track, step);
    assert (0 != particleChange);

    unsigned i(0);
    G4double weight = track.GetWeight()/nSplit;
...
```

Triggers

- Very simple functions/objects returning a boolean decision
- Are used to decide
  - In what situation physics list X is to be used
  - Or, in what situation is process Y is to be used
  - Or, In what situation is biasing Z is to be used
- Multiple types of triggers depending on where in the code they are evaluated:
  - G4TriggerTypes::Tracking::StartTracking : Evaluated at start tracking
    - Eg, want to apply biasing algorithm only on primary particle processes
  - G4TriggerTypes::Stepping::StartStep : Evaluated at start of step
    - Eg, want to use process Y when track is below a certain energy
  - G4TriggerTypes::Geometry::NewVolume
    - Eg, Use physics list Z in a particular volume. Can be extended to regions
  - …

A01 Demonstration Triggers

```
namespace A01Triggers {

  // Return true if track is the daughter of a primary
  G4bool DaughterOfPrimaryTrigger(G4Track* track)
  {
    return (track->GetParentID() == 1);
  }

  // Return true if in calorimeter volume
  G4bool CalorimeterTrigger(const G4Track& track, const G4Step& step)
  {
    return track.GetVolume()->GetName() == "cellPhysical";
  }

  // Return true if track energy is less than 5 GeV
  G4bool Hadronic_LeadingParticleBiasing_Trigger(const G4Track& track, const G4Step& step)
  {
    return (track.GetKineticEnergy() < 5*GeV);
  }
}
```

User Interface (1)

- G4VUserBiasing implemented in biasing equivalent of G4VUserPhysicsList
- G4VUserBiasing registered with run manager

```
class A01Biasing : public G4VUserPhysicsBiasing {

public:

  void ConstructBiasing();
  {
   // Configure biasing
   ...
```

```
int main(int argc,char** argv)
{
  // RunManager construction
  G4RunManager* runManager = new G4RunManager;
...

  runManager->SetUserInitialization(new A01DetectorConstruction);
  runManager->SetUserInitialization(new A01PhysicsList);
  runManager->SetUserInitialization(new A01Biasing_New_Calorimeter_PhysicsLists);
  //runManager->SetUserInitialization(new A01Biasing_Leading_Particle_Biasing);
  //runManager->SetUserInitialization(new A01Biasing_BremSplitting_With_Russian_Roulette);
```

User Interface (2)

- processes/biasing/test/physics_biasing/gpr_configuration/include/G4GPRBuilder.hh
- Implements utility functions used to configure biasing in G4VUserBiasing::ConstructBiasing Eg:
  - CreateDefaultPhysicsList
  - CreatePhysicsListWithTrigger
  - AddProcess
  - AddTriggeredBiasing
  - AddBiasing
  - …
- User interface can be whatever you want or user is comfortable with
  - Easy to add new functions

A01 Example Biasing Demonstration

- Electromagnetic and hadronic leading particle biasing
- Bremsstrahlung splitting with Russian Roulette
- New calorimeter physics list

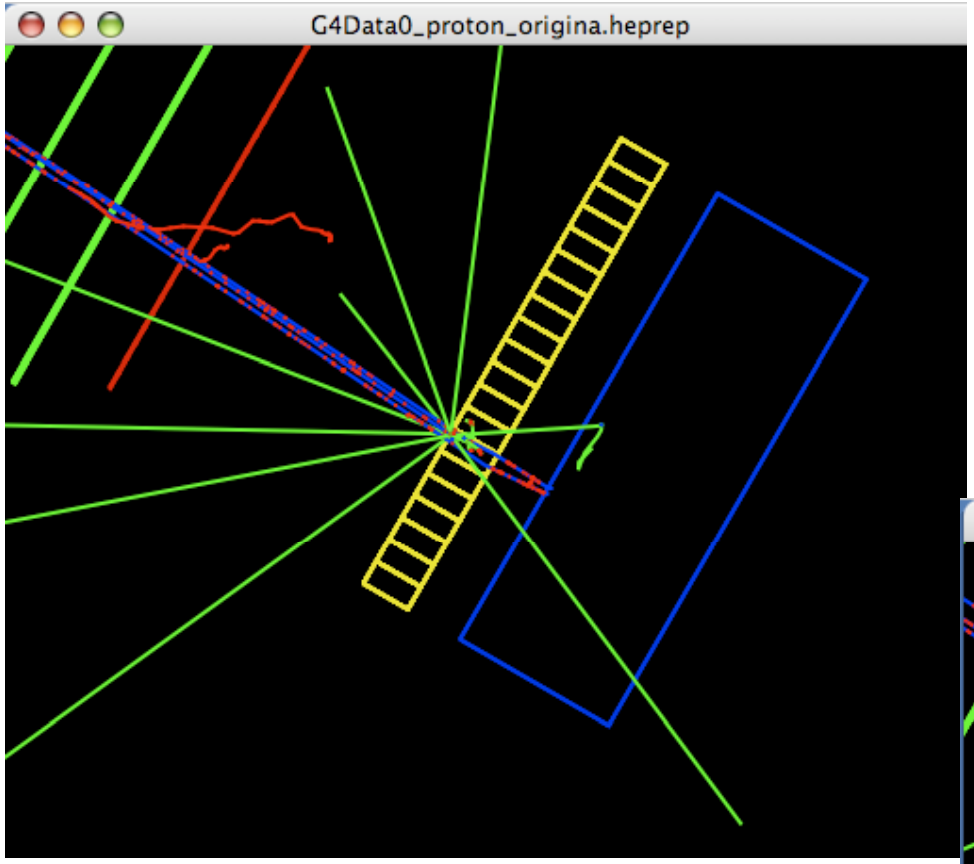
Leading Particle Biasing

- processes/biasing/test/physics_biasing/gpr_examples/A01/include/A01Biasing_Leading_Particle_Biasing.hh
- Implements simple (EGS style) electromagnetic leading particle biasing for e-, e+ and gammas in the electromagnetic calorimeter
- Implements equivalent of existing hadronic leading particle biasing
  - Bias only G4HadronicProcesses
  - Only apply to incoming tracks with energy < 5 GeV

Leading Particle Biasing Configuration

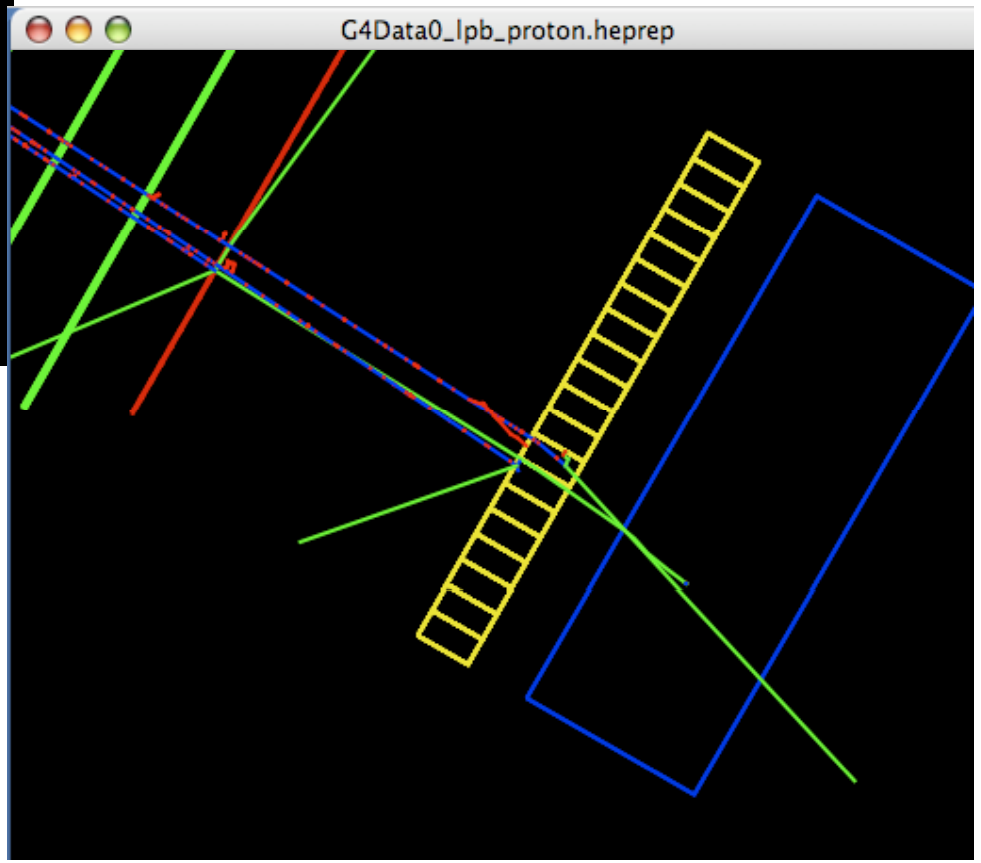
```
class A01Biasing_Leading_Particle_Biasing : public G4VUserPhysicsBiasing {

public:

  void ConstructBiasing()
  {
    // AddTriggeredBiasing<Particle, Process list, Trigger type>
    //   (Name, Process index, Biasing function, Trigger function)
    AddTriggeredBiasing<G4Electron, G4GPRProcessLists::DiscreteDoIt, G4GPRTriggerTypes::Geometry::NewVolume>
      ("LeadingParticlebiasing", 3, &A01LeadingParticleBiasing_EM::SimpleEM, &CalorimeterTrigger);


...
    // Slightly more complex configuration case - want to bias all G4HadronicProcess's
    G4GPRBiasingConfig hadronicConfig;
    hadronicConfig.SelectAllParticles();
    hadronicConfig.SelectVProcess<G4HadronicProcess>();

    // AddTriggeredBiasing<Process list, Trigger type>
    //   (Name, Biasing function, Trigger function, biasing placement configuration)
    AddTriggeredBiasing<G4GPRProcessLists::DiscreteDoIt,
                        G4GPRTriggerTypes::Stepping::StartStep>
      ("LeadingParticlebiasing_Hadronic",
       &A01LeadingParticleBiasing_Hadronic::Biasing,
       &Hadronic_LeadingParticleBiasing_Trigger, hadronicConfig);
  }
```

Primary Proton

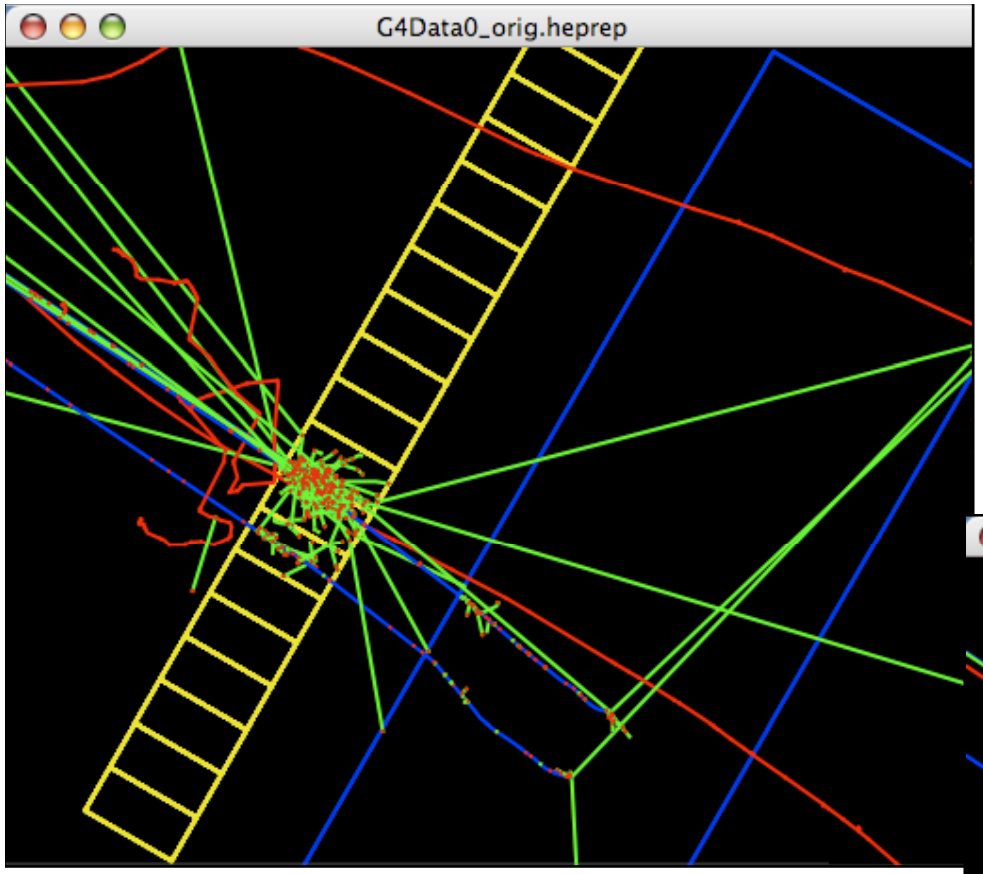


Regular processing

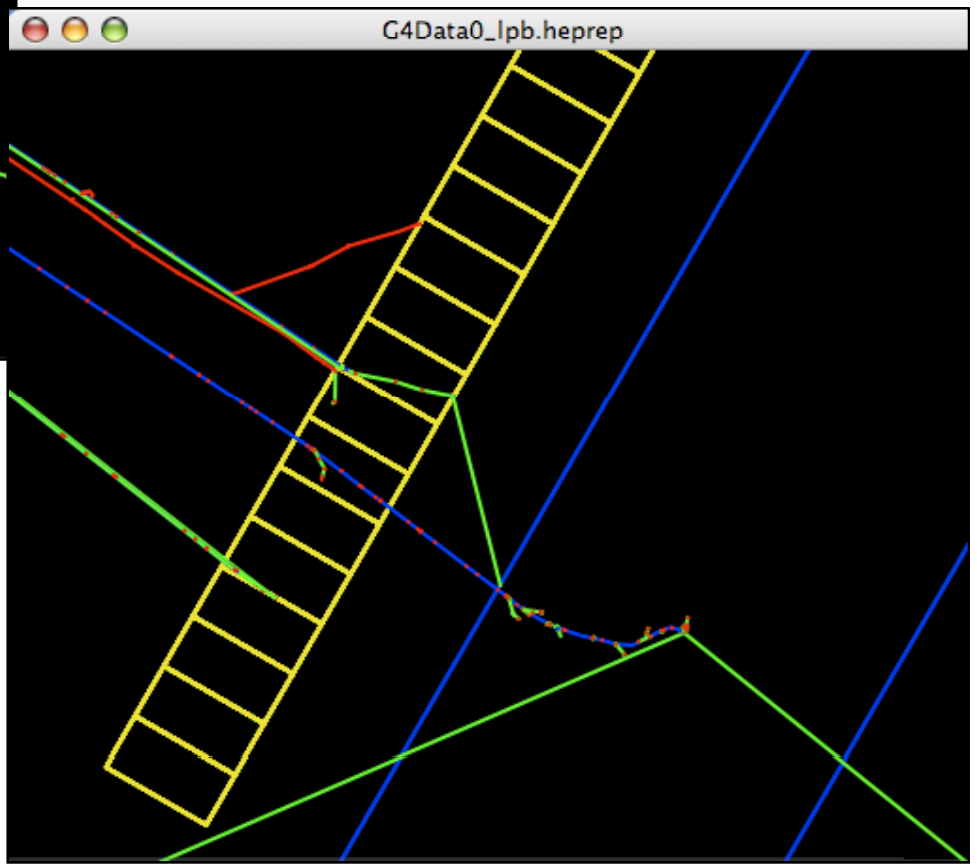
Leading particle biasing →



Random Primaries



Regular processing



Leading particle biasing →

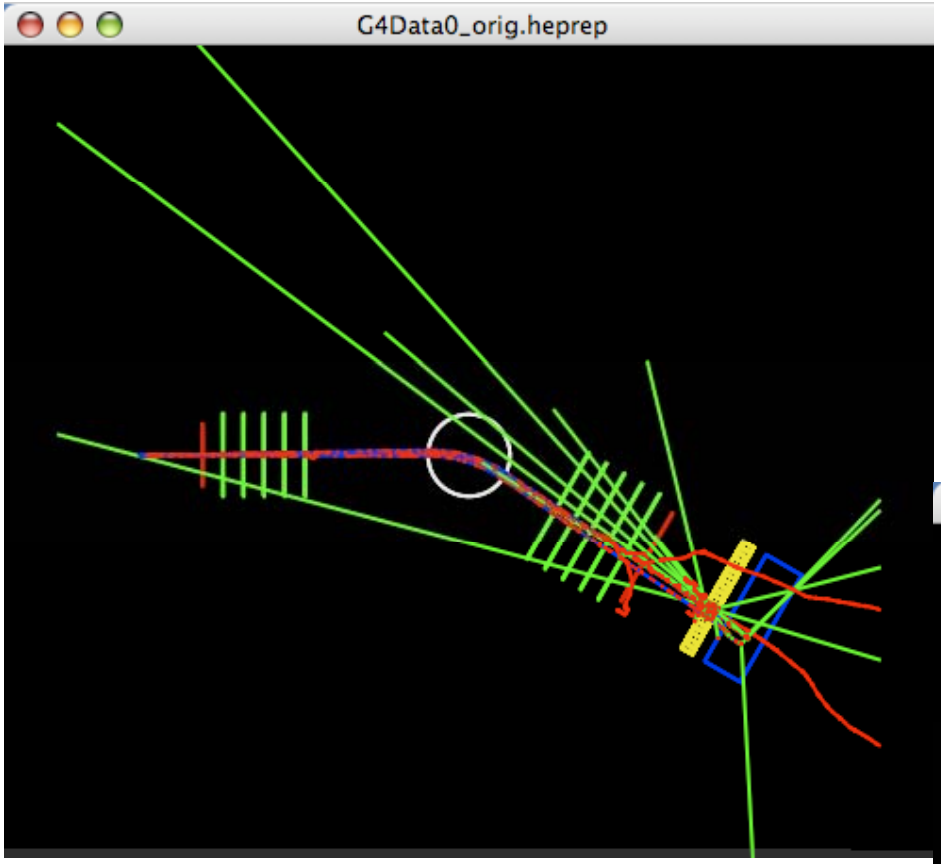
Brem Splitting With Russian Roulette

- processes/biasing/test/physics_biasing/gpr_examples/A01/include/A01Biasing_Leading_Particle_Biasing.hh
- Create 100 unique photons per brem split event
- Play Russian Roulette on charged secondaries for population control
- Apply biasing everywhere

Brem Splitting Configuration

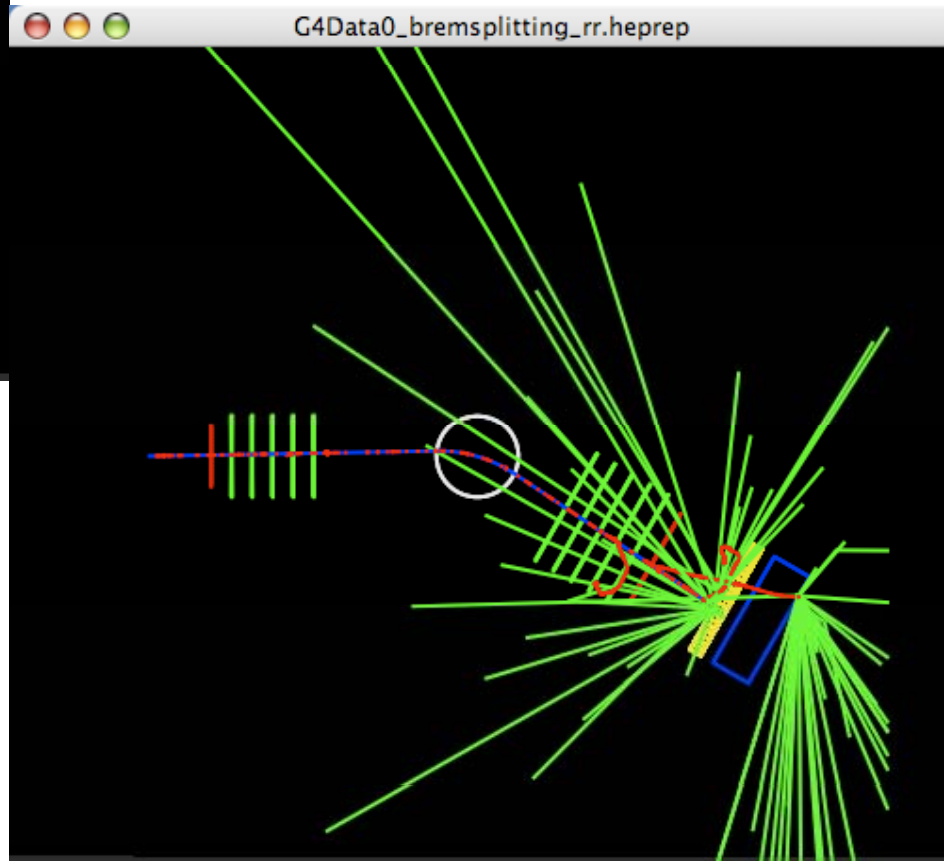
```
class A01Biasing_BremSplitting_With_Russian_Roulette : public G4VUserPhysicsBiasing {

public:

  void ConstructBiasing()
  {
    // AddBiasing<Process list>(Name, Biasing function, Process placement)
    G4GPRBiasingConfig electronCfg;
    electronCfg.SelectVProcess<G4eBremsstrahlung>();
    electronCfg.SelectParticle<G4Electron>();

    AddBiasing<G4GPRProcessLists::DiscreteDoIt>("Uniform Brem Splitting",
                                                &A01BremSplittingFunctions::BremSplitting,
                                                electronCfg);

    G4GPRBiasingConfig gammaCfg;
    gammaCfg.SelectVProcess<G4GammaConversion>();
    gammaCfg.SelectVProcess<G4ComptonScattering>();
    gammaCfg.SelectVProcess<G4PhotoElectricEffect>();
    gammaCfg.SelectParticle<G4Gamma>();

    AddBiasing<G4GPRProcessLists::DiscreteDoIt>("Roulette",
                                                &A01BremSplittingFunctions::Roulette,
                                                gammaCfg);

  }
};
```

Random Primaries



Regular processing

Brem splitting



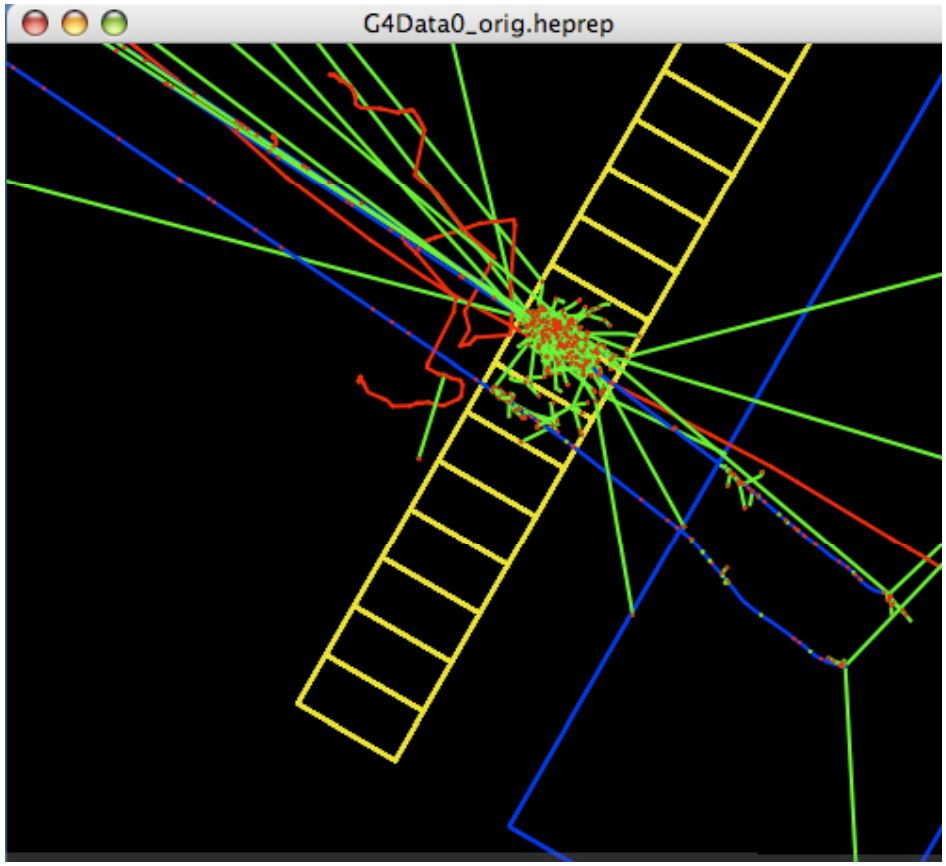
New calorimeter Physics list

- processes/biasing/test/physics_biasing/gpr_examples/A01/include/A01Biasing_Leading_Particle_Biasing.hh
- Create new physics lists for photon, e+, e- triggered only in calorimeter
- No photon, e-, e+ interactions in calorimeter

New Physics List Configuration

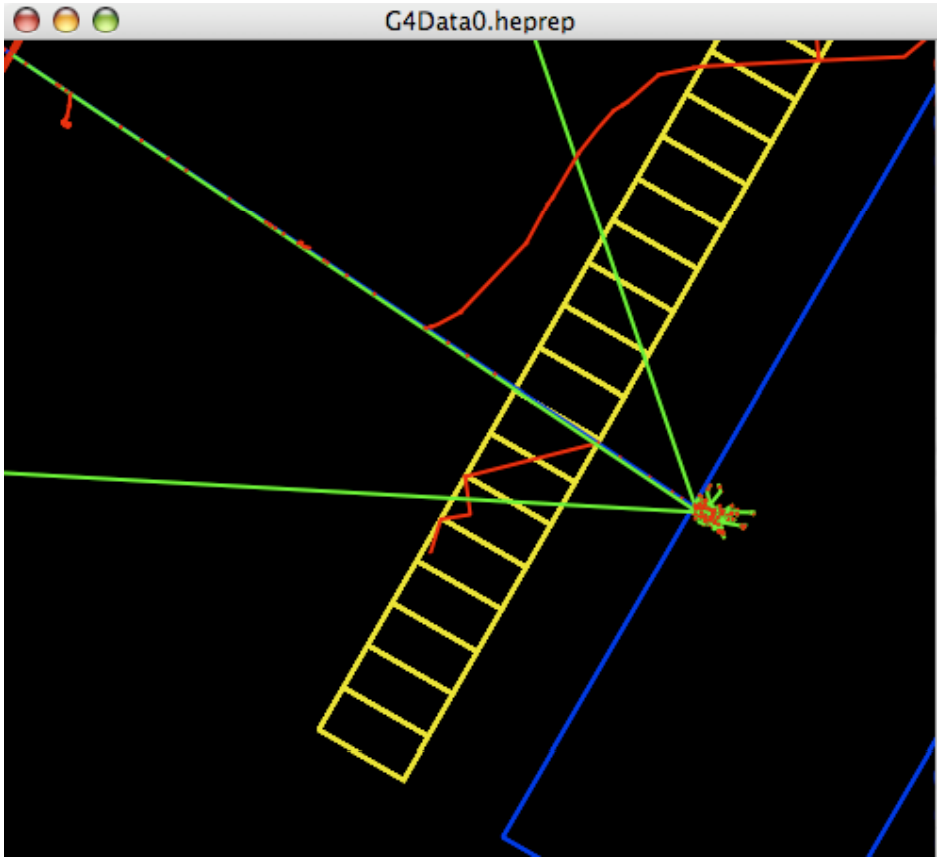
```
class A01Biasing_New_Calorimeter_PhysicsLists : public G4VUserPhysicsBiasing {

public:

  void ConstructBiasing()
  {
    // CreatePhysicsList<Particle>(Trigger type)(List name, TriggeR)
    // AddProcess<Particle>(process, atRestIdx, alongStepIdx, postStepIdx)

    G4String caloListName("Calorimeter_PhysicsList");

    CreatePhysicsListWithTrigger<G4Gamma, G4GPRTriggerTypes::Geometry::NewVolume>
      (caloListName, &CalorimeterTrigger);
    AddProcess<G4Gamma>(new G4Transportation,  -1, 0, 0, caloListName);

    CreatePhysicsListWithTrigger<G4Electron, G4GPRTriggerTypes::Geometry::NewVolume>
      (caloListName, &CalorimeterTrigger);
    AddProcess<G4Electron>(new G4Transportation,  -1, 0, 0, caloListName);

    CreatePhysicsListWithTrigger<G4Positron, G4GPRTriggerTypes::Geometry::NewVolume>
      (caloListName, &CalorimeterTrigger);
    AddProcess<G4Positron>(new G4Transportation,  -1, 0, 0, caloListName);
  }
};
```

Random Primaries



Regular processing

No e+, e-, gamma
Interactions in calorimeter



Summary

- Prototype developed and in CVS
  - Subset of final functionality implemented
- Some simple biasing techniques have been implemented an minially tested
- Try to implement more complex techniques - eg, forced interaction, cross section biasing & see where the holes are