Chroma for DUNE

Ben Land
June 7, 2021

What is Chroma?

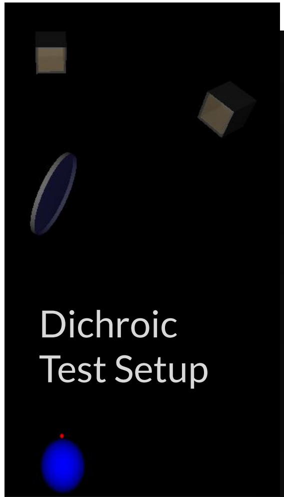
- Python package with CUDA code to propagate optical photons on a GPU
  - Written by S. Seibert and A. LaTorre in 2011
- Represents the geometry as a triangular mesh
  - Leverages well developed optical ray tracing algorithms
  - Much faster tracking compared to Geant4 volume-based approach
- Interfaces with Geant4 to generate photons from physical interactions
  - Photons are killed immediately in Geant4, propagated in Chroma, returned to Geant4
  - *Can also receive arbitrary photon propagation requests over network socket*
  - *Can further do stand-alone optical-only simulations*
- About 200x faster than an equivalent Geant4 simulation
  - Does require a reasonable GPU (supports any GPU with CUDA)
  - GPUs becoming increasingly available for high performance computing
- See the SNOWMASS LOI on Chroma and the Chroma GitHub repository

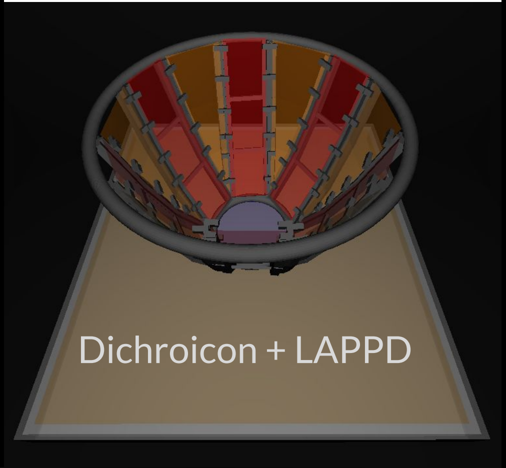
Why use Chroma?

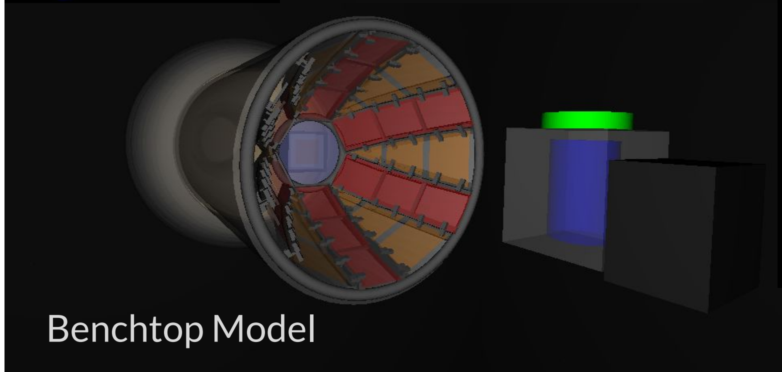
- Specifically for DUNE / liquid argon
    - LAr scintillation produces a *lot* of photons, well suited for GPU acceleration
    - Chroma already supports all critical optical processes
        - Wavelength shifting materials, dichroic filters, etc.
    - Recent work implementing a fully optical LAr detector looks promising
    - *Some work in the last several weeks prepping Chroma for DUNE simulations*
- In general
    - CAD drawings can be directly imported into simulation as STL meshes
    - Fine control over the surface properties of each triangle in mesh
    - Relatively small, easily maintainable Python codebase
    - Up to date with recent Geant4, Root, Boost, and Python3 versions
    - Easy integration into both ROOT and Python analysis frameworks

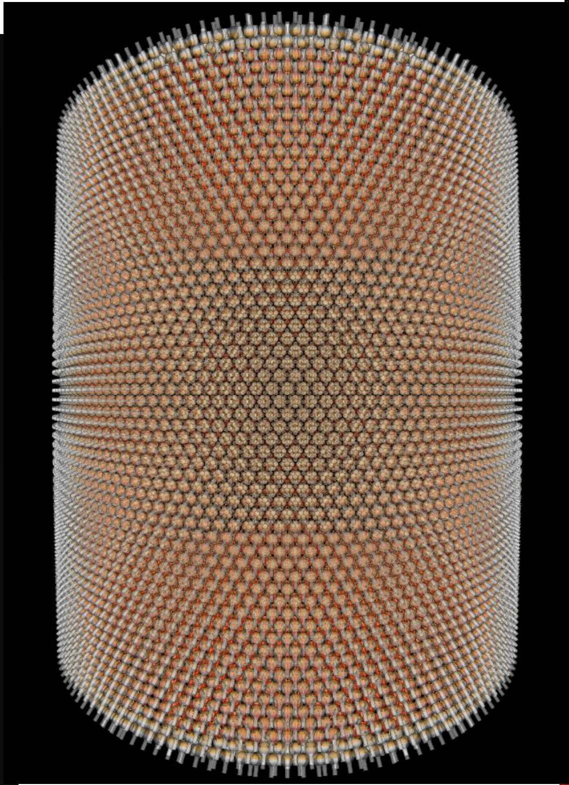
Phys. Rev. D 101, 072002 (2020)

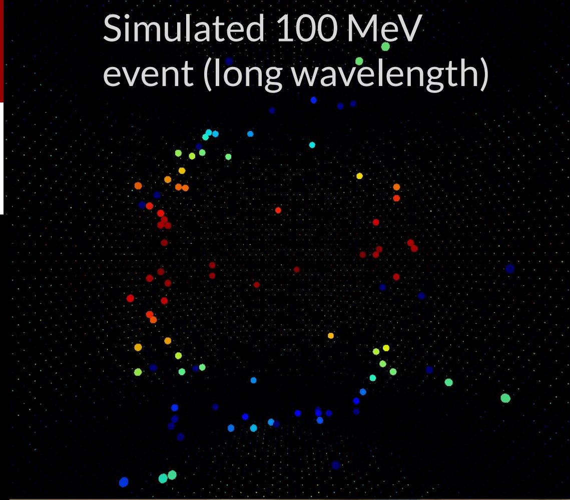
Chroma Example: Dichroicon

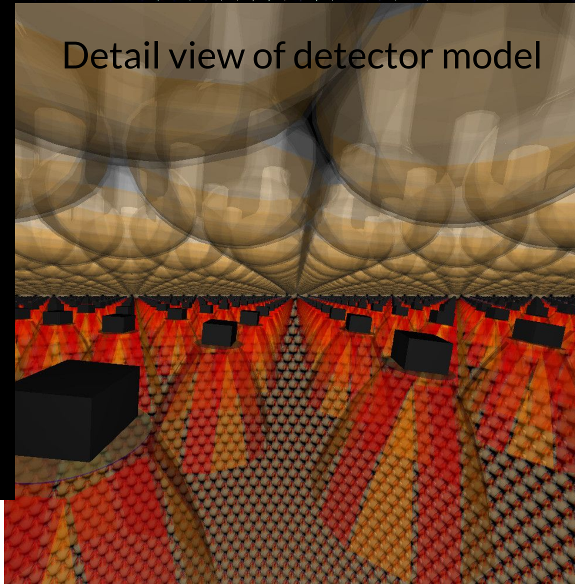

Dichroic Test Setup


Dichroicon + LAPPD

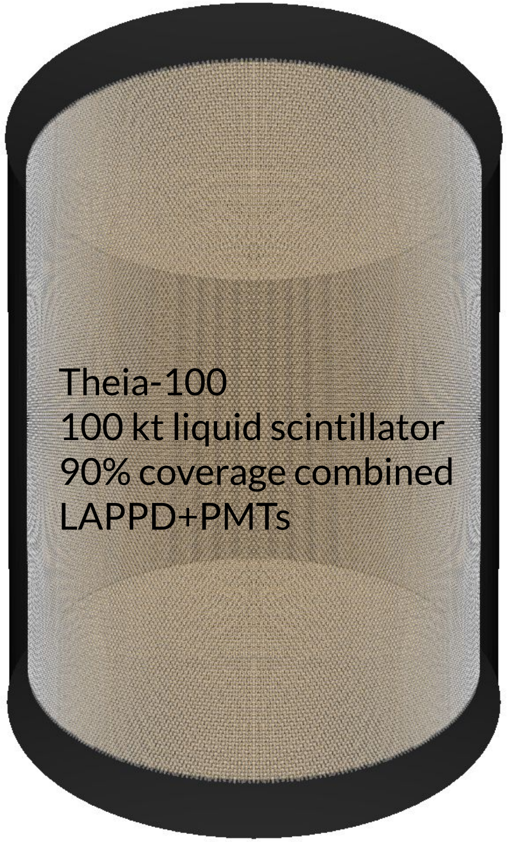

Benchtop Model

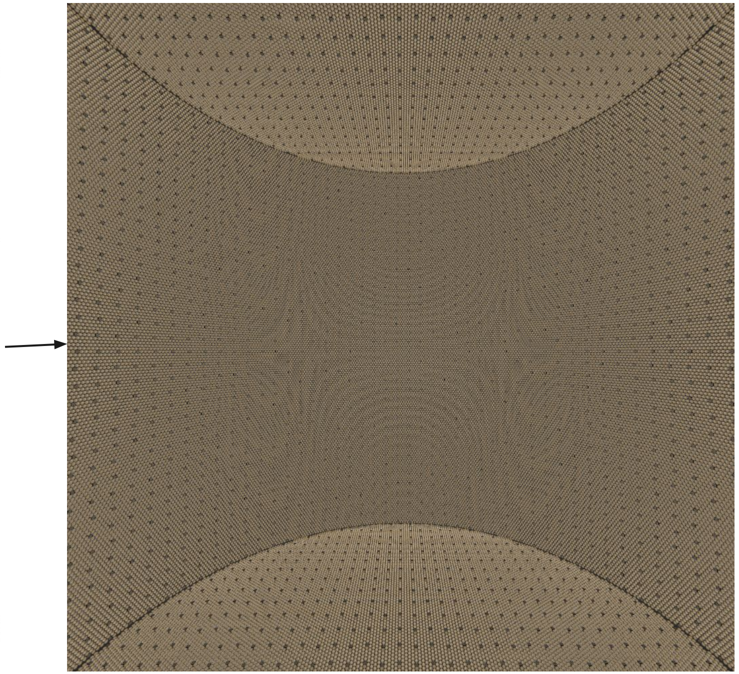

1 kt pure scintillator
90% coverage of dichroicons

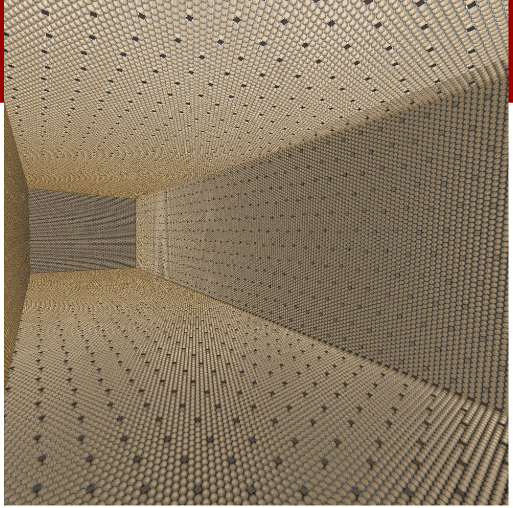

Simulated 100 MeV event (long wavelength)

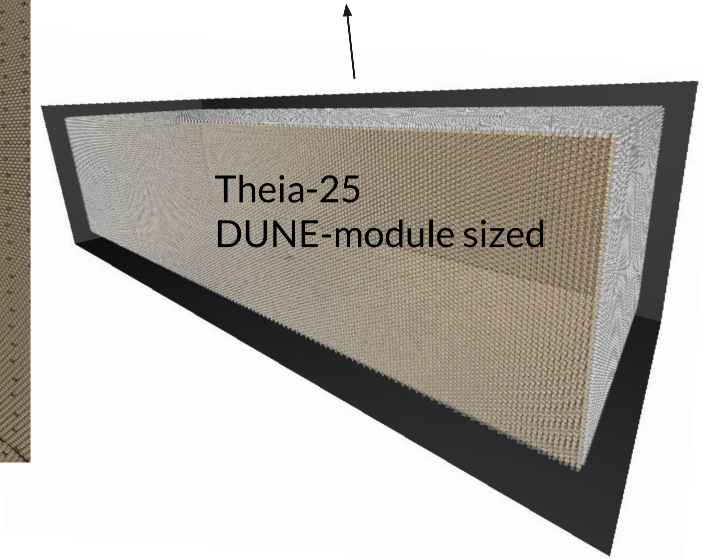

Detail view of detector model

Chroma Example: Theia

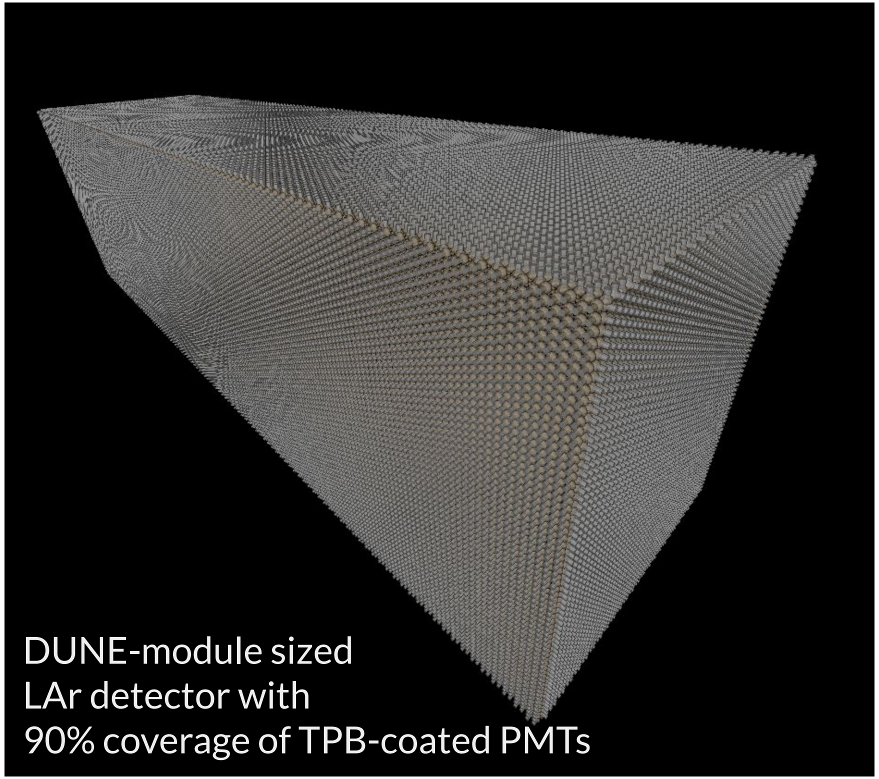


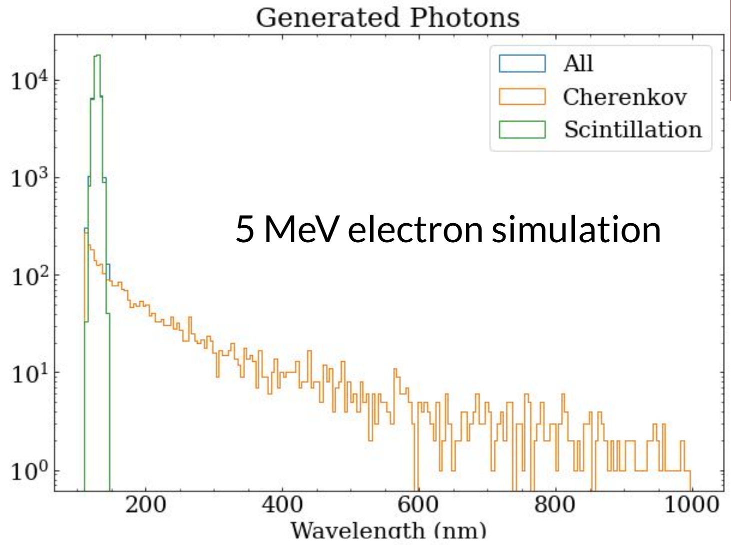


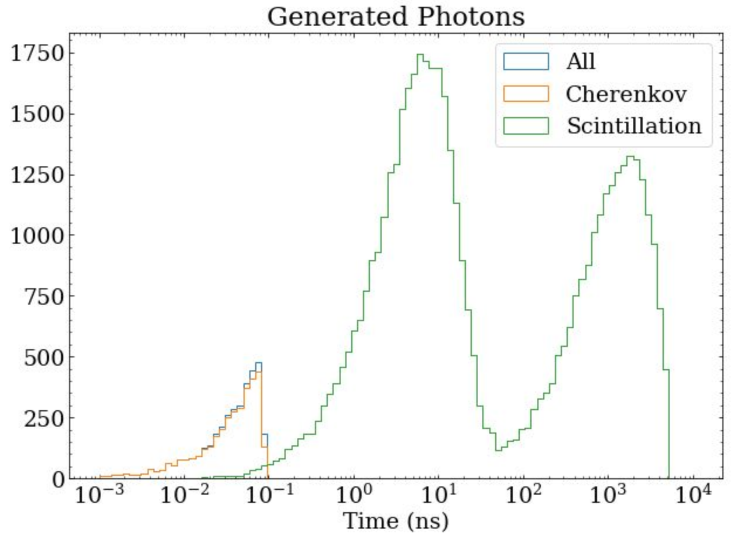




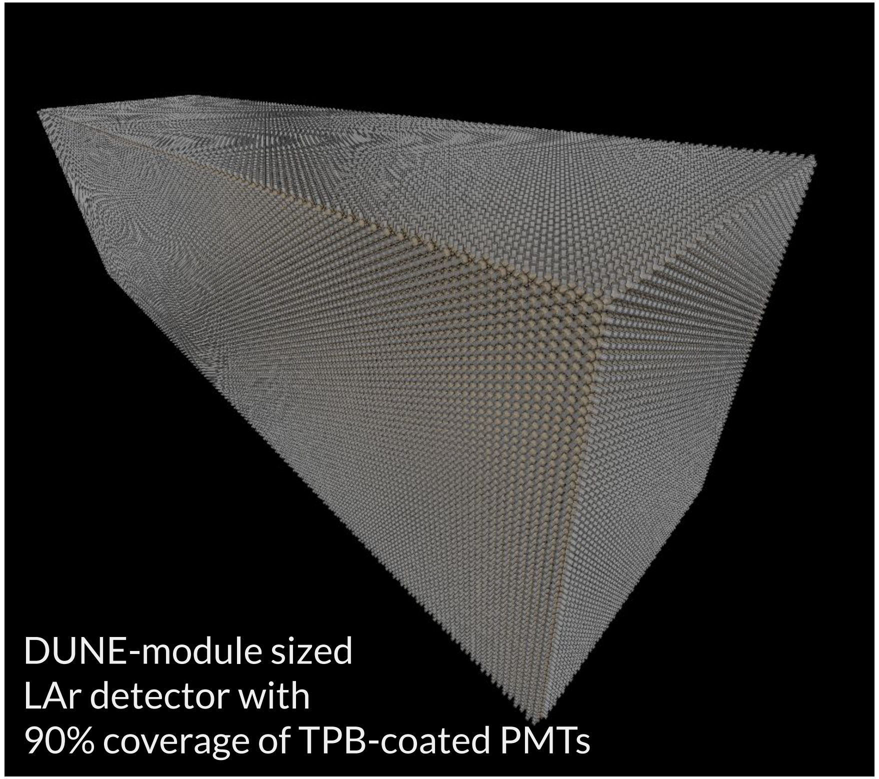
Chroma Example: Artemis


DUNE-module sized
LAr detector with
90% coverage of TPB-coated PMTs

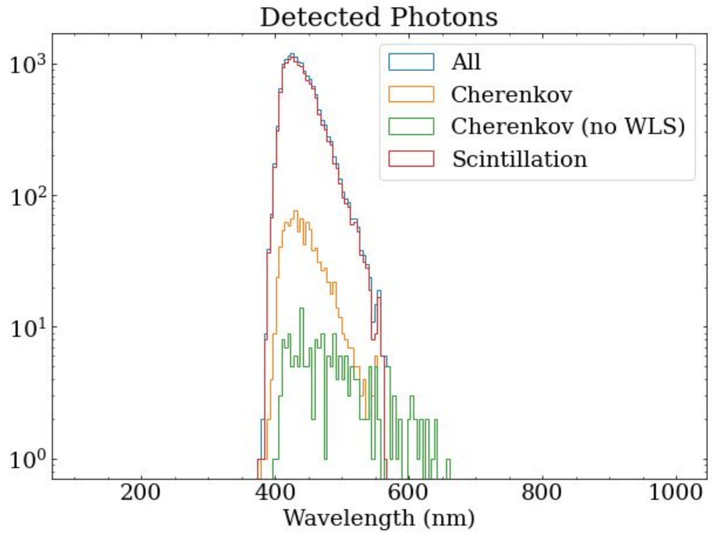


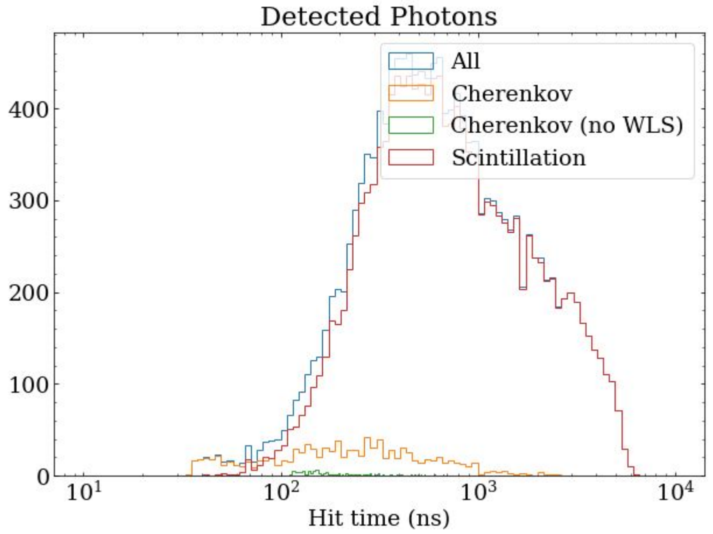


Chroma Example: Artemis



DUNE-module sized
LAr detector with
90% coverage of TPB-coated PMTs





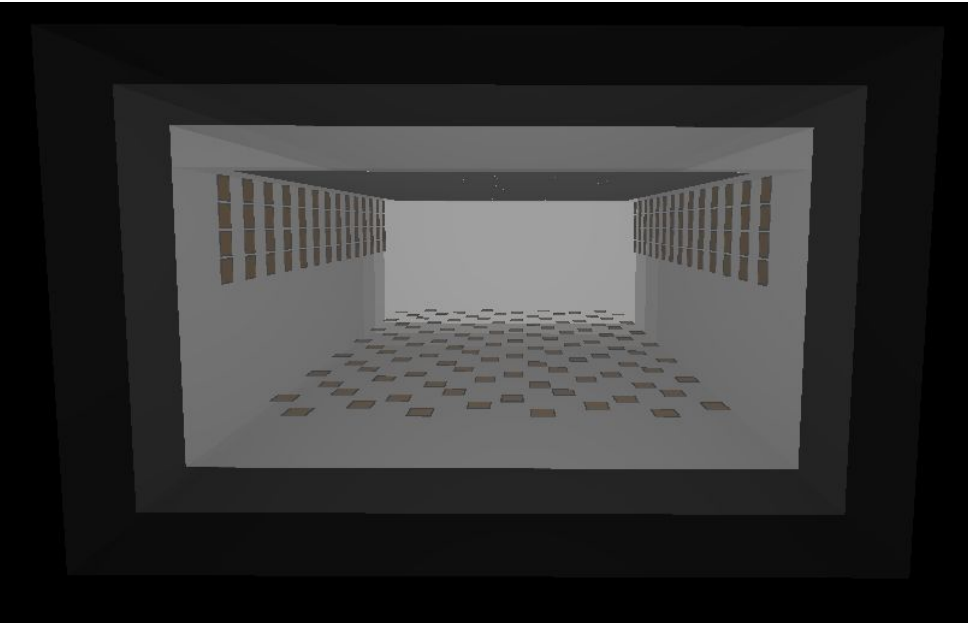
NEW: GDML Geometry import for Chroma

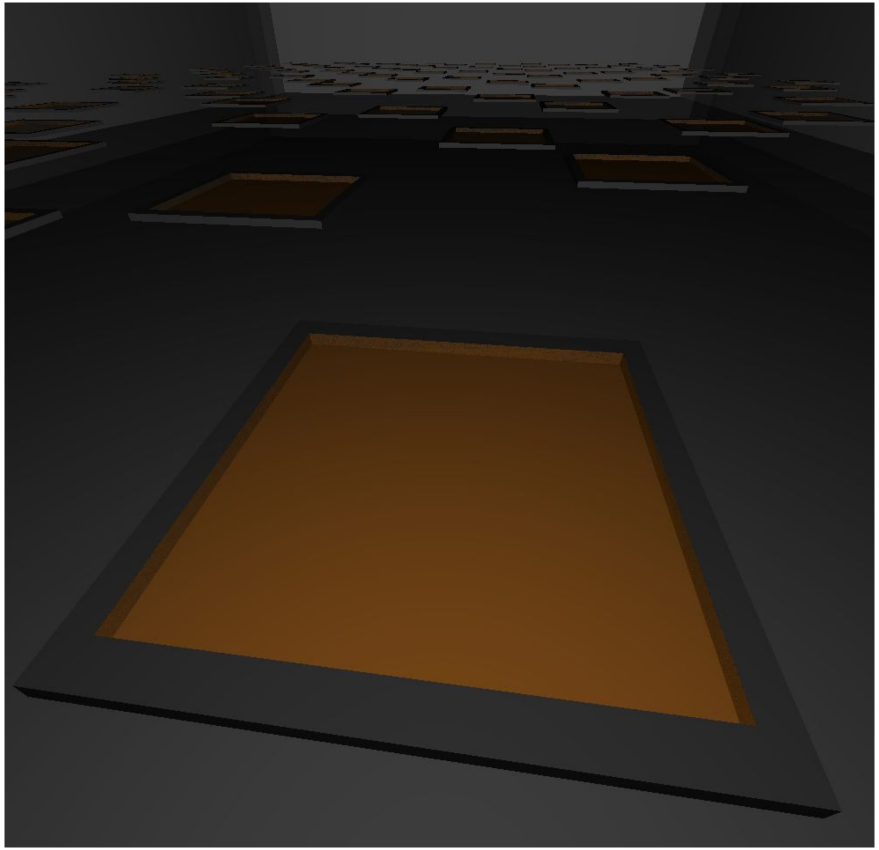
- Chroma can now parse GDML files and create matching geometries
  - For now, only supports features in GDML used by DUNE
  - Requires a python method to map volume names + material names to optical properties

```python
def dune_volume_classifier(volume_ref, material_ref, parent_material_ref):
    if parent_material_ref == material_ref:
        if 'OpDetSensitive' not in volume_ref:
            return 'omit',dict()
    if volume_ref == 'volWorld':
        return 'omit',dict()
    outer_material = material_map[material_ref][0]
    if 'OpDetSensitive' in volume_ref:
        channel_type = arapuca_to_id(volume_ref)
        assert volume_ref == id_to_arapuca(channel_type), \
            'Malformed identifier: '+volume_ref
        return 'pmt',dict(material1=custom_optics.vacuum,
                          material2=outer_material,
                          color=0xA0A05000,
                          surface=custom_optics.perfect_photocathode,
                          channel_type=channel_type)
    inner_material,surface,color = material_map[material_ref]
    return 'solid',dict(material1=inner_material,
                        material2=outer_material,
                        color=color,
                        surface=None)
```

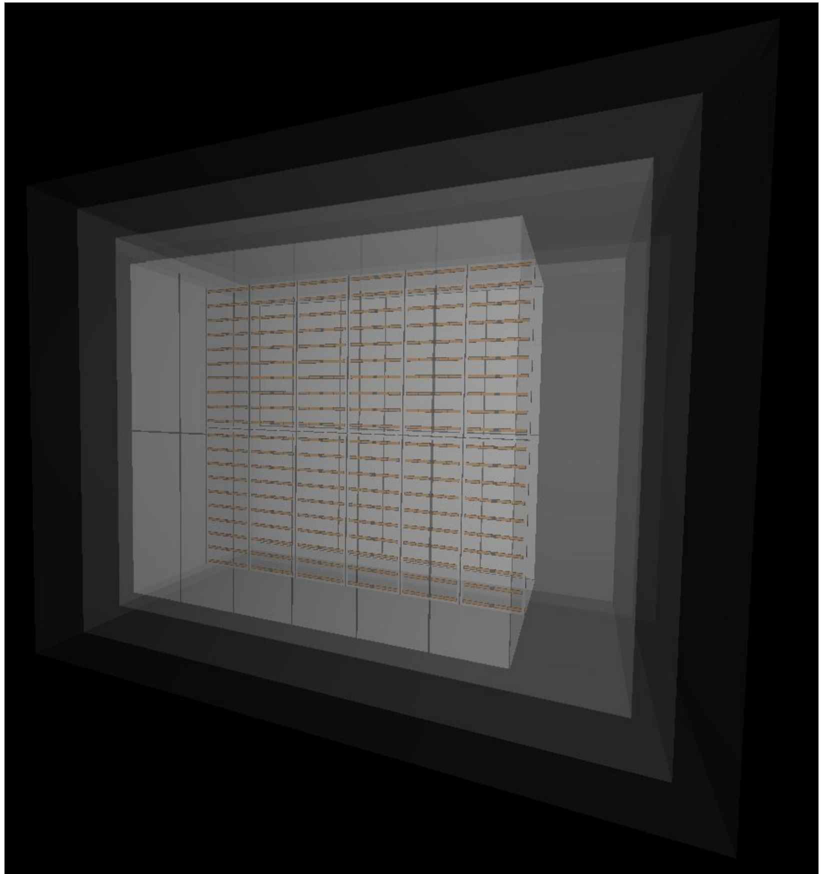
```python
from chroma.gdml import GDMLLoader
gdml = GDMLLoader('dunevd10kt_2view_v2_1x8x14pdsref.gdml')

from chroma.detector import Detector
det = Detector()
det = gdml.build_detector(
    detector=det,
    volume_classifier=dune_volume_classifier)

from chroma.loader import create_geometry_from_obj
from chroma.camera import Camera
geo = create_geometry_from_obj(det)
viewer = Camera(geo, size=(1000,1000))
viewer.start()
```

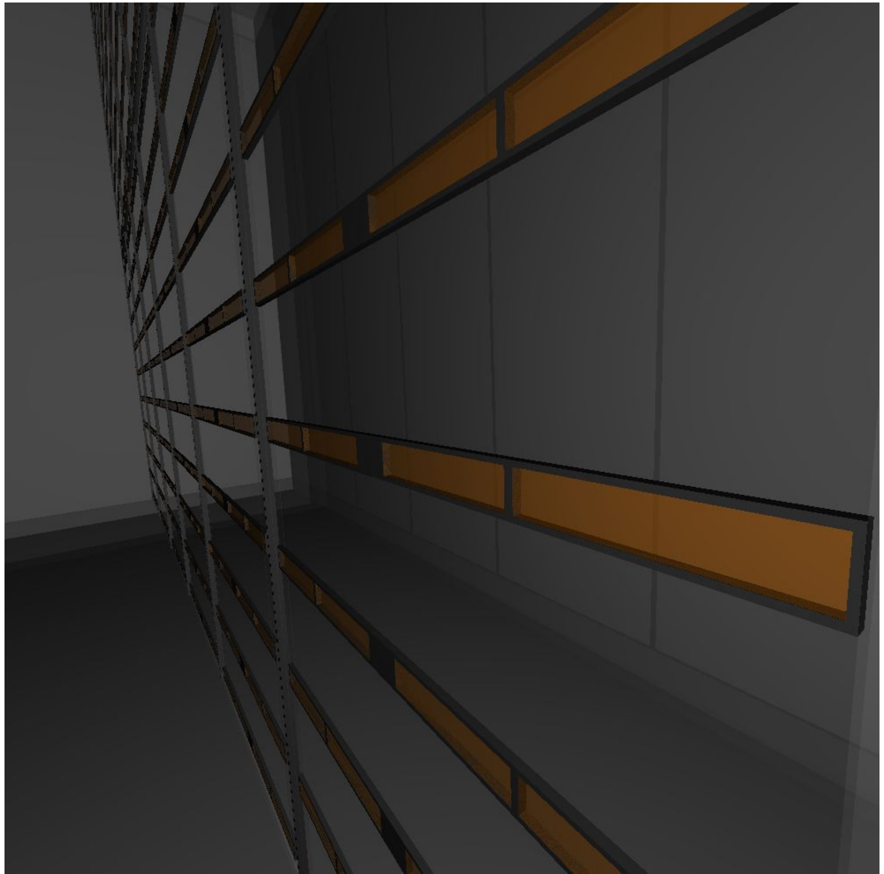
GDML Vertical Drift geometry



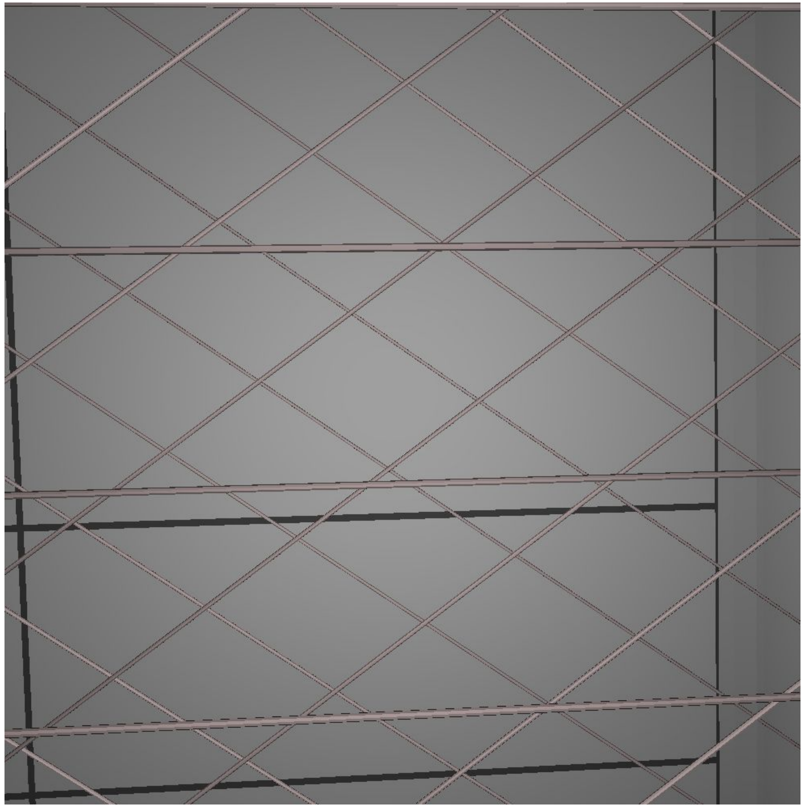


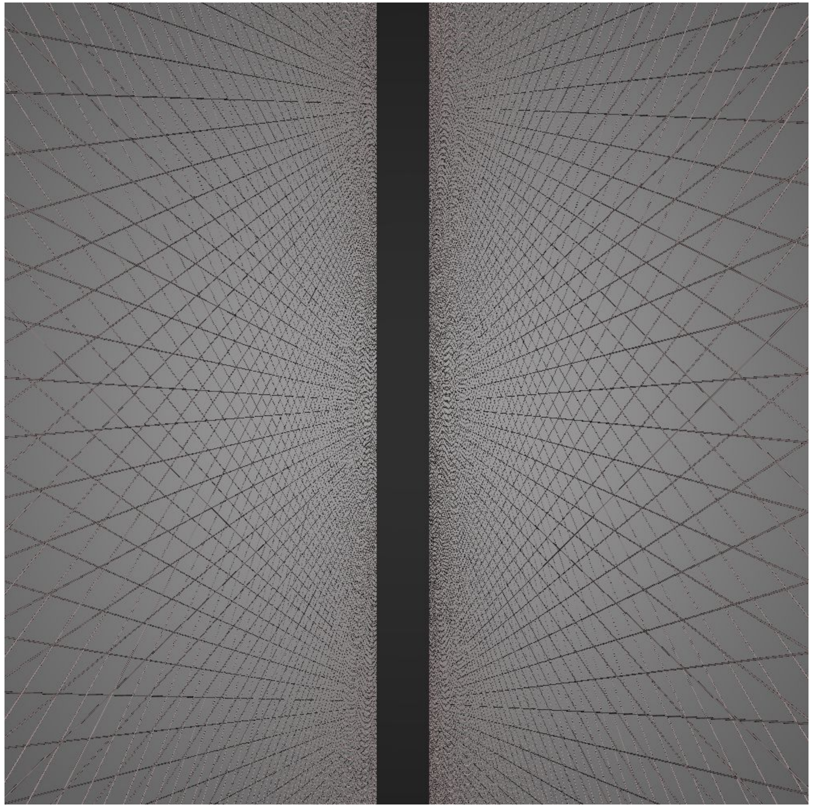
GDML Horizontal Drift geometry





GDML with wires works too! (but slower)





Runtime / GPU Requirements

- HDrift 5 MeV electrons
  - 1270 MB GPU RAM (no-wires)
  - 1340 MB GPU RAM (wires)
  - 53k generated photons
    - 2k detected photons
  - 40 ms / event (no-wires)
  - *2 seconds / event (wires)*
- VDrift 5 MeV electrons
  - 1220 MB GPU RAM (no-wires)
  - 1300 MB GPU RAM (wires)
  - 53k generated photons
    - 3k detected photons
  - 30 ms / event (no-wires)
  - 100 ms / event (wires)

- HDrift **5 GeV** electrons
  - Same geometry RAM requirements
  - 10m generated photons
    - *500MB of photon info (could be batched)*
    - 3.1m detected photons
  - 50 seconds / event (no-wires)
  - *40 minutes / event (wires)*
- VDrift **5 GeV** electrons
  - Same geometry RAM requirements
  - 10m generated photons
    - *500MB of photon info (could be batched)*
    - 340k detected photons
  - 40 seconds / event (no-wires)
  - 2 minutes / event (wires)

Benchmarking on GeForce GTX 1060
1280 CUDA cores / 6 GB of RAM
(in my laptop)

Next Steps: integration with DUNE simulations

- Option: add Chroma as photon propagator in LArSoft
  - c.f. existing fast photon simulation
  - Requires LArSoft to package and launch Chroma
  - Requires simulations to run on (or near) machine with GPU
    - Typically communicate with Chroma via ZMQ socket
- Option: use Chroma to process simulation outputs
  - Read generated photon information from .root files
  - A multi-stage simulation approach (simulate -> propagate -> analyze)
  - No requirement on larsim to know anything about Chroma
    - Chroma must be able to read/write files LArSoft understands
  - No requirement to run physics simulation on (or near) GPU machines