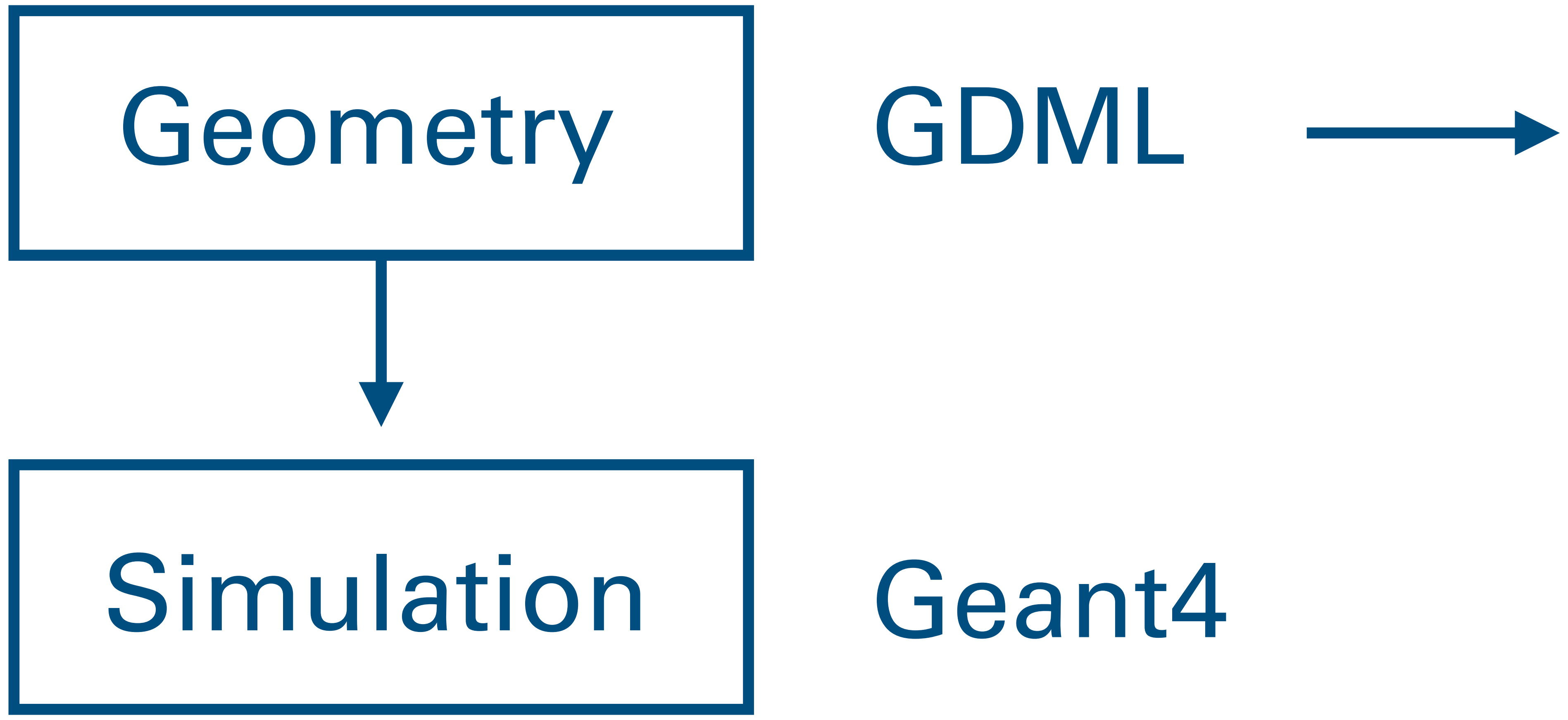
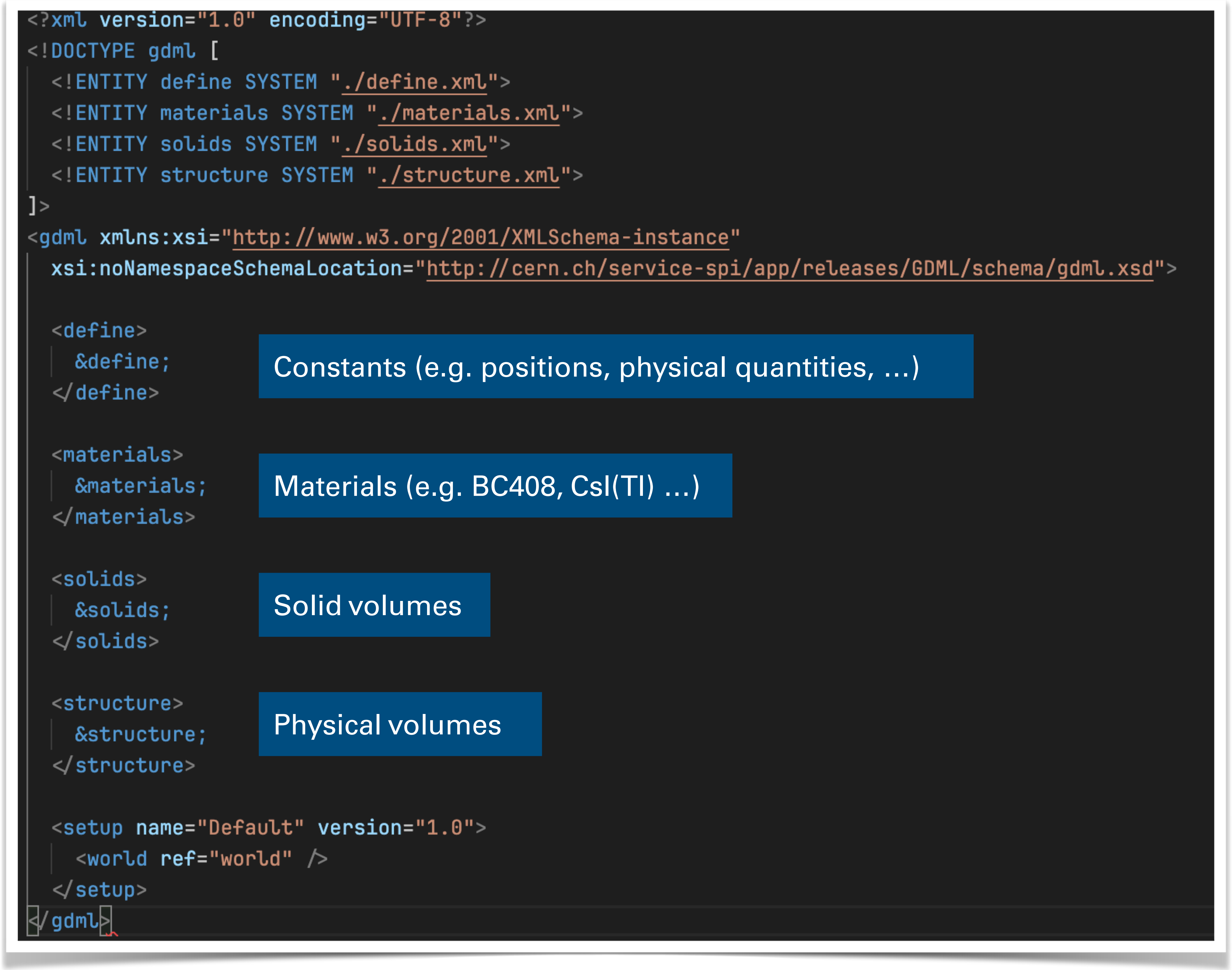
RIPTIDE-SIMRECO

Samuele Lanzi - 10/09/2025

RIPTIDE-G4: Geometry





```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE gdml [
  <!ENTITY define SYSTEM "./define.xml">
  <!ENTITY materials SYSTEM "./materials.xml">
  <!ENTITY solids SYSTEM "./solids.xml">
  <!ENTITY structure SYSTEM "./structure.xml">
]>
<gdml xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="http://cern.ch/service-spi/app/releases/GDML/schema/gdml.xsd">

  <define>
    &define;
  </define>

  <materials>
    &materials;
  </materials>

  <solids>
    &solids;
  </solids>

  <structure>
    &structure;
  </structure>

  <setup name="Default" version="1.0">
    <world ref="world" />
  </setup>
</gdml>
```

Constants (e.g. positions, physical quantities, …)

Materials (e.g. BC408, CsI(Tl) …)

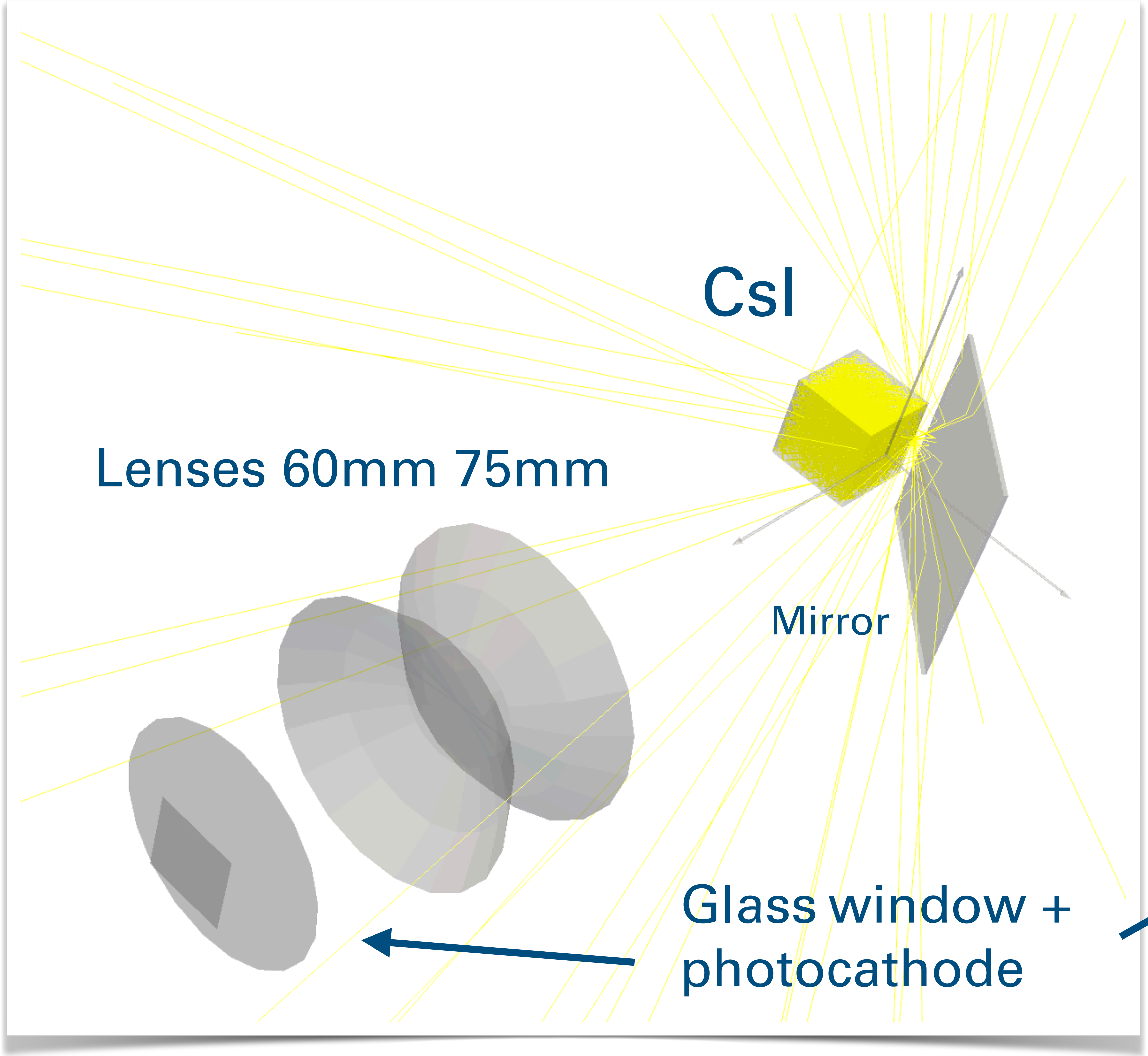
Solid volumes

Physical volumes

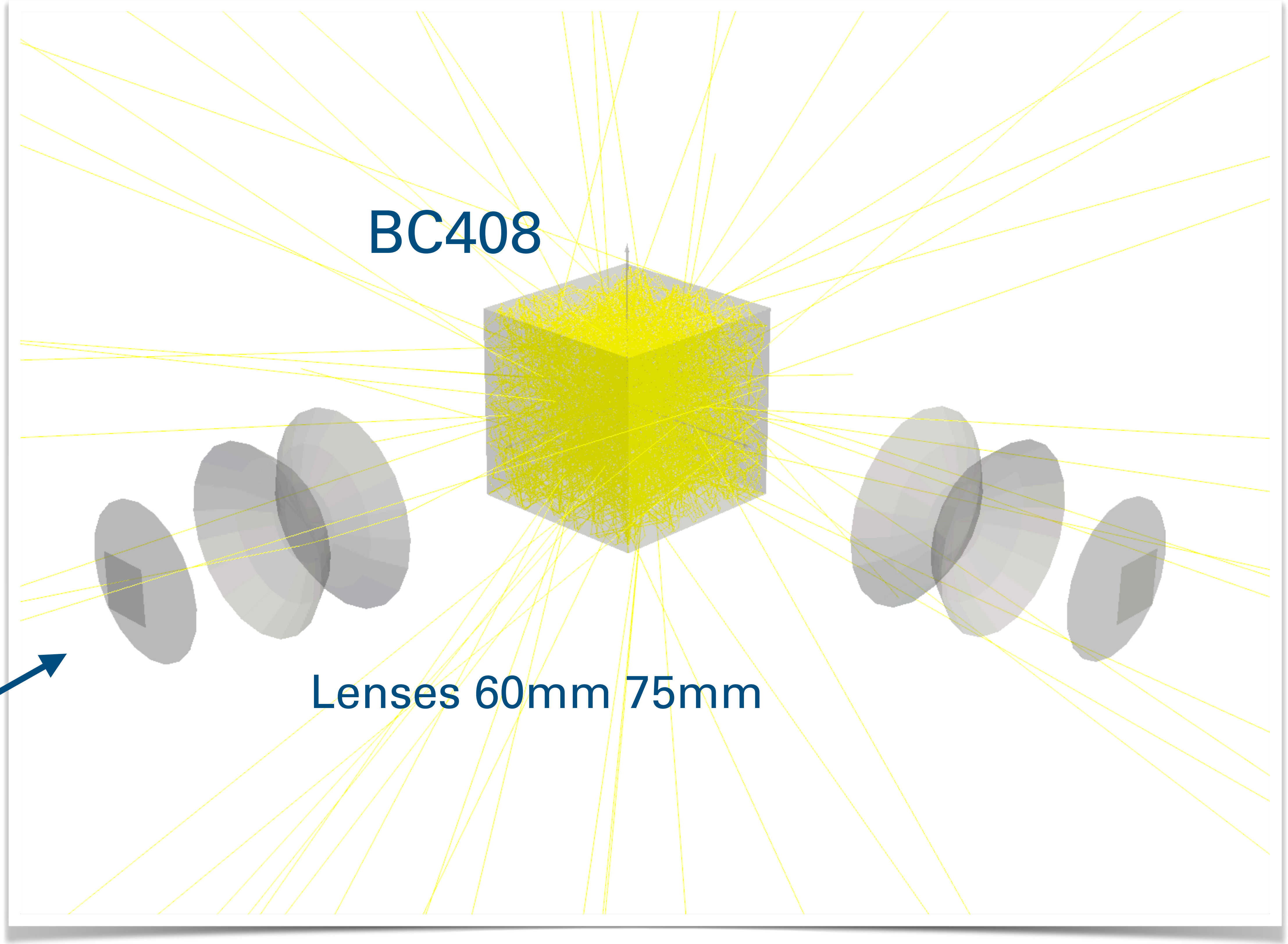
In this way the simulation is independent from the geometry

RIPTIDE-G4: Geometry

Two geometries are developed using GDML:

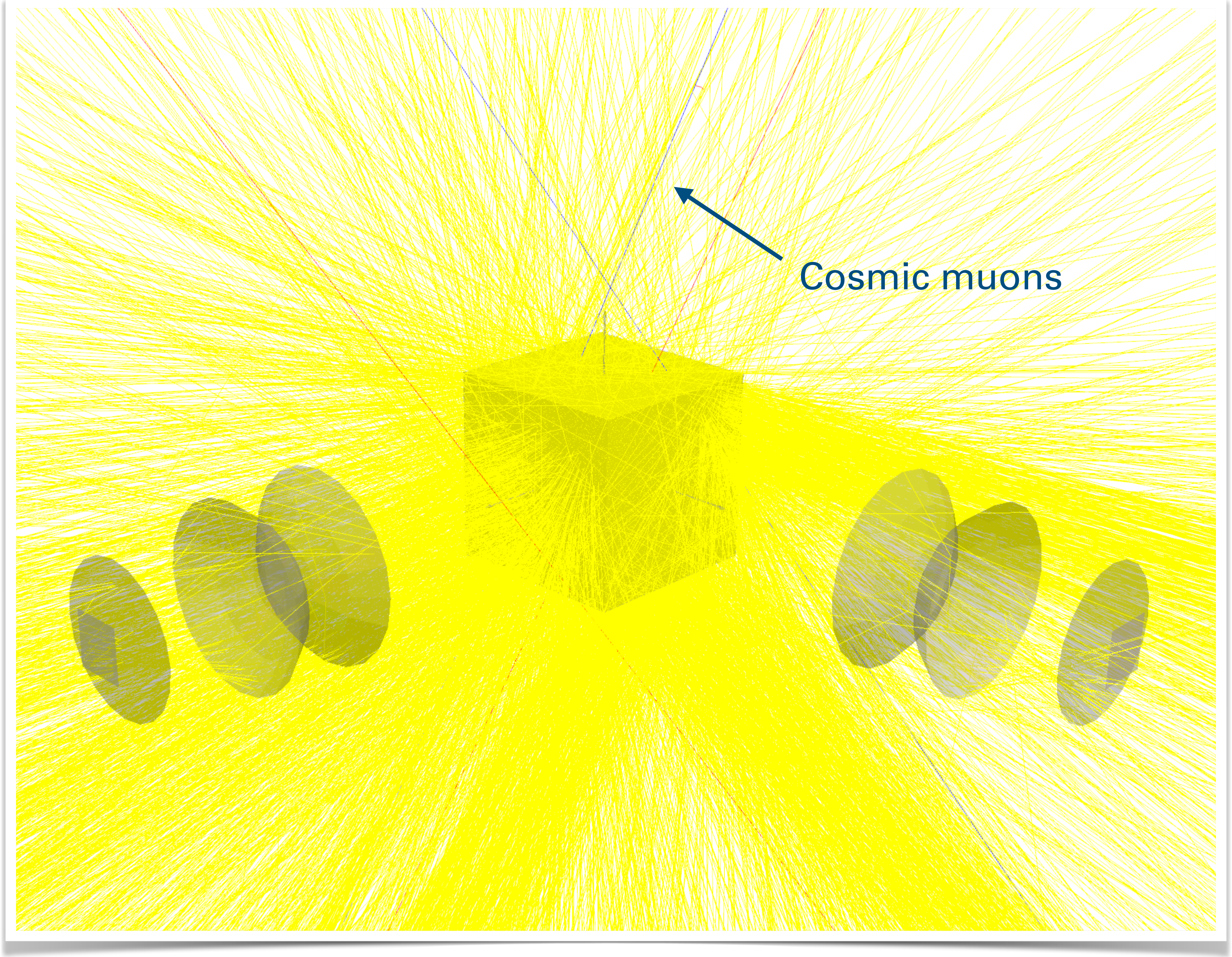


HiCAM FLUO Setup



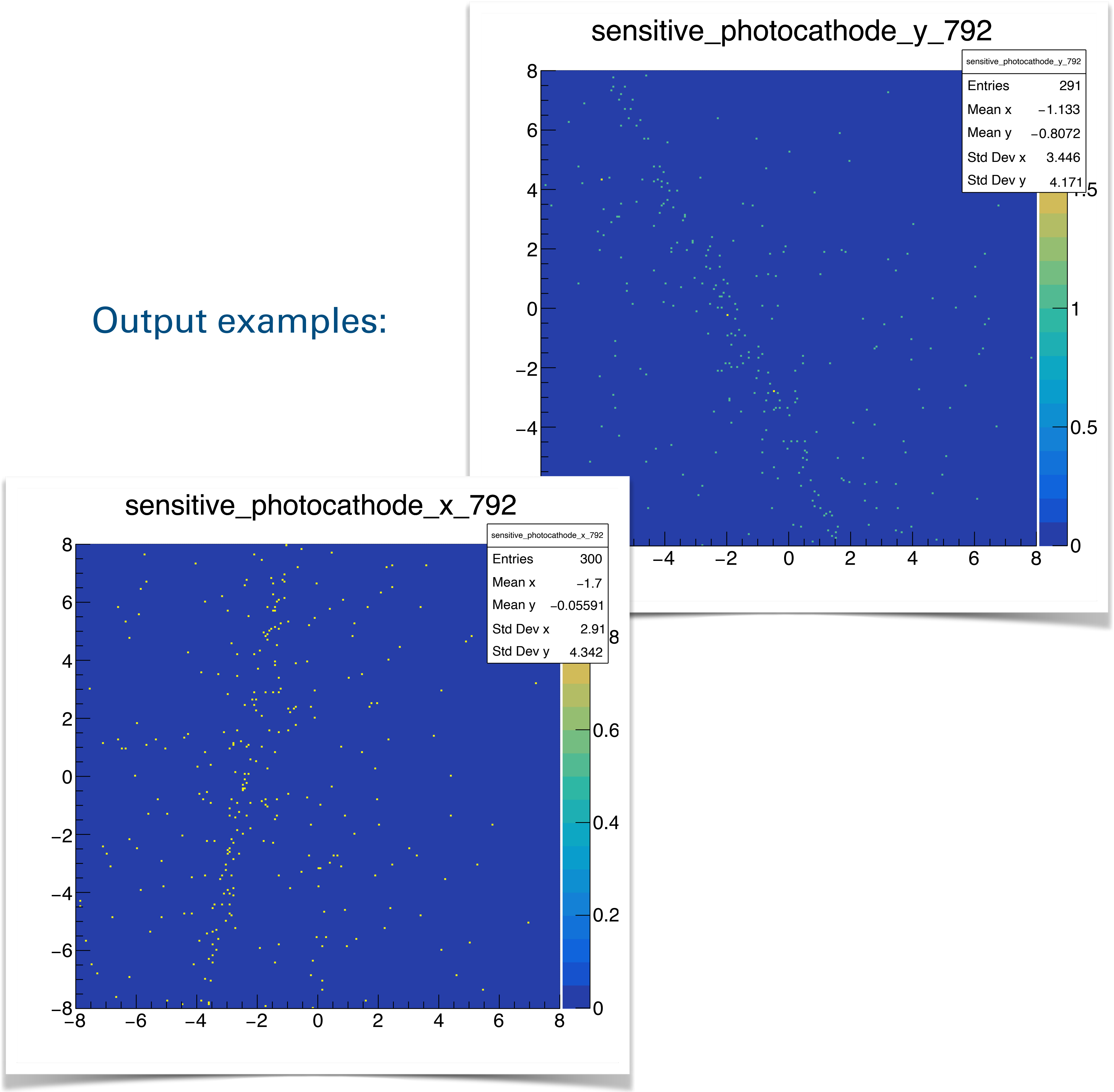
Original Setup

RIPTIDE-G4: Cosmic rays



Cosmic muons

Output examples:



To simulate cosmic muons that contains models on the angular distribution -> CRY_v.1.7 integrated in the CryPrimaryGeneratorAction class

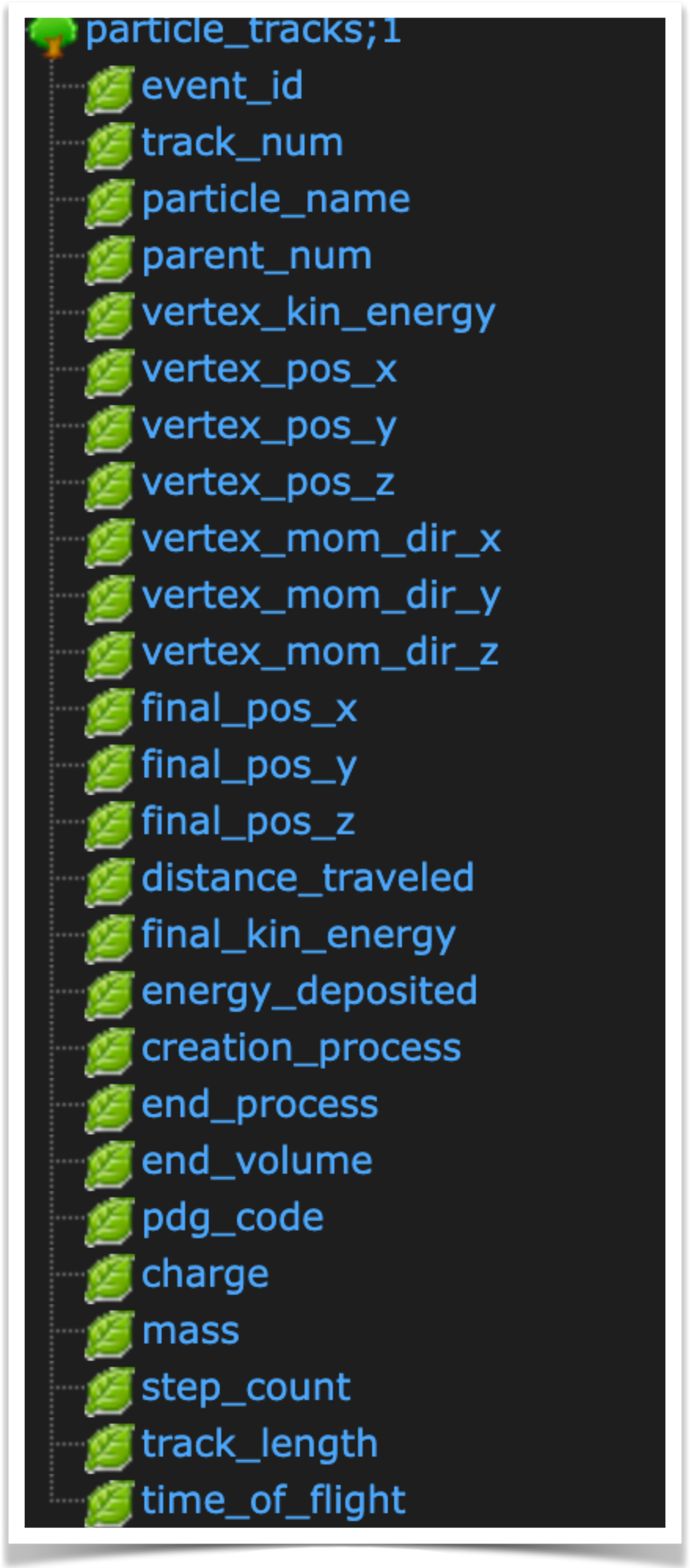
RIPTIDE-G4: Outputs

Trees:

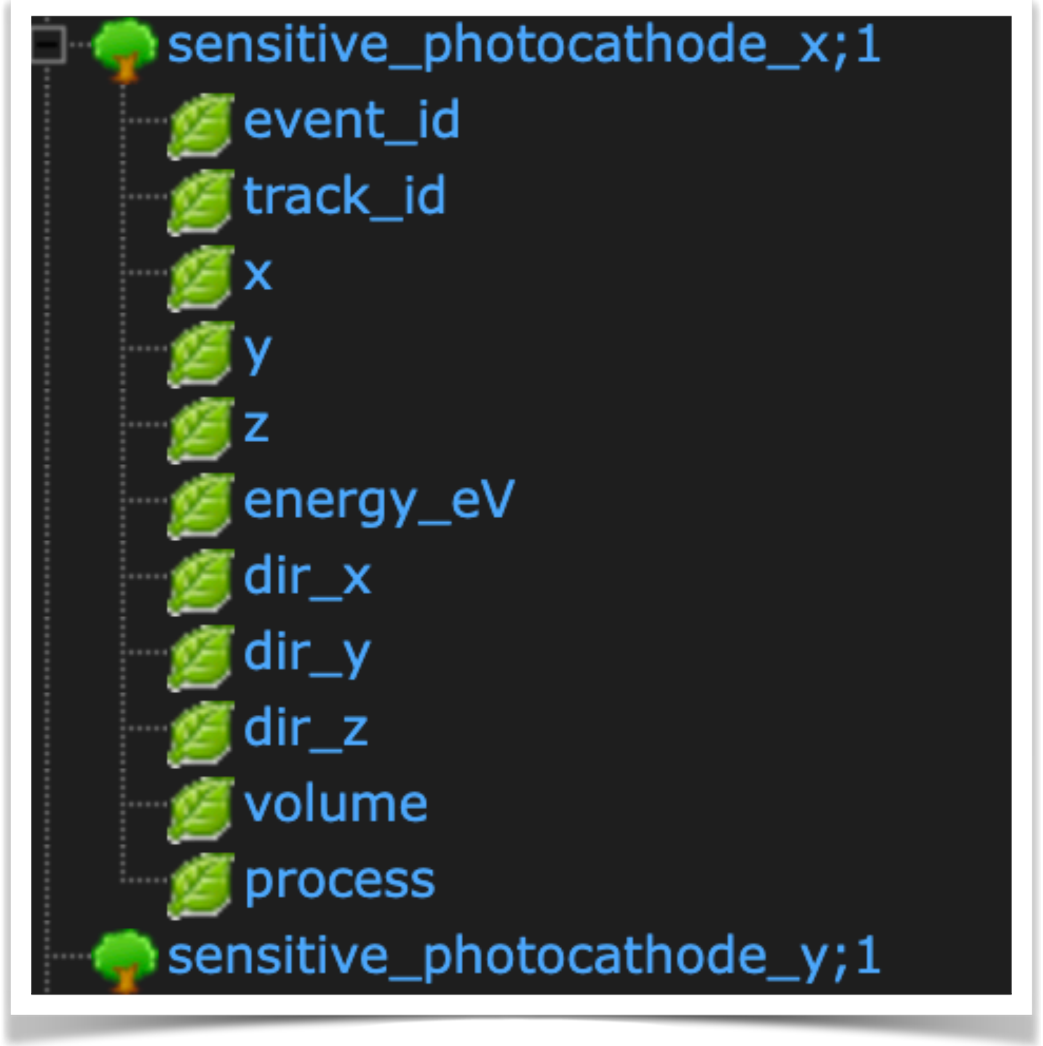
Tracks

Photocathode +x

Photocathode +y



```
particle_tracks;1
    event_id
    track_num
    particle_name
    parent_num
    vertex_kin_energy
    vertex_pos_x
    vertex_pos_y
    vertex_pos_z
    vertex_mom_dir_x
    vertex_mom_dir_y
    vertex_mom_dir_z
    final_pos_x
    final_pos_y
    final_pos_z
    distance_traveled
    final_kin_energy
    energy_deposited
    creation_process
    end_process
    end_volume
    pdg_code
    charge
    mass
    step_count
    track_length
    time_of_flight
```

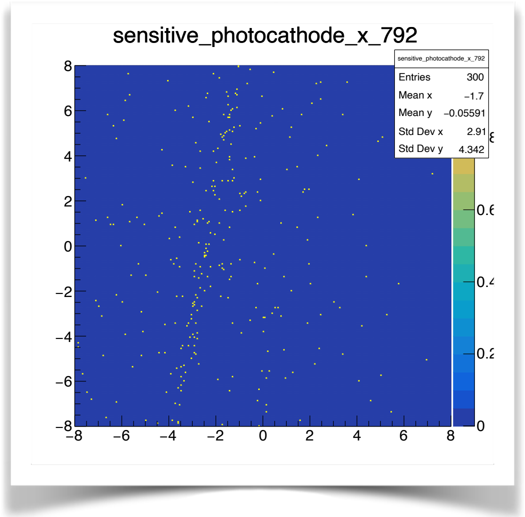


```
sensitive_photocathode_x;1
    event_id
    track_id
    x
    y
    z
    energy_eV
    dir_x
    dir_y
    dir_z
    volume
    process
sensitive_photocathode_y;1
```

DataProvider

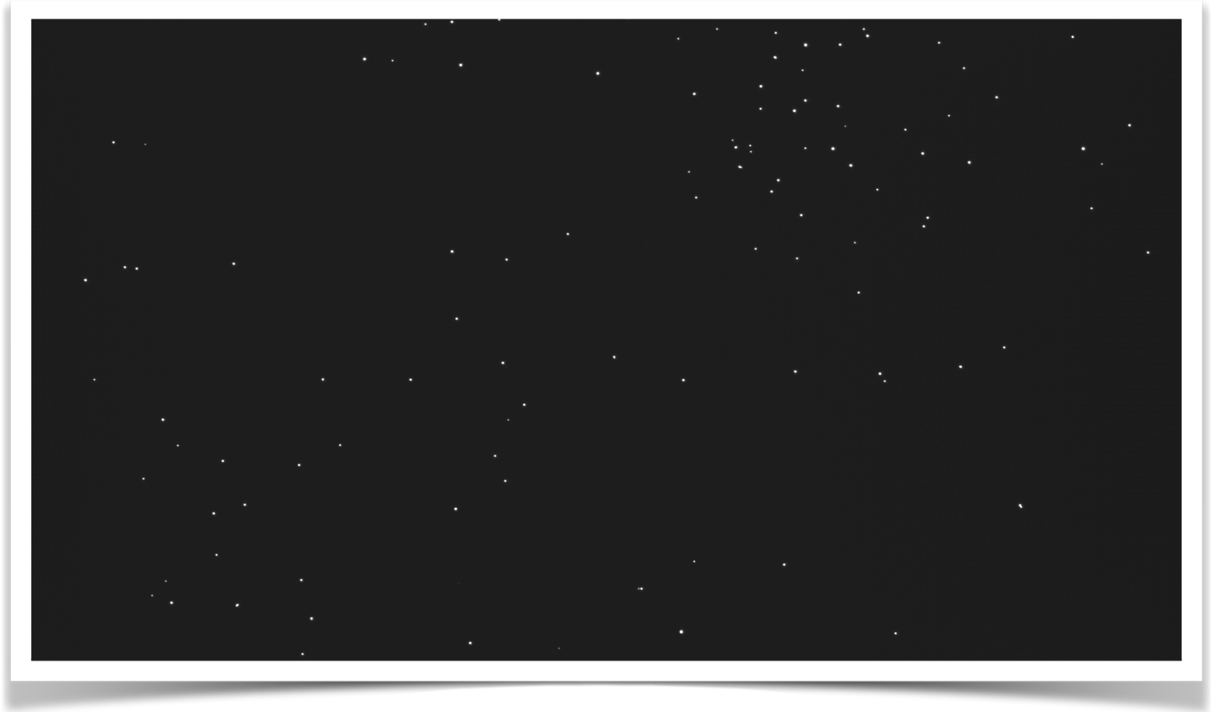
MONTE CARLO DATA

Provides useful informations from Monte Carlo simulations like photon hits in the sensor or track direction truth for reconstruction

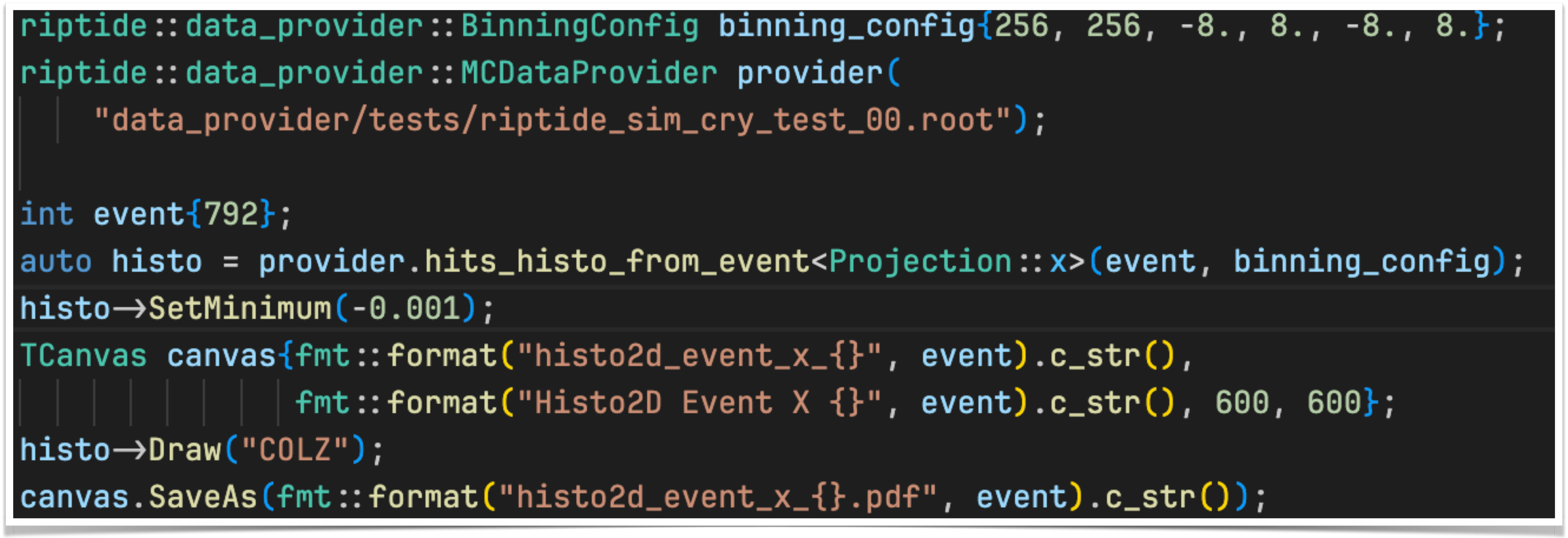


EXPERIMENTAL DATA

Provides frames from binary format provided by Lambert HiCAM (FLIC) and extract matrices for the reconstruction

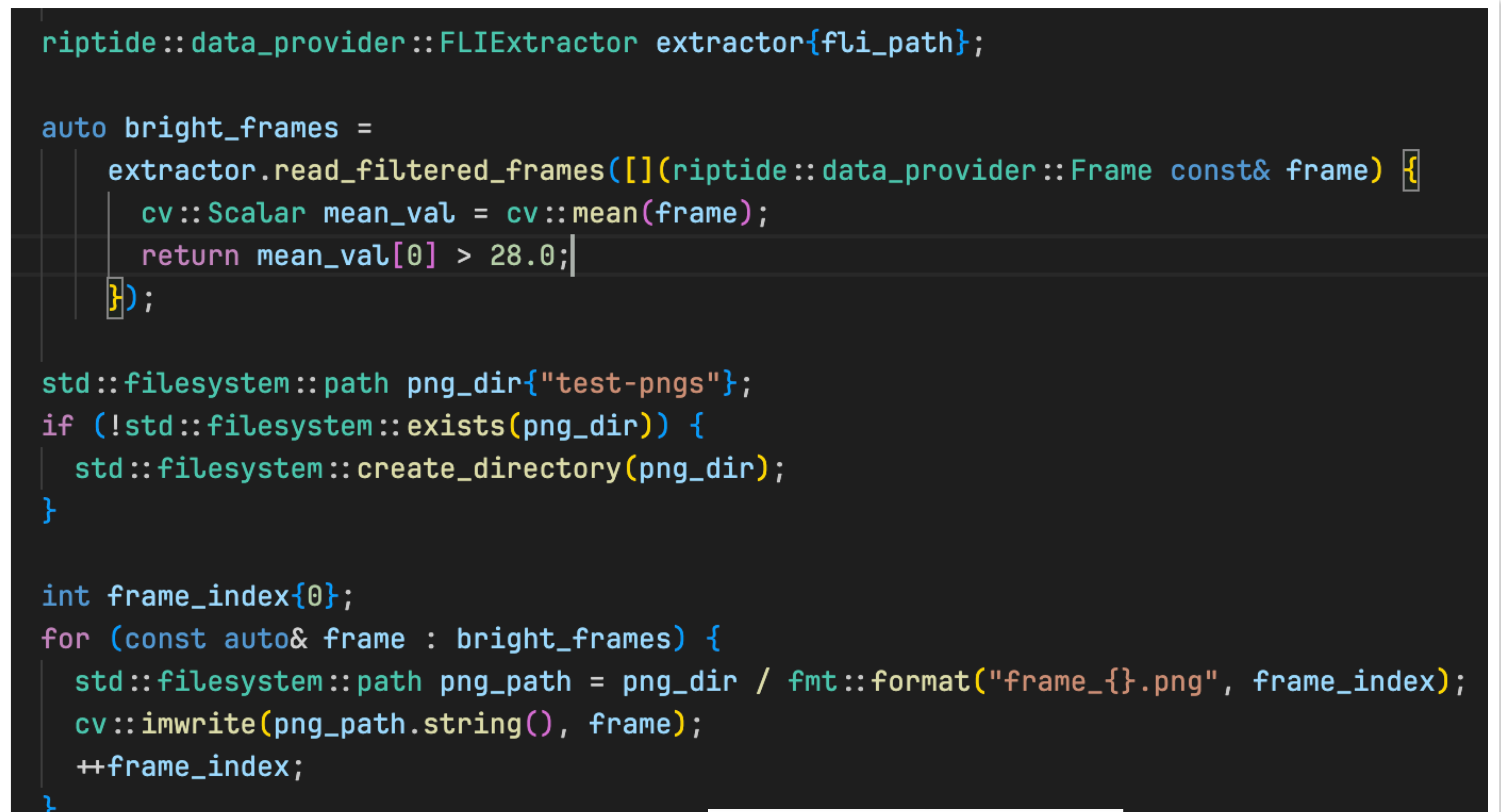


DataProvider: Monte Carlo



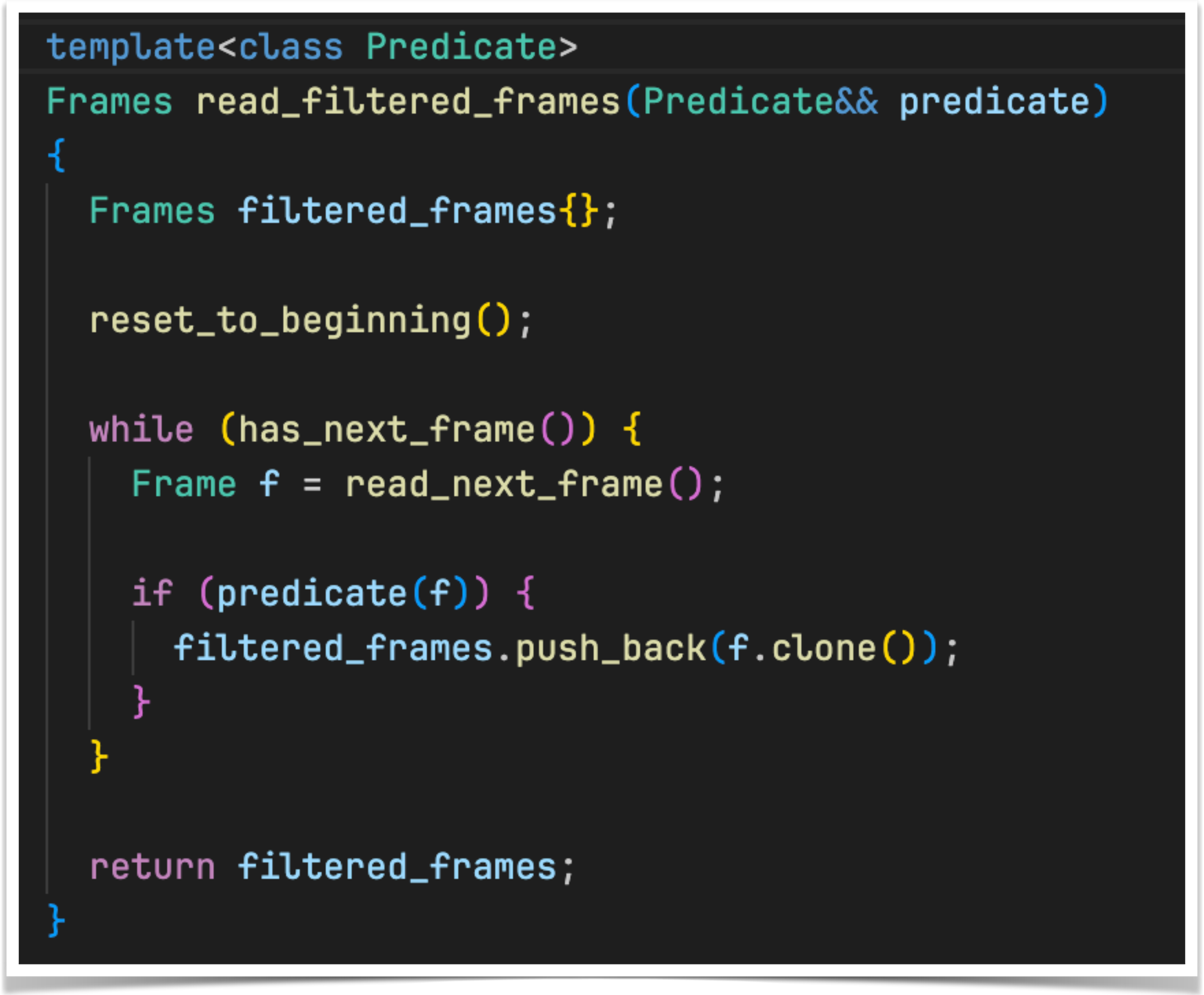
```
riptide::data_provider::BinningConfig binning_config{256, 256, -8., 8., -8., 8.};
riptide::data_provider::MCDataProvider provider(
    "data_provider/tests/riptide_sim_cry_test_00.root");

int event{792};
auto histo = provider.hits_histo_from_event<Projection::x>(event, binning_config);
histo->SetMinimum(-0.001);
TCanvas canvas{fmt::format("histo2d_event_x_{}", event).c_str(),
               fmt::format("Histo2D Event X {}", event).c_str(), 600, 600};
histo->Draw("COLZ");
canvas.SaveAs(fmt::format("histo2d_event_x_{}.pdf", event).c_str());
```

DataProvider: HiCAM FLUO



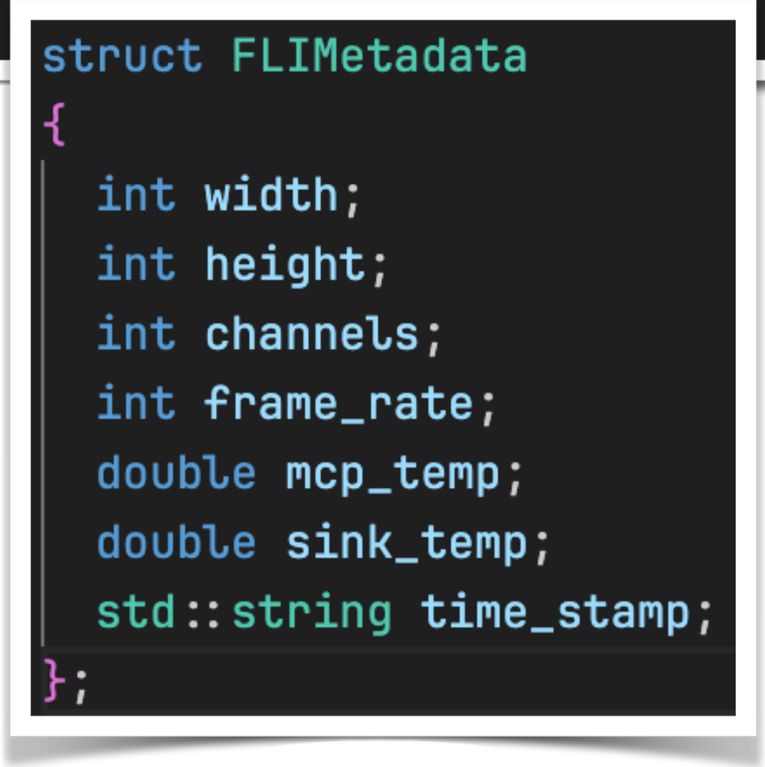
```cpp
riptide::data_provider::FLIExtractor extractor{fli_path};

auto bright_frames =
    extractor.read_filtered_frames([](riptide::data_provider::Frame const& frame) {
      cv::Scalar mean_val = cv::mean(frame);
      return mean_val[0] > 28.0;
    });

std::filesystem::path png_dir{"test-pngs"};
if (!std::filesystem::exists(png_dir)) {
  std::filesystem::create_directory(png_dir);
}

int frame_index{0};
for (const auto& frame : bright_frames) {
  std::filesystem::path png_path = png_dir / fmt::format("frame_{}.png", frame_index);
  cv::imwrite(png_path.string(), frame);
  ++frame_index;
}
```

Trigger threshold:



```cpp
template<class Predicate>
Frames read_filtered_frames(Predicate&& predicate)
{
  Frames filtered_frames{};

  reset_to_beginning();

  while (has_next_frame()) {
    Frame f = read_next_frame();

    if (predicate(f)) {
      filtered_frames.push_back(f.clone());
    }
  }

  return filtered_frames;
}
```

+ Metadata



```cpp
struct FLIMetadata
{
  int width;
  int height;
  int channels;
  int frame_rate;
  double mcp_temp;
  double sink_temp;
  std::string time_stamp;
};
```

Conclusions

- The RIPTIDE Geant4 simulation has been updated to be independent of the geometry.
- The simulation uses GDML to define the geometries.
- Two geometries are currently defined: the classic one and the mirror-scintillation one.
- Cosmic rays are simulated using CRY v1.7.
- A DataProvider is used to extract information from the simulation output files and from experimental binary files.
- The DataProvider also implements a filter to process images in a trigger-like manner.