Update of the attenuation algorithm and a new building pulse algorithm

Minho Kim
Argonne National Laboratory

BIC Simulation Meeting
June 24, 2025

Argonne NATIONAL LABORATORY

Adding propagation time

src/algorithm/calorimeter/SimCalorimeterHitProcessorConfig.h

```
25        // propagation speed of hits in the detector material
26        double propagationSpeed{};
```

src/detector/BEMC/BEMC.cc

```
41      decltype(SimCalorimeterHitProcessorConfig::propagationSpeed) EcalBarrelScFi_propagationSpeed = {
42          160 * edm4eic::unit::mm / edm4eic::unit::ns};
```

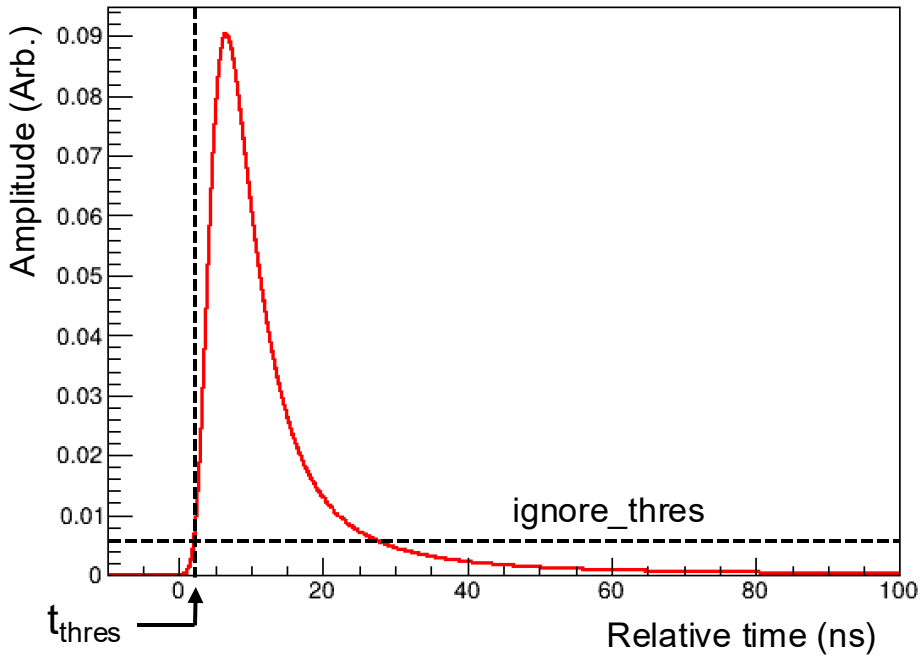
src/algorithm/calorimeter/SimCalorimeterHitProcessor.cc

```
195           const double propaTime =
196               m_attenuationReferencePosition
197                   ? std::abs(m_attenuationReferencePosition.value() - ih.getPosition().z) /
198                         m_cfg.propagationSpeed
199                   : 0.;
200           hit_accum.add(contrib.getEnergy() * attFactor, contrib.getTime() + propaTime,
201                         ih.getPosition());
```

- Since the propagation time and attenuation are coupled effects, the propagation time was applied in the same condition where the attenuation had been applied (m_attenuationReferencePosition ?).

Basic building pulse concept

A template Landau pulse from a hit



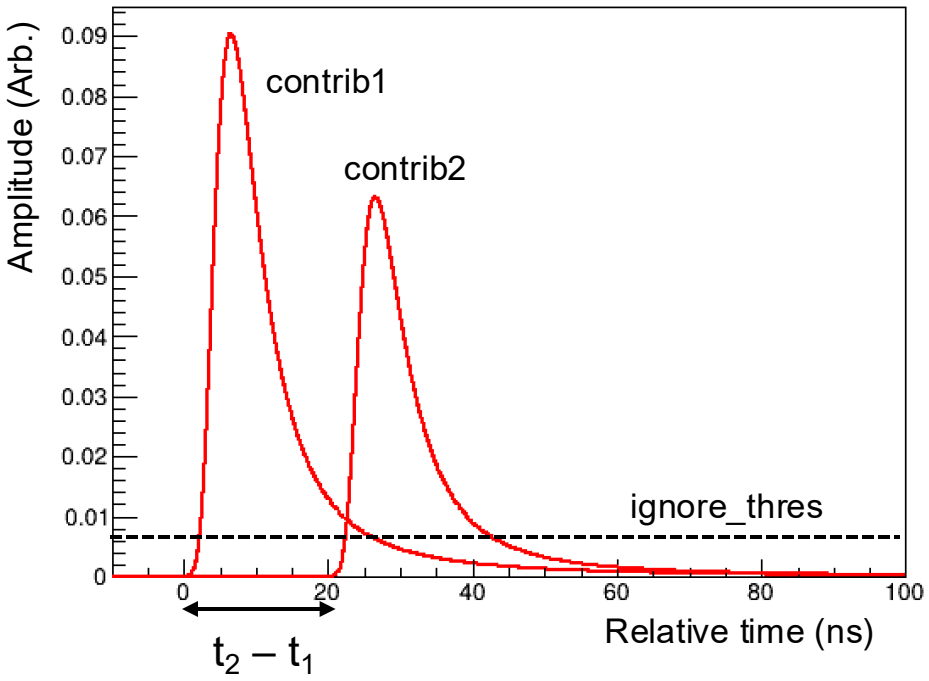
src/algorithm/digi/SiliconPulseGenerationConfig.h

```
14      double ignore_thres                     = 10;
15      double timestep          = 0.2 * edm4eic::unit::ns;
16      double min_sampling_time = 0 * edm4eic::unit::ns;
17      uint32_t max_time_bins   = 10000;
```

- Build pulse so that the amplitude starts rising at 0 s. → The for loop scans the amplitude at fixed time intervals (i * timestep) up to i < max_time_bins. → Break the loop if i > min_sampling_time && amplitude < ignore_thres.
- hit->getTime() + t_{thres} is stored as the time of the pulse.

Applying to contributions

From two contributions



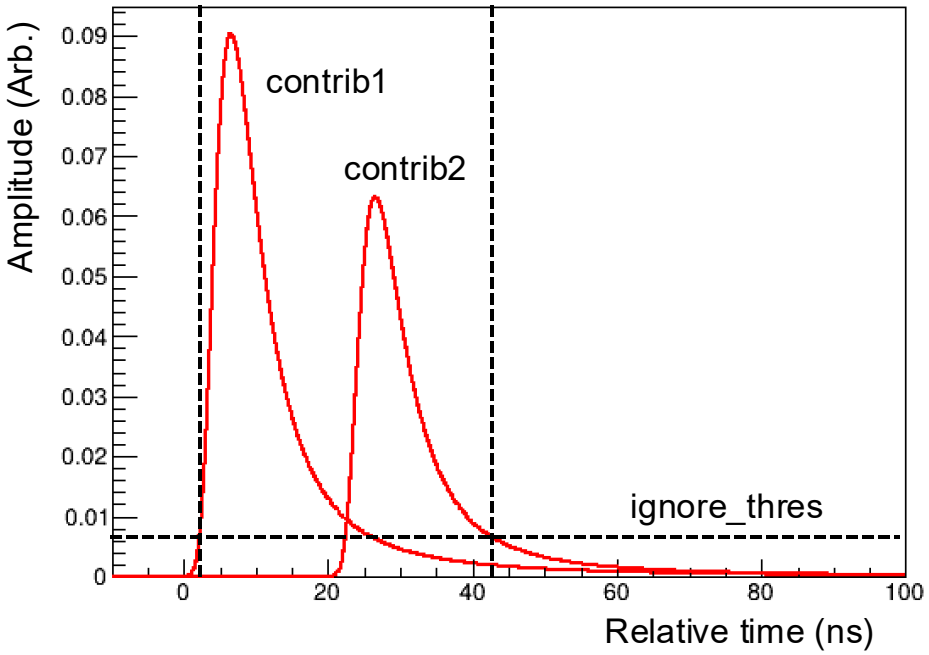
src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

```
for (const auto& sh : *simhits) {
  std::vector<edm4hep::CaloHitContribution> ordered_contribs;

  // fill the contributions in the editable form
  auto contribs = sh.getContributions();
  std::vector<edm4hep::CaloHitContribution> ordered_contribs(contribs.begin(), contribs.end());

  // sort the contributions by time
  std::sort(contribs.begin(), contribs.end(),
            [](const edm4hep::CaloHitContribution& a, const edm4hep::CaloHitContribution& b) {
              return a.getTime() < b.getTime();
            });

  auto earliest_time = contribs.front().getTime();
  auto latest_time   = contribs.back().getTime();
  int max_time_index =
      std::round((earliest_time - latest_time) / m_cfg.timestep) + m_cfg.max_time_bin;
  std::vector<double> amplitudes(max_time_index, 0.);
```

- Regarding the for loop that scans the amplitudes, the loop limit should be fixed and the earliest and latest times are necessary for it. → The contributions were sorted by time for convenience. This is convenient for making a new pulse sum when some contributions are separated in time from the others.
- Create a vector for summing the amplitudes.

Applying to contributions

From two contributions



src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

```
int min_time_index_store   = std::numeric_limits<int>::max();
int max_time_index_store   = 0.;

// build pulses for each contribution and combine them
for (const auto& contrib : sh.getContributions()) {
  double pulse_height = contrib.getEnergy();
  double hit_time     = contrib.getTime();

  // convert energy deposit to npe and apply poisson smearing ** if necessary **
  if (m_edep_to_npe) {
    double npe = pulse_height * m_edep_to_npe.value();
    std::poisson_distribution<> poisson(npe);
    pulse_height = poisson(m_gen);
```

- After iterating the for loop, the size of the vector will be reduced as only the elements bigger than ignore_thres remain.
- The energy deposit is converted to the Npe if m_edep_to_npe has a value. The Poisson smearing is applied accordingly. For the Poisson smearing, std::mt19937 m_gen is declared in the header file.