REVISIT / RETREAT PRODUCTION THRESHOLDS AND PHYSICS PROCESS FRAMEWORK

Parallel 3B

Marc Verderi
LLR/Ecole polytechnique
Lund Collaboration Meeting
August 2018

Layout

- Introduction
- Cuts & Kernel classes
- Stating on Present Scheme
- Considering an Other Scheme

Introduction

Production Thresholds: initial scheme

- Production thresholds ("cuts") initially considered by Geant4 as an issue fundamental enough to be addressed at the kernel classes level.

- Initial scheme:
  - The definition of cuts was mandatory
    - and had to be made in the physics list's pure virtual method:

      `void G4VUserPhysicsList::SetCuts()`
    - that, up to 2011
      - and then a default implementation came for `SetCuts()`.

  - An explicit declaration of particles subject to cuts was made in the base class

    `G4ParticleDefinition`
    - With the predefined and fixed set $\{e^-, e^+, \gamma$ and $p\}$

  - A control at tracking time of the conformance of the produced secondary particles wrt to their respective thresholds
    - Done by the `G4SteppingManager` after each process `DoIt` invocation
      - But allowing exceptions to this conformance, with the "`GoodForTracking`" flag

Particles with Production Thresholds

Particle produced	Production process	Motivation
e^-	Ionization	Heavy production (limited by energy binding to atoms). These are actually "recoil electrons". Threshold needed to limit the production.
e^+	Conversion	No divergence nor heavy production. Use case : production cut in mountain rock for, e.g., dark matter experiments.
γ	Bremsstrahlung	Cross-section divergence (actually limited by dielectric effects at very low energies). Threshold needed to limit the production.
p (ions)	Hadron elastic	Threshold on recoil proton, e.g. n scattering on proton, ejecting it. Mechanism adapted for ions. Threshold defines the "visibility" cut.

- In addition there is a cut for ions, defined internally in `G4HadronElasticProcess` as:
  - (100*keV)*proton_cut_in_mm
    - (Note: threshold advertised as such, but no explicit use of units in the code, so is this robust ?)
  - As for protons, this is not a threshold on actual production, but on recoil.

- We see that "cuts" have different functions:
  - **Practical view:** thresholds on heavy/diverging productions (e^-, γ), and visibility thresholds (e^+, p, ions)
  - **Physics-based view:** production thresholds (e^+, γ), and recoil thresholds (e^-, p, ions)
  - ☞ Note that the practical view is the one which is the most relevant to us.

Questions motivating this presentation

1. Where are we compared to the initial scheme ?

2. Isn't this scheme "overkilling" ?
   - Because only a few processes need thresholds
     - But issue is "broadcasted" down to fundamental toolkit classes
   - And because of the control at tracking time
     - And what about the usefulness of the `GoodForTracking` flag ?

3. Why having a "production cut" for e^+ –issued from a non-divergent process- and not for all other particles and processes ?
   - The argument for the mountain rock case can be made to any particle

4. Could we consider a simpler scheme ?
   - Delegating to the few processes concerned by divergence or heavy productions the full responsibility of handling "their" threshold issue
     - Which does not prevent to have centralized tools to configure the cuts
       - But without intervention of other entities at tracking time
     - And foresee some dedicated tests to check the proper working of these processes
       - By using a simple stepping action rather than letting manager caring of this.
   - Leaving open to all non-divergent processes the opportunity to define production cuts (as for e^+) if they wish or can ?

Cuts & Kernel classes

Cuts in particles category

- `G4ParticleDefinition` allows particles to remember if they are subjects to cuts:
  - Public methods:

```
void   SetApplyCutsFlag(G4bool);
G4bool GetApplyCutsFlag() const;
```

  - Implementation:

```
void G4ParticleDefinition::SetApplyCutsFlag(G4bool flg)
{
  if(theParticleName=="gamma"
  || theParticleName=="e-"
  || theParticleName=="e+"
  || theParticleName=="proton")
  { fApplyCutsFlag = flg; }
  else
  {
    G4cout
     << "G4ParticleDefinition::SetApplyCutsFlag() for " << theParticleName
     << G4endl;
    G4cout
     << "becomes obsolete. Production threshold is applied only for "
     << "gamma, e- ,e+ and proton." << G4endl;
  }
}
```

  - Note also the typedef G4ParticleWithCuts:
    - `typedef G4ParticleDefinition G4ParticleWithCuts;`
    - Used in some places.
- But it looks that `SetApplyCutsFlag(G4bool flg)` is never called !
  - at least for FTFP_BERT, FTFP_BERT_LIV/PEN & QBBC
  - Making `G4bool GetApplyCutsFlag()` always false

Cuts in run category

- Cut flags for `G4ParticleDefinition` objects are set in the physics list base class:

```
void G4VUserPhysicsList::SetApplyCuts(G4bool value, const G4String& name)
{
(…)
  if(name=="all") {
    theParticleTable->FindParticle("gamma")->SetApplyCutsFlag(value);
    theParticleTable->FindParticle("e-")->SetApplyCutsFlag(value);
    theParticleTable->FindParticle("e+")->SetApplyCutsFlag(value);
    theParticleTable->FindParticle("proton")->SetApplyCutsFlag(value);
  } else {
    theParticleTable->FindParticle(name)->SetApplyCutsFlag(value);
  }
}
```

- This is this method which does not look to be called
  - hence leaving all G4ParticleDefinition objects with `GetApplyCutsFlag()` being false.
- **Be reassured, this does not mean we don't have cuts in Geant4 !** ;)
  - Processes use the G4ProductionCuts and G4MaterialCutCouple objects to get the cut values
- But this makes void the control of conformance of secondary particles to cuts at tracking time…

Cuts in tracking category

- The stepping manager DoIt methods:
  - `void G4SteppingManager::InvokeAtRestDoItProcs()`
  - `void G4SteppingManager::InvokeAlongStepDoItProcs()`
  - `void G4SteppingManager::InvokePostStepDoItProcs()`
    - `void G4SteppingManager::InvokePSDIP(size_t np)`
- call for each secondary created by the current process
  - the method `ApplyProductionCut( secondary )`
    - If cuts apply to this secondary particle type
  - Stepping manager snippet code involved:

```
if(tempSecondaryTrack->GetDefinition()->GetApplyCutsFlag())
{ ApplyProductionCut(tempSecondaryTrack); }
```

- `ApplyProductionCut` method:
  - Checks if the (secondary) track conforms to production cuts
  - Two cases:
    - If the track is set “GoodForTracking” by the process, it is accepted anyway
      - Use case: production near boundary
      - Mainly (and likely only) for EM processes
    - Otherwise if its energy is below the cut, it is set to zero kinetic energy, transferring the energy to local deposit
      - And will be later killed if not AtRest processes are attached to it.

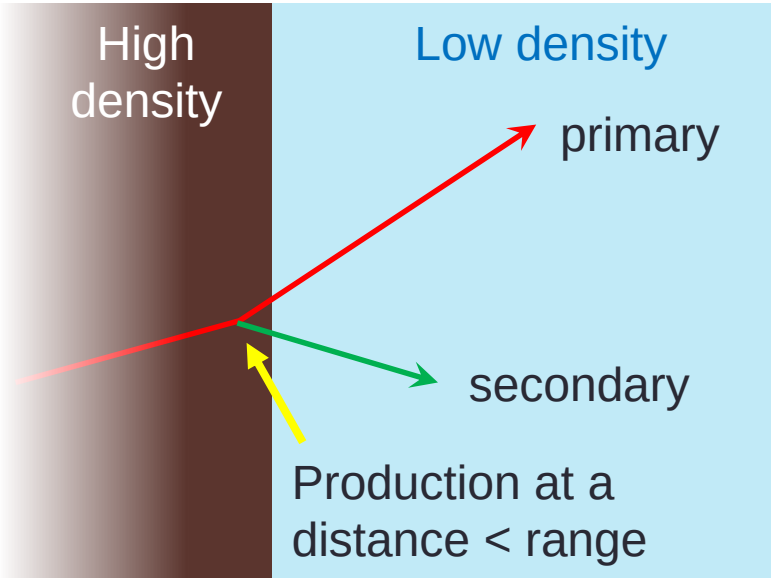
- But, because of the previous issue, `ApplyProductionCut(…)` is not called
  - And the control at tracking time is not effective

Stating on Present Scheme

Stating on present scheme : machinery

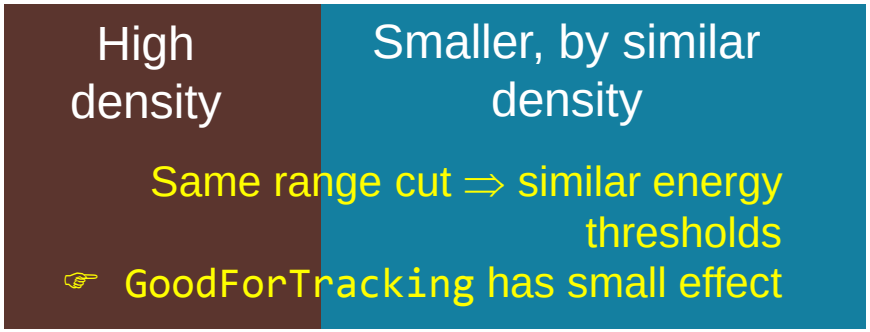
1. Where are we are compared to the initial scheme ?
   - Apparently far…
   - `G4ParticleDefinition` is not used anymore to define the particle with cuts.
     - Making ineffective the stepping manager conformance check implemented in the `ApplyProductionCut( secondary )` method !
   - In addition, the `GoodForTracking` flag is not used:
     - Was meant to allow production of secondary tracks below threshold (in a “high” density material) that could escape into a low density volume next to it
     - Was not giving much improvement, as reported by Vladimir.

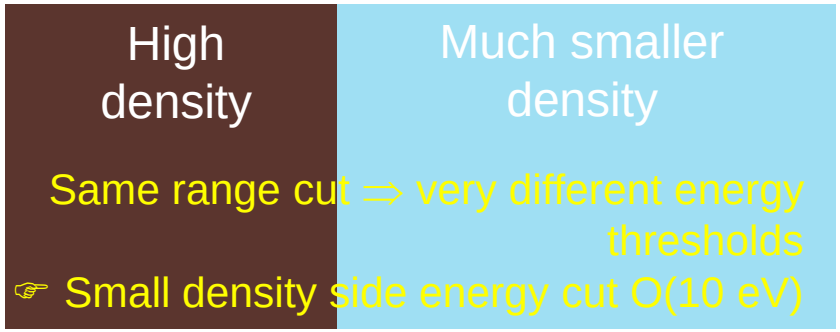
☞ So, in practice, the stepping manager is void wrt threshold conformance checks.



Stating on present scheme : relevance

2. Isn't this scheme "overkilling" ?
   - Because only a few processes need thresholds
     - But issue is "broadcasted" down to fundamental toolkit classes
     - ☞ We see there is no fundamental need to push the issue to fundamental toolkit classes
       - For example, the G4HadronElasticProcess lives without that.
   - And because of the control at tracking time
     - And what about the usefulness of the `GoodForTracking` flag ?





   - The issue of simulating properly the interface == the issue of simulating properly the lower energy demand
     - Physics processes have the knowledge of their capability to serve the low energy demand
     - The tracking can't judge by itself
     - ☞ The `GoodForTracking` flag looks irrelevant.

Stating on present scheme : non-divergence

3. Why having a "production cut" for e^+ –issued from a non-divergent process- and not for all other particles and processes ?
   - The argument for the mountain rock case can be made to any particle
   - Note : the non-produced for e^+ are accounted for local energy deposit

- **Why for e^+ only at kernel level ?**
  - My answer : I can't see why…

- We should either:
  - Get the kernel rid of e^+ threshold
  - Or have the kernel able to provide threshold for all particles

Considering an Other Scheme

Proposal

- Kernel classes are offloaded from cuts control
  - Including control at tracking time
  - Classes involved: `G4ParticleDefinition, G4SteppingManager`
- Processes are given the full responsibility to manage their production thresholds
  - Whatever if this is due to divergences or not
  - With possibly, and preferably, common tools, shared among packages, to expose the configuration to the user
- The machinery for material-cut couple becomes extendable:
  - It has the set $\{e^-, \gamma$ and $p\}$ by default
  - But is extendable to any other type of particle
- Dedicated tests are added to check for conformance of secondary production
    - A test using a simple user stepping action could do it
- Backward compatibility should be considered as well
  - At least for some time