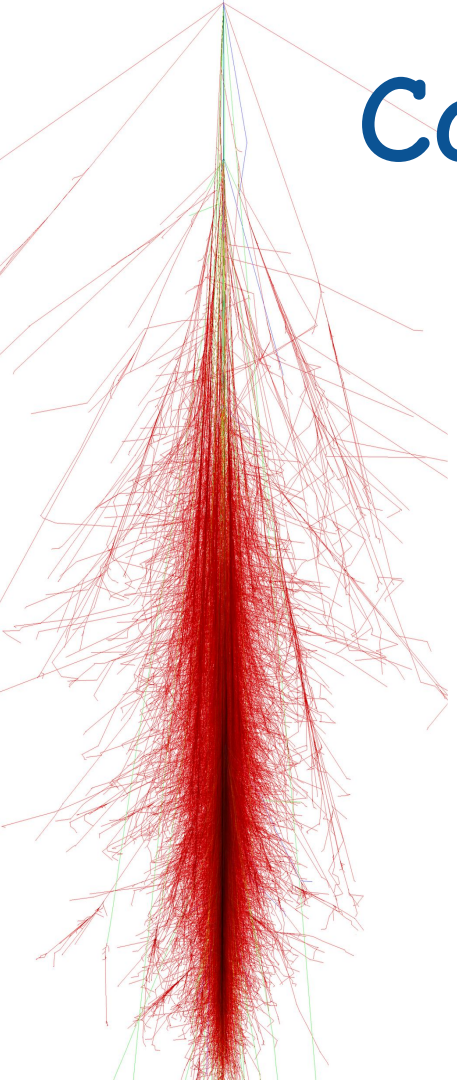


Cascade.inl bug report

local C8 call
22.02.2022

Nikos Karastathis

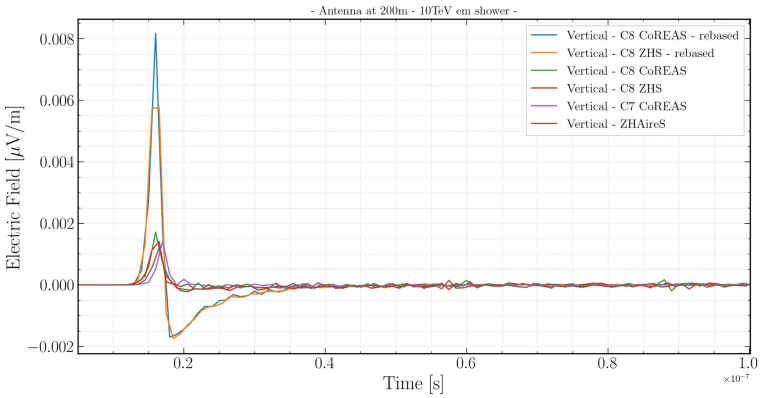
KIT

How did we detect the issue?

master branch

```
// apply all continuous processes on particle + track
if (sequence_.doContinuous(particle, step, limitingId) ==
    ProcessReturn::ParticleAbsorbed) {
  CORSIKA_LOG_DEBUG("Cascade: delete absorbed particle PID={} E={} GeV",
                    particle.getPID(), particle.getEnergy() / 1_GeV);
  if (particle.isErased()) {
    CORSIKA_LOG_WARN(
        "Particle marked as Absorbed in doContinuous, but prematurely erased. This "
        "may be bug. Check.");
  } else {
    particle.erase();
  }
  return; // particle is gone -> return
}
particle.setPosition(step.getPosition(1));
particle.setDirection(step.getDirection(1));
particle.setTime(particle.getTime() + step.getDuration());

if (isContinuous) {
  return; // there is nothing further, step is finished
}
```



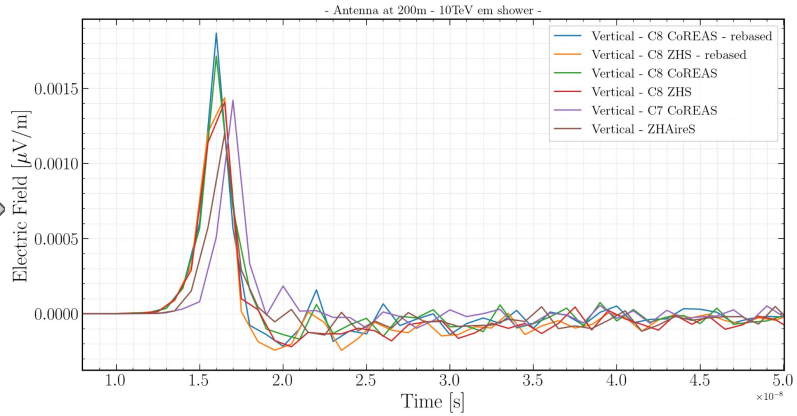
How did we detect the issue?

radio branch

```
particle.setPosition(step.getPosition(1));
particle.setDirection(step.getDirection(1));

// apply all continuous processes on particle + track
if (sequence_.doContinuous(particle, step, limitingId) ==
    ProcessReturn::ParticleAbsorbed) {
  CORSIKA_LOG_DEBUG("Cascade: delete absorbed particle PID={} E={} GeV",
                    particle.getPID(), particle.getEnergy() / 1_GeV);
  if (particle.isErased()) {
    CORSIKA_LOG_WARN(
        "Particle marked as Absorbed in doContinuous, but prematurely erased. This "
        "may be bug. Check.");
  } else {
    particle.erase();
  }
  return; // particle is gone -> return
}
particle.setTime(particle.getTime() + step.getDuration());

if (isContinuous) {
  return; // there is nothing further, step is finished
}
```



Cascade::step

The *step* function does the following:

1. Sample the grammage until the next interaction occurs
2. Determine the track that the particle will take to the next interaction (Taking into account magnetic fields and possible intersections with other media)
3. Apply continuous processes (doContinuous) on the calculated track
4. Sample and calculate interaction (interaction) that happens at the end of the track

```
// apply all continuous processes on particle + track
if (sequence_.doContinuous(particle, step, limitingId) ==
    ProcessReturn::ParticleAbsorbed) {
  CORSIKA_LOG_DEBUG("Cascade: delete absorbed particle PID={} E={} GeV",
                    particle.getPID(), particle.getEnergy() / 1_GeV);
  if (particle.isErased()) {
    CORSIKA_LOG_WARN(
        "Particle marked as Absorbed in doContinuous, but prematurely erased. This "
        "may be bug. Check.");
  } else {
    particle.erase();
  }
  return; // particle is gone -> return
}
particle.setPosition(step.getPosition(1));
particle.setDirection(step.getDirection(1));
particle.setTime(particle.getTime() + step.getDuration());

if (isContinuous) {
  return; // there is nothing further, step is finished
}
```

This is point 3. After doContinuous has been called we can see that position, direction and time of the particle gets updated through the track object (step).

PROPOSAL::ContinuousProcess::doContinuous

```cpp
template <typename TOutput>
template <typename TParticle, typename TTrajectory>
inline ProcessReturn ContinuousProcess<TOutput>::doContinuous(TParticle& vP,
                                                              TTrajectory const& track,
                                                              bool const) {
  if (!canInteract(vP.getPID())) return ProcessReturn::Ok;
  if (track.getLength() == 0_m) return ProcessReturn::Ok;

  // calculate passed grammage
  auto dX = vP.getNode()->getModelProperties().getIntegratedGrammage(track);

  // get or build corresponding track integral calculator and solve the
  // integral
  auto c = getCalculator(vP, calc);
  auto final_energy = (c->second).disp->UpperLimitTrackIntegral(
                        vP.getEnergy() / 1_MeV, dX / 1_g * 1_cm * 1_cm) *
                      1_MeV;
  auto dE = vP.getEnergy() - final_energy;

  // if the particle has a charge take multiple scattering into account
  if (vP.getChargeNumber() != 0) scatter(vP, dE, dX);
  vP.setEnergy(final_energy); // on the stack, this is just kinetic energy, E-m

  // also send to output
  TOutput::write(track, vP.getPID(), dE);

  return ProcessReturn::Ok;
}
```

✅ Updates energy successfully

✅ Updates time successfully

✅ Updates information connected to particle object

❌ Fails to update information connected to the track object

PROPOSAL::ContinuousProcess::scatter

```cpp
template <typename TOutput>
template <typename TParticle>
inline void ContinuousProcess<TOutput>::scatter(TParticle& particle,
                                                HEPEnergyType const& loss,
                                                GrammageType const& grammage) {

  // get or build corresponding calculators
  auto c = getCalculator(particle, calc);

  // Cast corsika vector to proposal vector
  auto particle_dir = particle.getDirection();
  auto d = particle_dir.getComponents();
  auto direction = PROPOSAL::Cartesian3D(d.getX().magnitude(), d.getY().magnitude(),
                                         d.getZ().magnitude());

  auto E_f = particle.getEnergy() - loss;

  // draw random numbers required for scattering process
  std::uniform_real_distribution<double> distr(0., 1.);
  auto rnd = std::array<double, 4>();
  for (auto& it : rnd) it = distr(RNG_);

  // calculate deflection based on particle energy, loss
  auto deflection = (c->second).scatter->CalculateMultipleScattering(
      grammage / 1_g * square(1_cm), particle.getEnergy() / 1_MeV, E_f / 1_MeV, rnd);

  [[maybe_unused]] auto [unused1, final_direction] =
      PROPOSAL::multiple_scattering::ScatterInitialDirection(direction, deflection);

  // update particle direction after continuous loss caused by multiple
  // scattering
  particle.setDirection(
      {particle_dir.getCoordinateSystem(),
       {final_direction.GetX(), final_direction.GetY(), final_direction.GetZ()}});
}
```

✅ *scatter* sets the direction of the particle on the particle object

❌ *scatter* has no access to the track object

Cascade.inl

❌ direction of the particle resulting from *scatter* gets overwritten by step (track object) in Cascade

```cpp
particle.setDirection(step.getDirection(1));
```

This is simply **wrong**!
We don't take into account multiple scattering and the position and direction of the particle are not valid!

Proposed solutions

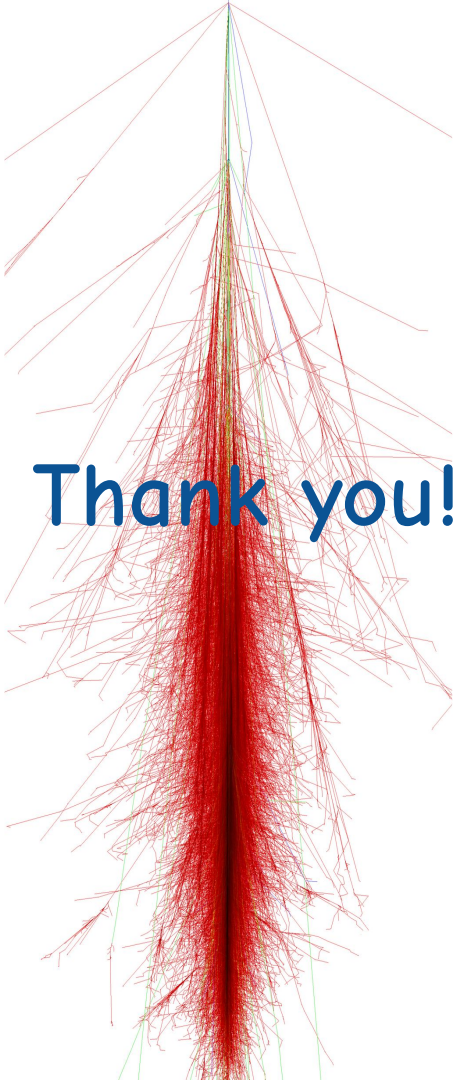
Quick fix (only for now & for validation purposes)

```
if (particle.getDirection() != step.getDirection(1)) {
    // we know that something inside `doContinuous changed,
    // so we don't want to overwrite these changes
} else {
    particle.setDirection(step.getDirection(1));
}
```

Do not touch the Continuous Processes and simply force the direction calculated from PROPOSAL to the particle object if needed. This means altering Cascade.inl a bit and adjusting PROPOSAL::ContinuousProcess ~ not sure this will work.

Long term fix

Work on issue #391 and rethink the structure of continuous processes in C8. Multiple scattering should be treated by the tracking algorithm. Hence, we need another tracking algorithm (?). This highly non-trivial and could really take some time to design and implement. See issue #484 for more ideas about the long term fix.



Thank you!