Update of the attenuation algorithm and a new building pulse algorithm

Minho Kim
Argonne National Laboratory

BIC Simulation Meeting
July 1, 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};
```

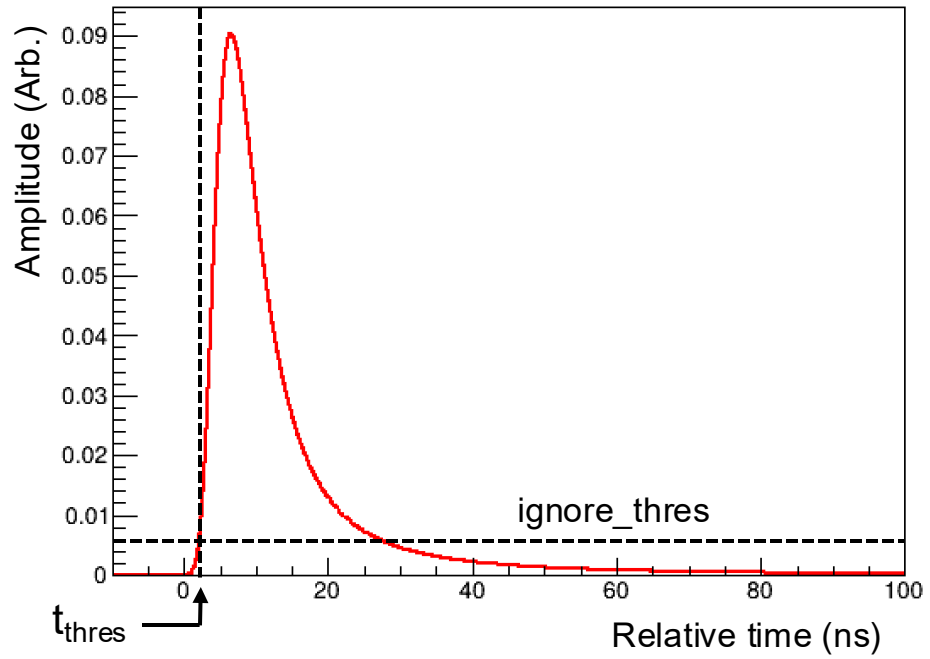
src/algorithm/calorimeter/SimCalorimeterHitProcessor.cc

```
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 ?).
- Default value of the propagationSpeed is currently zero. → Will be updated.

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.

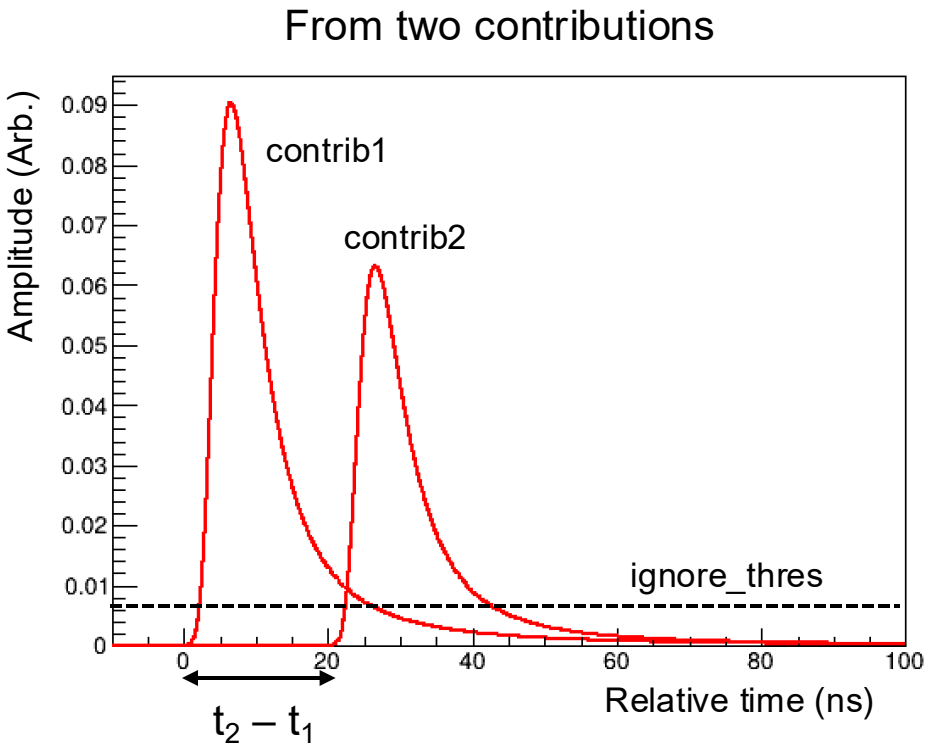
→ The for loop scans the amplitude at fixed time intervals (timestep) up to i < max_time_bins.

→ Break the loop if (i > min_sampling_time && amplitude < ignore_thres).

→ Only the amplitudes above the ignore_thres are stored.

→ hit->getTime() + t_{thres} is stored as the time of the pulse.

Applying to contributions



src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

```
for (const auto& sh : *simhits) {
  // 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(ordered_contribs.begin(), ordered_contribs.end(),
            [](const edm4hep::CaloHitContribution& a, const edm4hep::CaloHitContribution& b) {
              return a.getTime() < b.getTime();
            });

  // Get the earliest and latest contribution times and fix a maximum
  // time bin based on them to scan amplitudes of all the contributions
  // and sum them up.
  double earliest_time = ordered_contribs.front().getTime();
  double latest_time   = ordered_contribs.back().getTime();
```

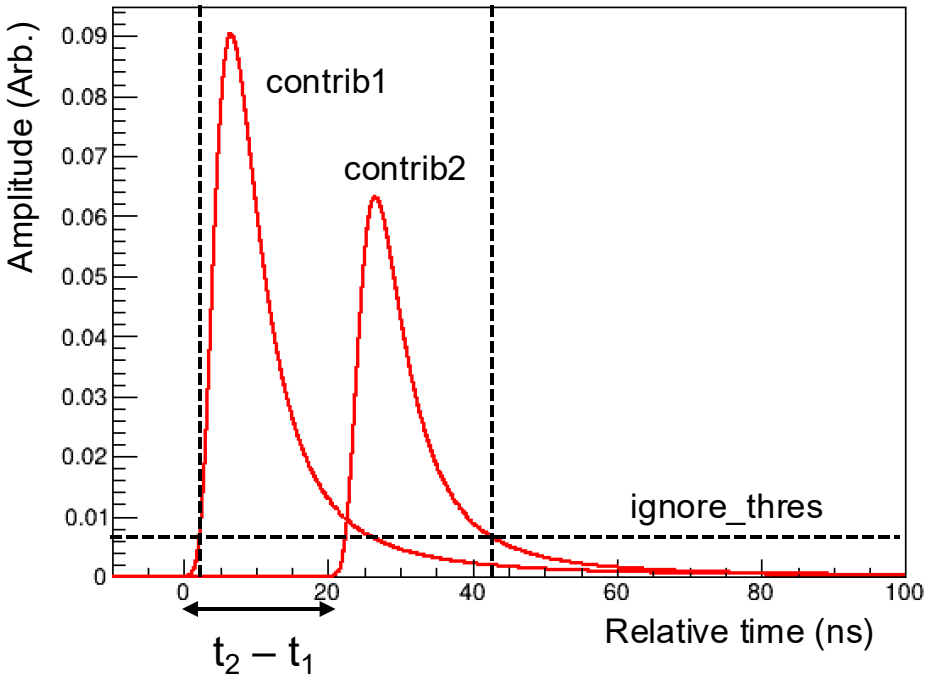
Regarding the for loop that scans the amplitudes, the loop limit should be fixed and the earliest and latest times are necessary for it to calculate $t_2 - t_1$.

→ The contributions were sorted by time first.

→ Get the earliest and latest times.

Applying to contributions

From two contributions



src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

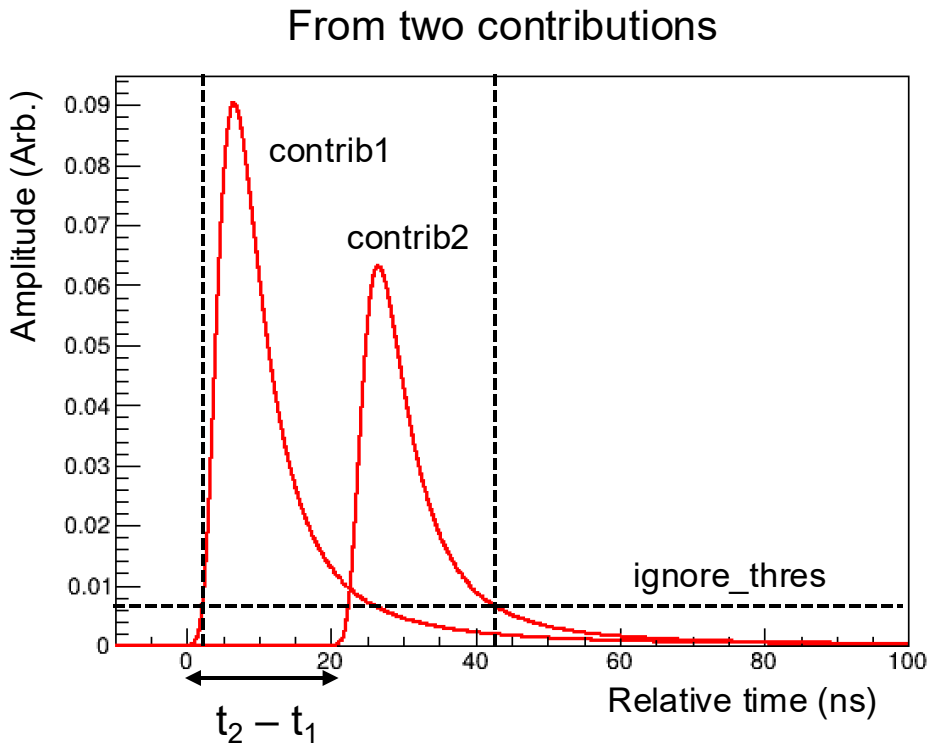
```
int max_time_bin_hit =
    std::round((latest_time - earliest_time) / m_cfg.timestep) + m_cfg.max_time_bin_contrib;

// Summations of the amplitudes will be done within this vector.
std::vector<double> amplitudes(max_time_bin_hit, 0.);

// Finally, the vector size will be reduced so that only the amplitudes
// greater than m_ignore_thres are stored.
int min_time_bin_store = std::numeric_limits<int>::max();
int max_time_bin_store = 0;
```

Fix the maximum time bin to scan all the contributions.

→ Create a vector to accumulate the amplitude of each contribution into the corresponding time bin.

→ The vector size will be reduced so that only amplitudes greater than ignore_thres are stored. Declare the corresponding two time bins.

Applying to contributions



src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

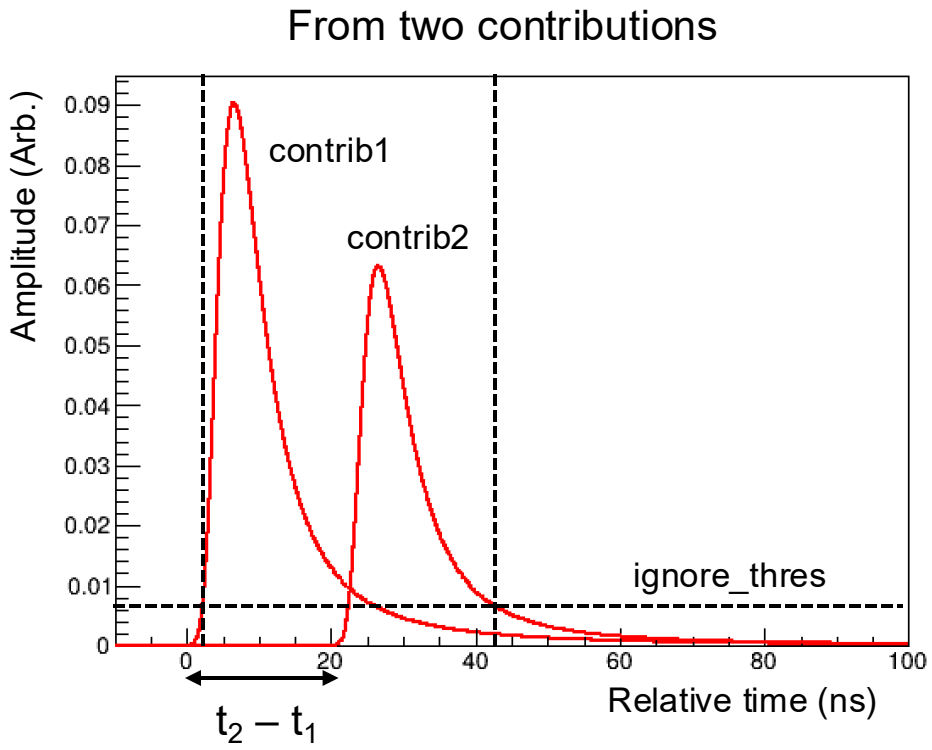
```
for (const auto& contrib : sh.getContributions()) {
  double pulse_height = contrib.getEnergy();
  double time         = contrib.getTime();

  // 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);
  }

  // If the pulse height is lower than m_ignore_thres, it is not necessary to scan it.
  if ((*m_pulse)(m_pulse->getMaximumTime(), pulse_height) < m_ignore_thres)
    continue;
```

In the for-loop, the energy deposit is converted to npe if m_edep_to_npe is defined.

→ Poisson smearing is applied for the same condition.

→ Now, it's time to accumulate the amplitudes of each contribution, but if the pulse height is smaller than m_ignore_thres, that contribution will be skipped.

Applying to contributions



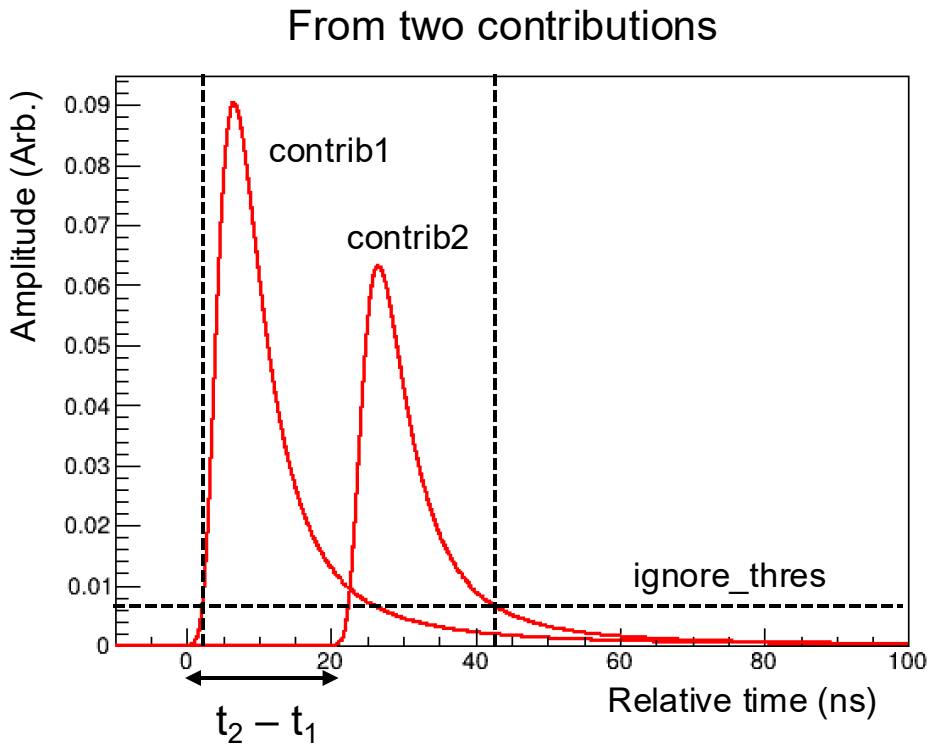
src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

```
for (std::uint32_t i = 0; i < m_cfg.max_time_bin_contrib; i++) {
  double rel_time  = i * m_cfg.timestep;
  double amplitude = (*m_pulse)(rel_time, pulse_height);

  int abs_time_bin = i + std::round((time - earliest_time) / m_cfg.timestep);

  // To find the two indices where the pulse meets the m_ignore_thres
  if (std::abs(amplitude) < m_ignore_thres) {
    if (passed_threshold == false) {
      last_skip_bin = i;
      continue;
    }
    if (rel_time > m_min_sampling_time) {
      max_time_bin_store = abs_time_bin;
      break;
    }
  }

  passed_threshold = true;
  amplitudes[abs_time_bin] += pulse_height;
}
```

Scan each contribution. Get the amplitudes using the relative time bin, find the absolute time bin, and accumulate the amplitude in the vector.

→ Find the two bin values to reduce the vector size.

Applying to contributions



src/algorithm/calorimeter/CalorimeterPulseGeneration.cc

```
    int pulse_time_bin_contrib =
        last_skip_bin + std::round((time - earliest_time) / m_cfg.timestep);
    min_time_bin_store = std::min(min_time_bin_store, pulse_time_bin_contrib);
  }

  // If all the pulse heights are lower than the m_ignore_thres,
  // it is not necessary to store this hit.
  if (max_time_bin_store == 0)
    continue;

  double pulse_time_hit = earliest_time + min_time_bin_store * m_cfg.timestep;
```

Find the two bin values to reduce the vector size.

→ If all the contributions are smaller than m_ignore_thres, this hit will be skipped.

→ The pulse time will be determined.