



Hadr08

Hadronic Extended Example

Alberto Ribon
CERN EP/SFT

Hadr08 : hadronic-model per-region

- Geant4 does not allow physics lists per region.
However, it is possible to specify EM models per region.
But the hadronic framework does not allow hadronic models per region
- This example shows how it is possible to achieve in practice “hadronic-models per region” by using “generic biasing”
 - We use the powerful "generic biasing" machinery available in Geant4, but the actual weights of all tracks remain to the usual value (1.0) as in the normal (unbiased) case
 - In our example (inspired by a real application in ALICE) we want to use the physics lists `FTFP_BERT` everywhere, except in the tracker region, where we want to use the a more accurate but slower intranuclear cascade model, `INCLXX`, instead of `BERT`
 - In the case of ALICE, replacing `FTFP_BERT` with `FTFP_INCLXX` would slow down the performance of the simulation by a factor of 2, whereas with this trick they can keep practically the same computing performance

A few details

- Very simple geometry
 - A box of Silicon (Tracker) + a box of CsI (EM calo) + a box of Iron (HAD calo)
- Pi^+ primary particle along the z-axis by default
 - hadr08.in : 100 events, 5 GeV pi^+
- No output
- Can run both in sequential or multi-threaded mode

In the following few slides, some code highlight on how “generic biasing” is used in this example

- For brevity, shown only the case of a proton, but this example treats also pion^+ , pion^- and neutron

```
int main( ... ) {  
    ...  
    FTFP_BERT* physicsList = new FTFP_BERT;  
    G4GenericBiasingPhysics* biasingPhysics = new G4GenericBiasingPhysics;  
    biasingPhysics->Bias( "proton" );  
    physicsList->RegisterPhysics( biasingPhysics );  
    ...  
}
```

```
void GB02DetectorConstruction::ConstructSDandField() {  
    ...  
    BiasingOperator* biasingOperator = new BiasingOperator;  
    biasingOperator->AddParticle( "proton" );  
    biasingOperator->AttachTo( trackerLogicalVolume );  
}
```

```

class BiasingOperator : public G4VBiasingOperator {
public:
    BiasingOperator();
    virtual ~BiasingOperator() {}
    void AddParticle( G4String particleName );
    virtual G4VBiasingOperation* ProposeFinalStateBiasingOperation( const G4Track* track,
                                                                    const G4BiasingProcessInterface* callingProcess ) final;

    // Not used:
    virtual G4VBiasingOperation* ProposeNonPhysicsBiasingOperation(...) { return 0; }
    virtual G4VBiasingOperation* ProposeOccurenceBiasingOperation(...) { return 0; }

private:
    std::vector< const G4ParticleDefinition* > fParticlesToBias;
    BiasingOperation* fBiasingOperation;
};

```

```
BiassingOperator::BiassingOperator() : G4VBiasingOperator( "BiassingOperator" ) {  
    fBiassingOperation = new BiassingOperation( "BiassingOperation" );  
}
```

```
void BiassingOperator::AddParticle( G4String particleName ) {  
    const G4ParticleDefinition* particle = G4ParticleTable::GetParticleTable()->FindParticle( particleName );  
    ... // check that particle is not nullptr  
    fParticlesToBias.push_back( particle );  
}
```

```
G4VBiasingOperation* BiassingOperator::ProposeFinalStateBiassingOperation( const G4Track* ,  
                                                                           const G4BiassingProcessInterface* callingProcess ) {  
    // Apply the biasing operation only for proton inelastic process  
    if ( callingProcess && callingProcess->GetWrappedProcess() &&  
        callingProcess->GetWrappedProcess()->GetProcessName() == "protonInelastic" ) {  
        return fBiassingOperation;  
    } else {  
        return 0;  
    }  
}
```

```

class BiasingOperation : public G4VBiasingOperation {
public:
    ...
    virtual G4VParticleChange* ApplyFinalStateBiasing(...);

    // Unused :
    virtual const G4VBiasingInteractionLaw* ProvideOccurenceBiasingInteractionLaw( ...) { return 0; }
    virtual G4double DistanceToApplyOperation( ...) { return DBL_MAX; }
    virtual G4VParticleChange* GenerateBiasingFinalState(...) { return 0; }

private:
    G4ProtonInelasticProcess* fProtonInelasticProcess;
};

BiasingOperation::BiasingOperation( G4String name ) : G4VBiasingOperation( name ) {
    fProtonInelasticProcess = new G4ProtonInelasticProcess;
    // Create hadronic models and cross sections for the alternative "FTFP_INCLXX"
    // to be used only in the TrackerLogicalVolume , and register them in fProtonInelasticProcess
    ...
}

G4VParticleChange* BiasingOperation::ApplyFinalStateBiasing( const G4BiasingProcessInterface* ,
                                                             const G4Track* track, const G4Step* step, G4bool& ) {
    if ( track->GetParticleDefinition() == G4Proton::Definition() ) {
        return fProtonInelasticProcess->PostStepDolt( *track, *step );
    }
}

```