REVIEW OF PRODUCTION THRESHOLDS

Plenary 5

Marc Verderi
LLR/Ecole polytechnique
Jefferson Lab Collaboration Meeting
September 2019

Layout

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

Introduction

Production Thresholds: initial scheme

- Production thresholds (aka “cuts”) were initially considered in RD44/Geant4 as an issue fundamental enough to be addressed at the kernel classes level:
  1. With a mandatory definition in `G4VUserPhysicsList` :

     ```
     virtual void SetCuts() = 0;
     ```

     - that, up to 2011
     - and then a default implementation came for `SetCuts()`.
  2. An explicit declaration of particles subject to cuts in

     `G4ParticleDefinition`

     - With the predefined and fixed set $\{e^-, e^+, \gamma$ and $p\}$
  3. A control at tracking time by the `G4SteppingManager` of the conformance of the produced secondaries wrt to their thresholds
     - Done after each process `DoIt` invocation
     - But allowing exceptions, though, with the “`GoodForTracking`” flag

Particles Under 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	Hadron elastic	Threshold for recoil protons, e.g. n scattering on proton, ejecting it. Threshold defines the "visibility" cut.
Ion	Hadron elastic	Threshold on recoil, as for protons, defined internally in `G4Hadron-ElasticProcess` as (100*keV)*proton_cut_in_mm

- We use cuts for very different reasons:
  - For $e^-$and γ they are essentially unavoidable, vital
  - For proton and (silently) for ions they are physically very important but not vital
  - For e^+they are for convenience for a quite special use-case
    - And if we accept the use-case for e^+ we have to accept it for all particles !
- Promoting these cuts, on the same foot, at the kernel level, is certainly puzzling.

Questions motivating this review

- Isn't this scheme "overkilling" ?
  - Because only a few processes need thresholds
  - And because of the control at tracking after every process `DoIt()`
- Is this scheme effective ?
  - Because of the `GoodForTracking` flag which "offers" to bypass the control anyway
- Why having a "production cut" for e^+ and not for all other particles ?
- Why attaching production cuts to particles while these are essentially a matter of processes ?

- Could we consider a simpler scheme ?
  - Giving full responsibility to the few processes concerned to handle "their" cuts
    - Which does not prevent to have centralized tools to configure the cuts
  - Offloading kernel classes, in particular the `G4SteppingManager`, from this responsibility
  - Leaving open to all processes the opportunity to define cuts (as for e^+) if they wish ?

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.
- `SetApplyCutsFlag(G4bool flg)` is never called by default
  - It is called upon user's request, to extend the application of cuts whatever process produces these secondaries

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 “clean-up” mechanism, which is ~50 lines long.

Each of these methods are 90 – 130 lines long, among which ~50 are for cuts

- `ApplyProductionCut` method:
  - Checks if the secondary 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.

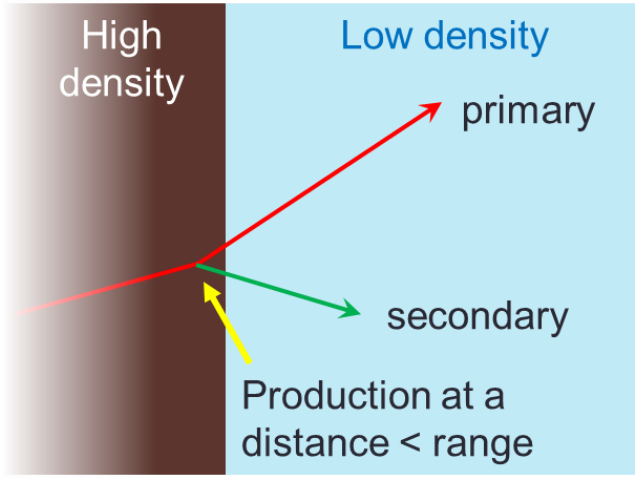
- Mechanism hence activated if `ApplyProductionCut(…)` has been called by the user
  - And is not effective otherwise

Stating on Present Scheme

Stating on present scheme

- By default, this “cleaning” mechanism is not activated:
  - Processes most involve with cuts (ionisation, brem, had elastic) manage their production without it
  - And the code of the mechanism is “dormant”
  - When activated, a big “consumer” is the photoelectric process
    - That terminates the gammas
      - In what case the mechanism terminates the newly borned electrons
    - But any/many more “clients” ?
    - **If not, interest of keeping this mechanism is questionable.**

- What about the motivation for the `GoodForTracking` flag ?
  - It is meant to authorize production of secondary tracks below threshold, near a boundary
  - Issue of simulating properly the interface == issue of simulating properly the lower energy demand
    - The tracking can’t judge by itself !
    - Only the process can know
    - Hence the `GoodForTracking` flag.
  - So `GoodForTracking` appears as a “corrective action” for having activated the mechanism



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
  - Common tools are used to expose the cuts configuration to the user and allow her/him to set it up
  - And if a user needs cuts applied whatever process there are solutions:
    - A stacking action
    - Or a wrapper of the few processes too "lazy" to control production by themselves
- 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