Chroma-Simulation Tutorial

nEXO Light Simulations Workshop

McGill University

October 21st, 2024

David Gallacher, PhD Candidate

Tutorial Part 1 - Outline

- What is Chroma
- What is Chroma-Simulation
- Chroma Simulation Components
  - Simulation.py, Detector.py, Event.py, Output.py, Photons.py
- Creating a Simulation Yaml card
- Creating an OpticalProperties Yaml card
- Demo 1 - Building your first Detector
  - Exporting Geometries into Chroma

What is Chroma?

- Chroma is a high performance optical photon simulation for particle physics detectors
  - Originally written by A. LaTorre and S. Seibert
- nEXO uses the Chroma Framework from https://github.com/BenLand100/chroma
  - Chroma has recent renewed support and is being developed under https://github.com/pennneutrinos/chroma
  - We will fast-forward to the stable development branch soon!
- Chroma + GPU libraries are kept in a singularity container
  - Works with NVIDIA GPUs
  - Compatible with HPC cluster GPUs
- Geometry is defined by STLs
  - No need for simplifying assumptions, export the whole detector CAD as STL files and configure optical properties in data tables

```
[==============================================================

  _____  _   _ ______ ________  ___  ___
 /  __ \| | | || ___ \  _  |  \/  | / _ \
 | /  \/| |_| || |_/ / | | | .  . |/ /_\ \
 | |    |  _  ||    /| | | | |\/| ||  _  |
 | \__/\| | | || |\ \\ \_/ / |  | || | | |
  \____/\_| |_/\_| \_|\___/\_|  |_/\_| |_/
         Light Simulation for nEXO

 ==============================================================
```

- GPU-based photon transport simulation
- Surface-based triangular mesh geometry
- Development in Python (core simulation in CUDA-C)
- Does:
  - Optical Photon Transport
  - Wavelength Shifting
  - Photon Detection

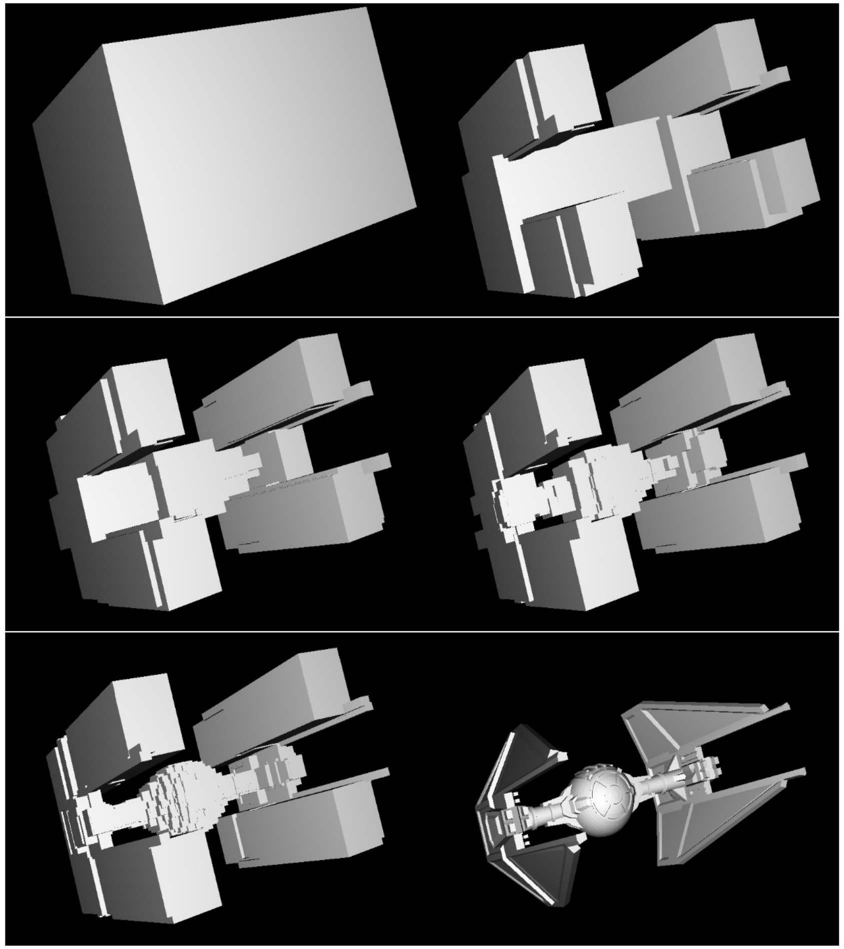
What is Chroma – Comparison with G4

GEANT4:

A detector is a tree of nested solids, each composed of some material and mathematically implemented by a particular C++ class.

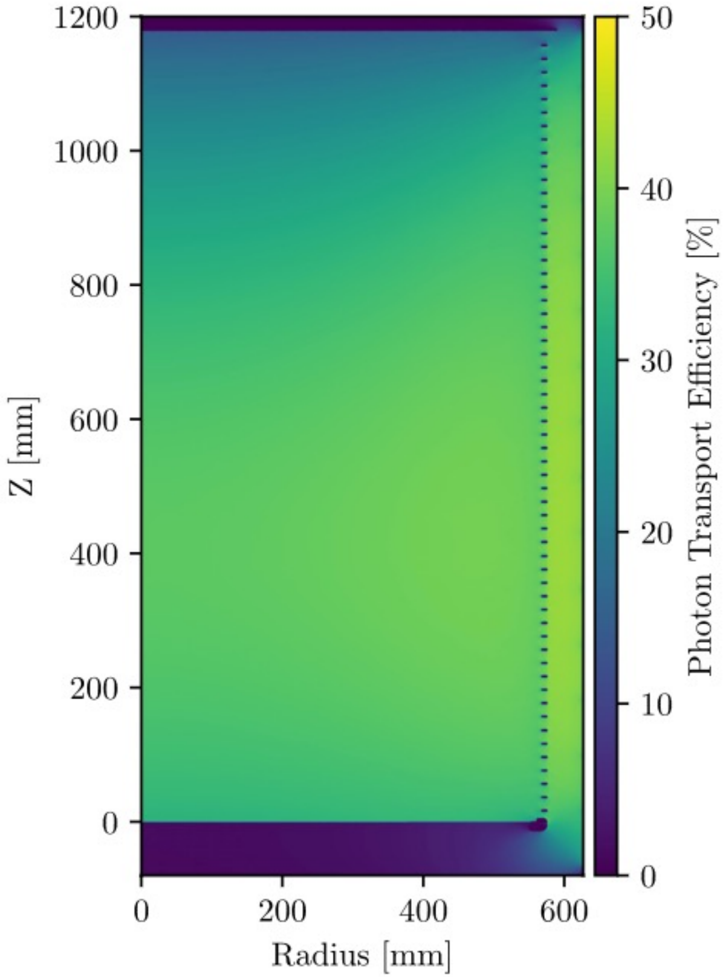
Chroma:

A detector is a list of oriented triangles, each representing the boundary between an "inside" and "outside" material.



What is Chroma-Simulation?

- Originally written by Ako Jamil, starting in 2019
- Used to generate light maps for 2021 nEXO 0vbb sensitivity study
- Python-based wrapper around Chroma source code, standardizes input geometries, output files, optical inputs and simulation control
- **Not currently embedded in nEXO-offline full GEANT4 simulation + reconstruction pipe-line**



How does Chroma-Simulation work?

- Singularity container contains Chroma source code + everything needed to run simulation
  - https://github.com/nEXO-collaboration/chroma-container
  - https://github.com/nEXO-collaboration/chroma
  - Automatically rebuilds containers when updated
  - **Requires NVIDIA GPU**
- Chroma-simulation main program – “RunSim.py”
  - https://github.com/nEXO-collaboration/chroma-simulation
- Define optical properties and simulation parameters in a subdir in Yaml/
- Loads geometries from Geometry/ (STL Files)
- Example:
  -
    ```
    singularity exec --nv ../Chroma.sif python RunSim.py -y Yaml/LoLX/LoLX.yaml
    ```

Run the singularity container
Chroma-simulation ‘main’ program
Specify what to run for the simulation

Chroma-Simulation Core Components

Simulation.py

Detector.py

Generator.py

Photons.py

Event.py & Output.py

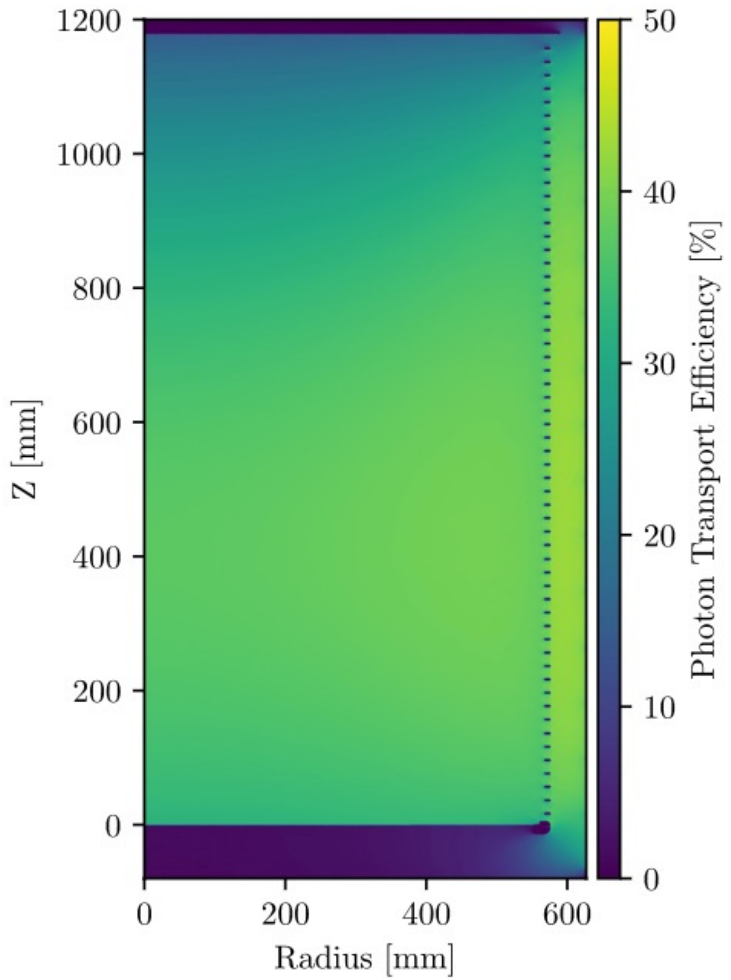
Chroma-Simulation Core Components

Simulation.py

Main simulation program, calls and initializes other classes for running simulation. Fairly lightweight.

Propagates photons to GPU using chroma commands

Delegates work to Detector.py, Generator.py, Events.py



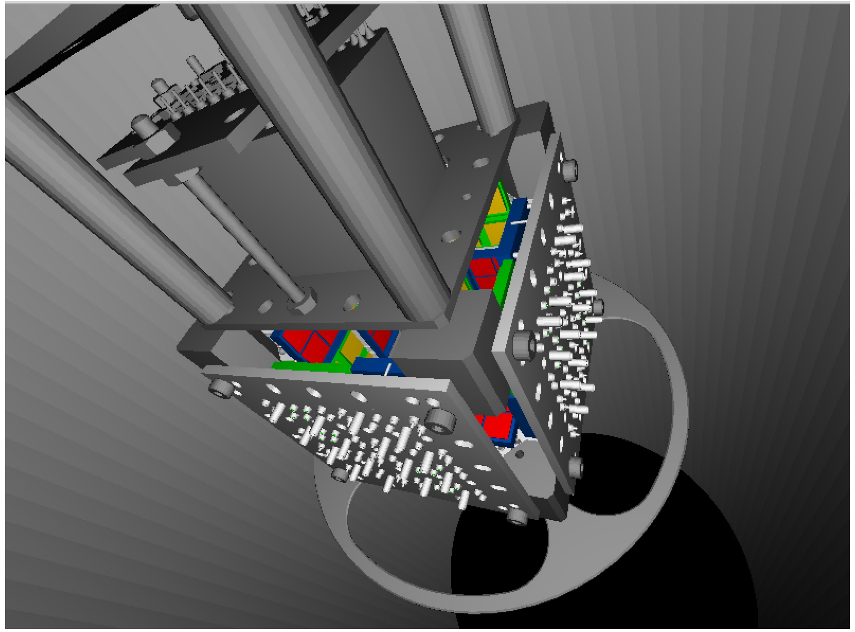
Chroma-Simulation Core Components

Detector.py

Simulation geometry builder and optical property handler

Reads in STL files using info from simulation yaml card, and applies optical properties to meshes

Optical properties can be applied as granularly as triangle-level (In and out separately), but generally applied to entire mesh



Example of detector visualizer (flag –v) for LoLX2 detector

Chroma-Simulation Core Components

Generator.py

Manages position generation for chroma-simulation

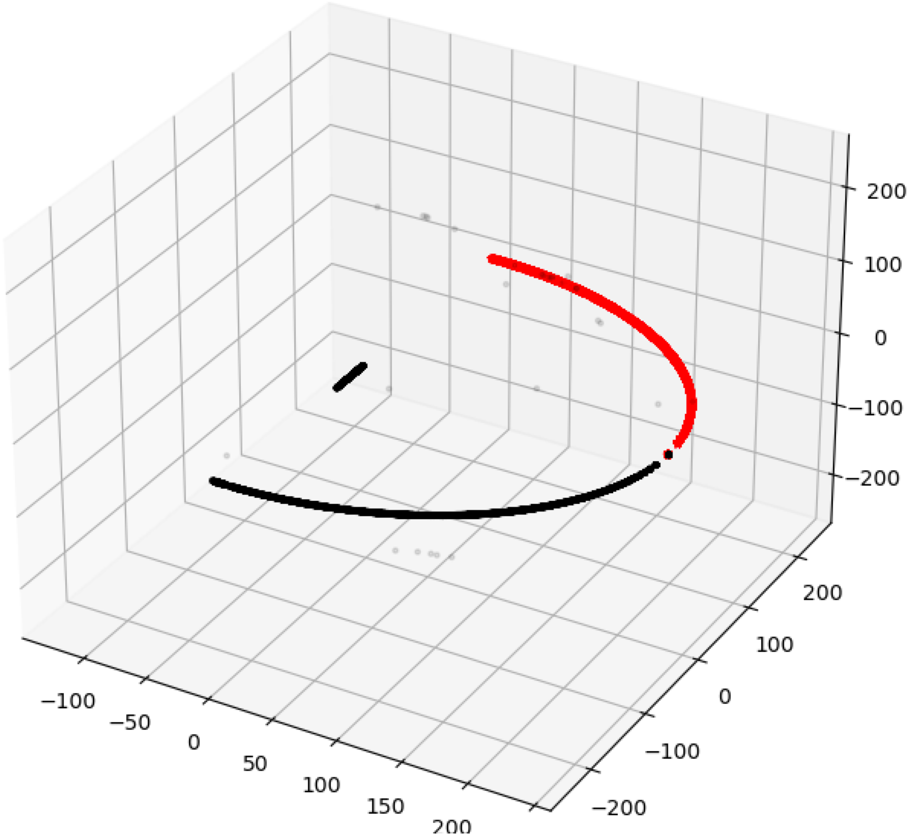
Choose position generator in Simulation table with

- `PhotonLocation: 'Point = 100.0,0.0,0.0’`

Lots of options for position generators, options for coincidence (multiple sources/point or multiple points per source)

Can configure additional yaml inputs if needed, outputs 3 arrays:

- Position (Nx3)
- Direction (Nx3)
- ExtraData (NxM)



Example of mounted laser generator, starting photon points (red) and end-locations (black) for test geometry

Chroma-Simulation Core Components

Photons.py

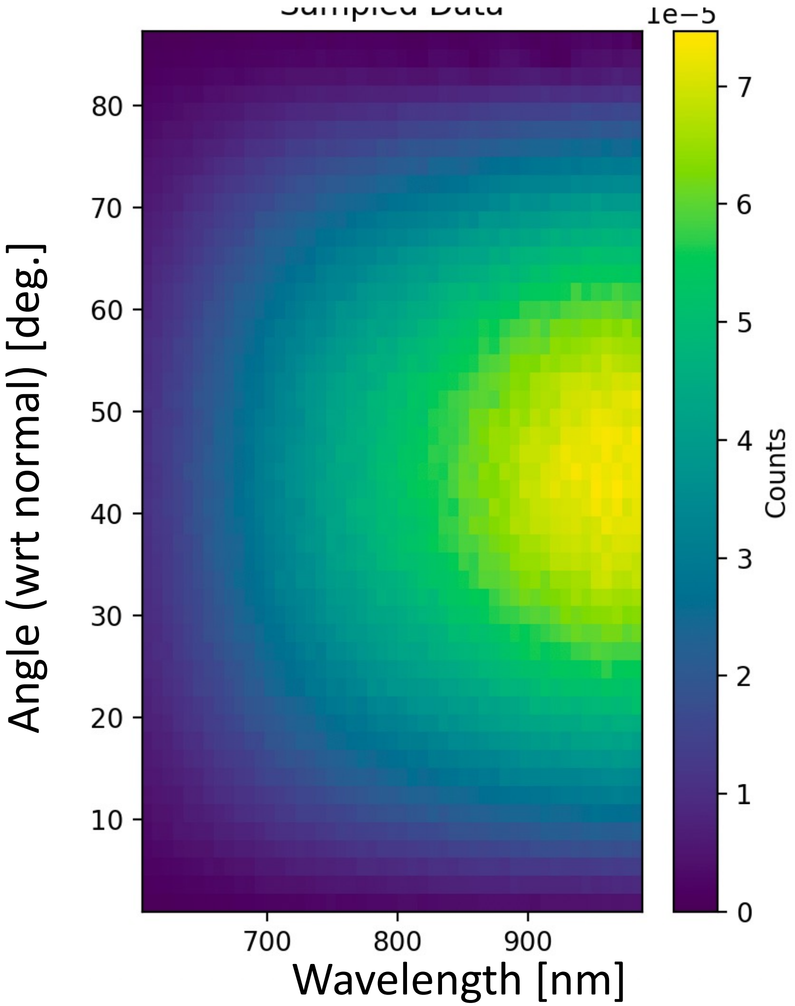
Defines simulated photon properties

- Wavelength, direction (from Generator, or custom), time off-set, polarization

Returns Chroma.photons objects to be sent to GPU for transport

Can define simple or complex photon sources, for example:

- “Photonbomb” - Isotropic randomly polarized emission from a point
- “Cherenkov” – Full pythonic Cherenkov emission simulation, takes in charged particle info from Generator::ExtraData and requires additional table in Simulation.py to define material properties



Example of Photon generator sampling (External Crosstalk)

Chroma-Simulation Core Components

Event.py & Output.py

Helper class definitions to clean up code and improve maintainability

3 Classes:

- H5Writer
  - Definitions of groups and structure of output H5 File, additional methods for writing meta-data
- EventReader
  - Reads Simulation.yaml and OpticalProperties.yaml and creates “Events”
    - Number of Events = NumberOfSources
  - Calls Generator.py, then passes results to Photons.py to create simulation events
- EventWriter
  - Takes Chroma output from GPU and formats it into output using H5Writer, uses Utilities.py to calculate additional information for output analysis

Other Parts

- Geometry/
  - Experiment dependent subdirs for STL files
- Data/
  - Info for simulations, more complex input data files
- Analysis/
  - Project specific analysis scripts and notebooks, look here for examples!
- Documentation/
  - Markdown documentation folder, we will be updating this together!
- ROOT/
  - Some utilities and examples for running ROOT input or using UpROOT
- Utilities/
  - Additional useful scripts for generating simulation inputs or looking at output

Simulation Yaml Cards

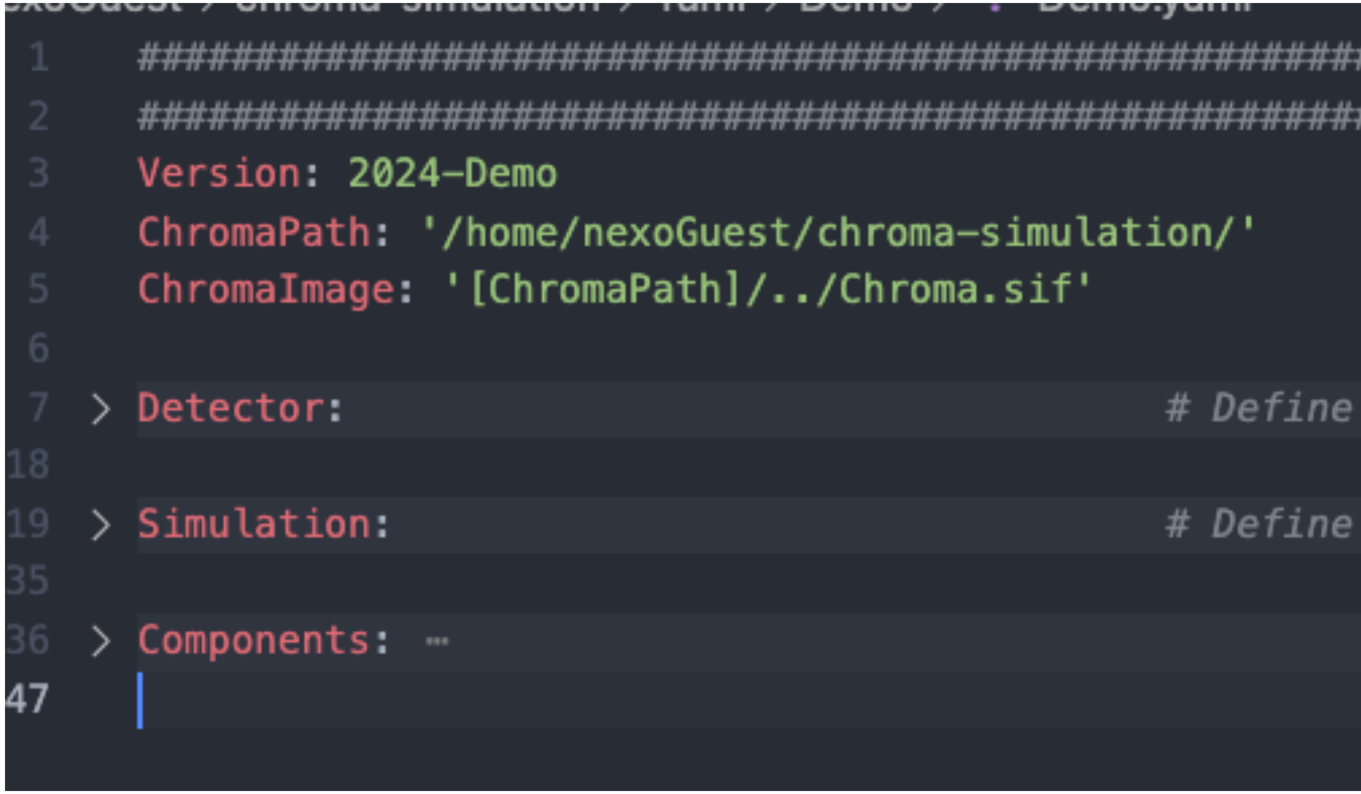
Contains 3 main tables

Detector

- Name of detector and path to STL files
- Options to cache detector for faster startup
- Define photon "detector" components and orientations
- Choose origin of simulation geometry

Simulation

Components

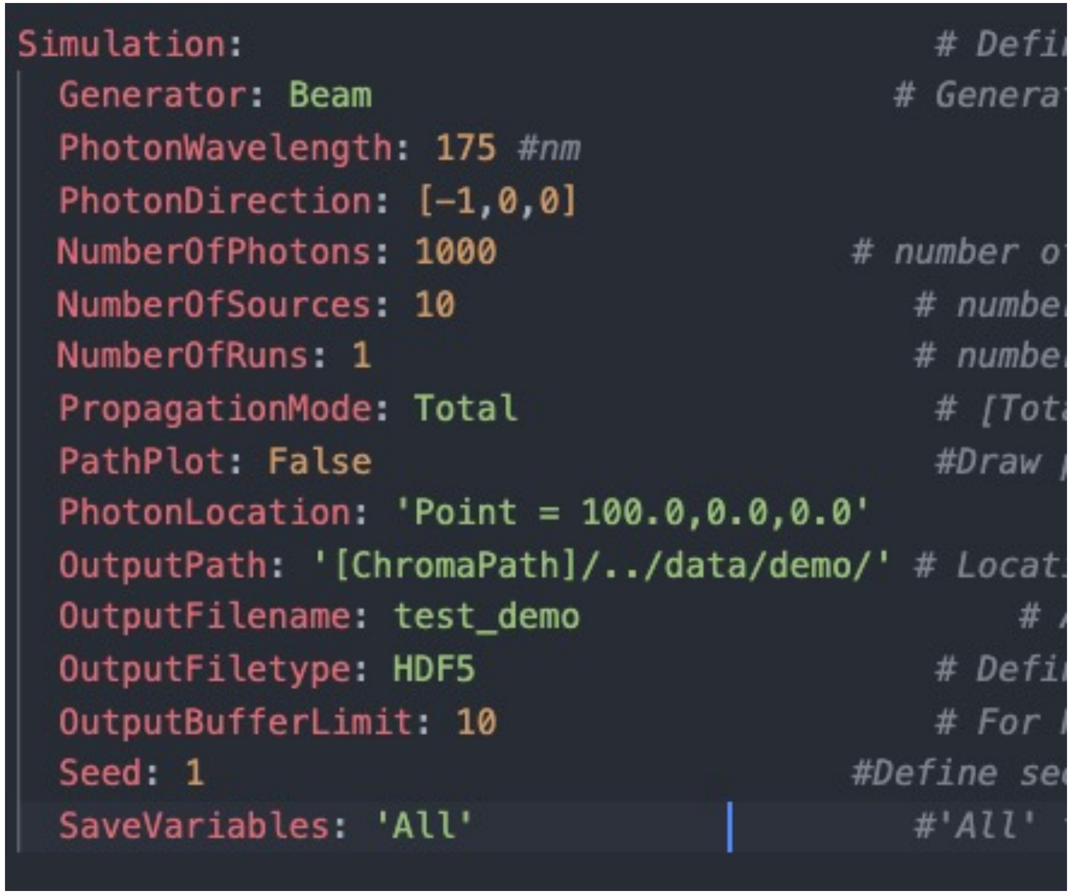


```
##################################################
##################################################
Version: 2024-Demo
ChromaPath: '/home/nexoGuest/chroma-simulation/'
ChromaImage: '[ChromaPath]/../Chroma.sif'

Detector:                                    # Define

Simulation:                                  # Define

Components: ⋯

```

Simulation Yaml Cards

Simulation

- Generator:
  - Photon generator name
- PhotonWavelength:
  - Optional override of photon wavelength (Default is 178 nm, or custom from Photons.py)
- NumberOfPhotons:
  - Photons per "event" overridden by many generators
- NumberOfSources:
  - Number of "events" for most generators where 1 position ≣ 1 event
- NumberOfRuns:
  - Repeats simulation 'N' times

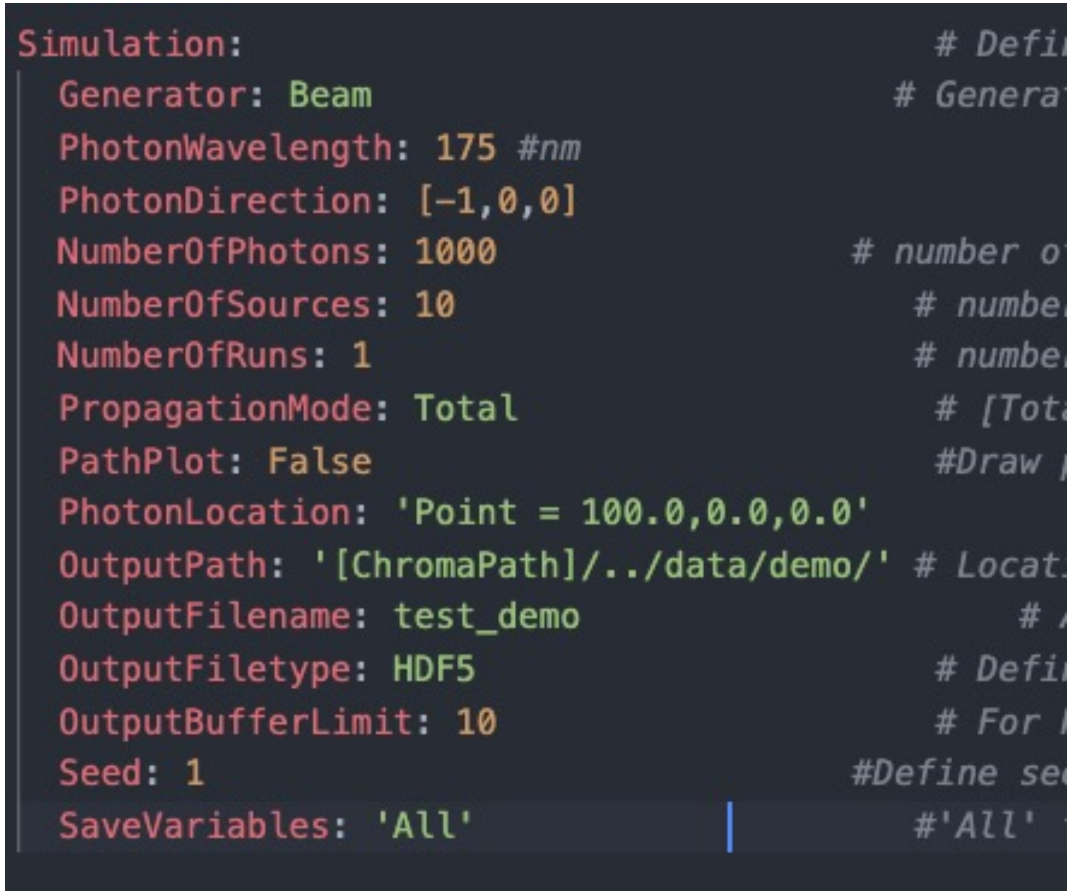


```yaml
Simulation:                                 # Defi
  Generator: Beam                         # Genera
  PhotonWavelength: 175 #nm
  PhotonDirection: [-1,0,0]
  NumberOfPhotons: 1000                 # number o
  NumberOfSources: 10                      # numbe
  NumberOfRuns: 1                          # numbe
  PropagationMode: Total                    # [Tota
  PathPlot: False                           #Draw
  PhotonLocation: 'Point = 100.0,0.0,0.0'
  OutputPath: '[ChromaPath]/../data/demo/' # Locat
  OutputFilename: test_demo                     #
  OutputFiletype: HDF5                      # Defi
  OutputBufferLimit: 10                     # For
  Seed: 1                               #Define se
  SaveVariables: 'All'                     #'All'
```

Simulation Yaml Cards

Simulation

- PropogationMode:
  - Step or total, most simulations use Total, use Step for debugging or detail checks
- PathPlot:
  - Option to draw photon path from Step propogation
- OutputPath/OutputFilename/OutputFiletype:
  - Output file specifications, will make directory if it doesn't exist
  - Only HD5F works reliably now
- Seed:
  - -1 for random seed, or specify for repeated sims
- SaveVariables:
  - "All" or "Most" options for full or almost-complete output
  - Option to just specify a list of variables you want for faster sim writing



```
Simulation:                                               # Defi
  Generator: Beam                                       # Genera
  PhotonWavelength: 175 #nm
  PhotonDirection: [-1,0,0]
  NumberOfPhotons: 1000                                 # number o
  NumberOfSources: 10                                     # numbe
  NumberOfRuns: 1                                         # numbe
  PropagationMode: Total                                   # [Tota
  PathPlot: False                                          #Draw
  PhotonLocation: 'Point = 100.0,0.0,0.0'
  OutputPath: '[ChromaPath]/../data/demo/' # Locat
  OutputFilename: test_demo                                    #
  OutputFiletype: HDF5                                     # Defi
  OutputBufferLimit: 10                                    # For
  Seed: 1                                              #Define se
  SaveVariables: 'All'                                     #'All'
```

Simulation Yaml Cards

Components

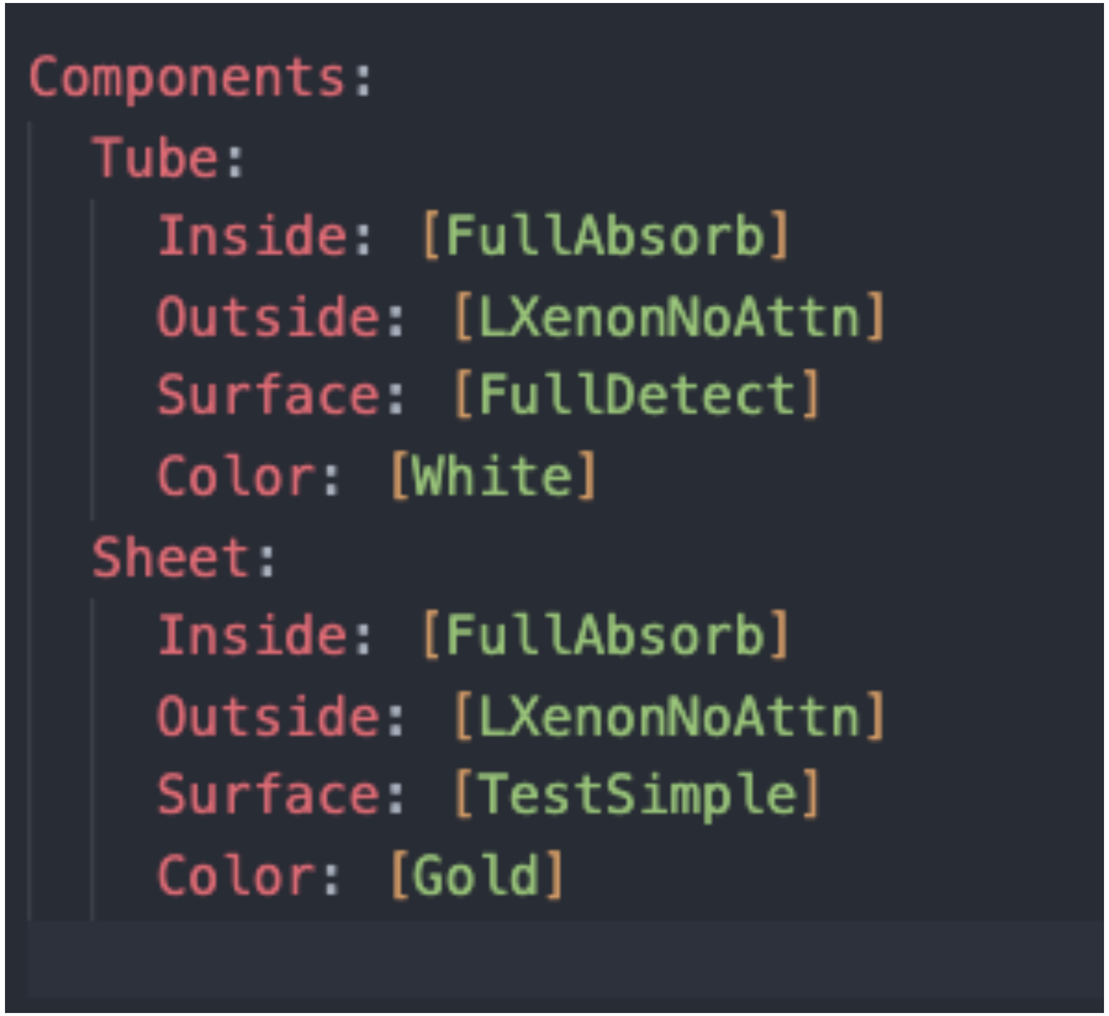
Map of STL file names to optical properties

- Key of OpticalProperties.yaml tables become entries in Components table
  - **Spelling and case sensitive!!**

May be lists or singular, colors defined in Detector.py

Detector orientation in Detector table defines list ordering (See Detector.py)

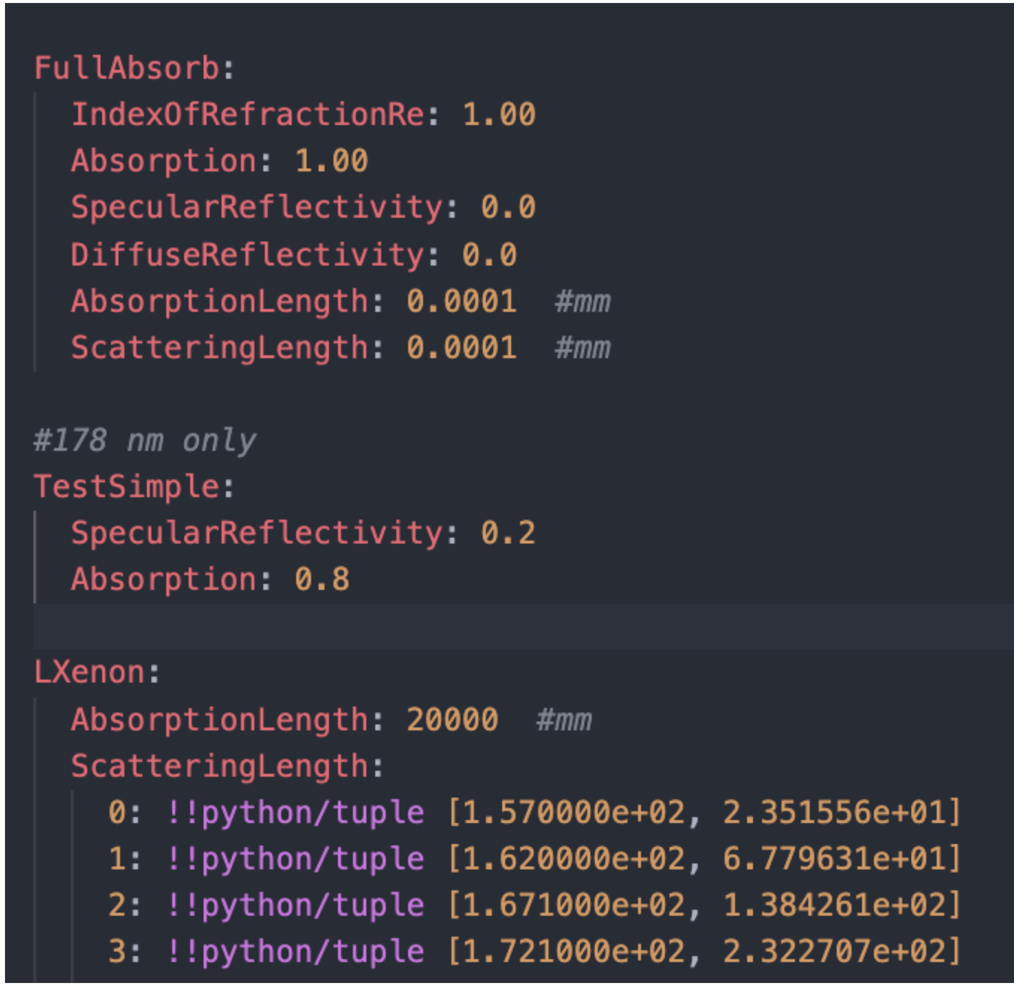
Name of component must be identical to a substring in desired STL Filename



```yaml
Components:
  Tube:
    Inside: [FullAbsorb]
    Outside: [LXenonNoAttn]
    Surface: [FullDetect]
    Color: [White]
  Sheet:
    Inside: [FullAbsorb]
    Outside: [LXenonNoAttn]
    Surface: [TestSimple]
    Color: [Gold]
```

Optical Properties Yaml Cards

- Two types of properties
  - Surface Properties
  - Bulk Properties
- Surface Properties:
  - Absorption, Transmission, DiffuseReflectivity, SpecularReflectivity, Reemission, IndexOfRefractionRe,IndexOfRefractionIm,Thicknes s, SurfaceModel
- Bulk Properties:
  - IndexOfRefractionRe, AbsorptionLength, ScatteringLength, Density, Composition
- Can be singular or WL dependent
- Helper scripts for converting CSV to Yaml for WL-dep in Utilities/
- **Wavelength in nm, lengths in mm**



```
FullAbsorb:
  IndexOfRefractionRe: 1.00
  Absorption: 1.00
  SpecularReflectivity: 0.0
  DiffuseReflectivity: 0.0
  AbsorptionLength: 0.0001  #mm
  ScatteringLength: 0.0001  #mm

#178 nm only
TestSimple:
  SpecularReflectivity: 0.2
  Absorption: 0.8


LXenon:
  AbsorptionLength: 20000  #mm
  ScatteringLength:
    0: !!python/tuple [1.570000e+02, 2.351556e+01]
    1: !!python/tuple [1.620000e+02, 6.779631e+01]
    2: !!python/tuple [1.671000e+02, 1.384261e+02]
    3: !!python/tuple [1.721000e+02, 2.322707e+02]
```

Building your own detector

Demo – 1 - CAD for Chroma-Simulation and Exporting Geometries

(David switches to Fusion-360)

Recap – Part 1

- Chroma – Source code written primarily in CUDA-C for ray-tracing photon transport on NVIDIA GPUs
- Chroma-Simulation – Python wrapper around Chroma for nEXO and other LXe setups to use Chroma easily
- Singularity Container – Virtualized OS with Chroma and all simulation software needs pre-installed and compiled
- Core code for Chroma-Sim is in Simulation/
- Parameters for simulation and optics are kept In Yaml cards (JSON but better) – Be careful about spelling always!
- Geometries exported from CAD need to share a single origin and coordinate system! (Very important)

Questions?

Break time

Tutorial - Part 2 Outline

- Simulation Inputs
- Adding a new position generator
- Adding a new photon generator
- Demo 2- Running your first Simulations
  - Detector visualizing
  - Photon Tracking
- Output File Components
- Analyzing Output Files
- Additional Info
- Demo 3 – nEXO Offline to Chroma (Time permitting)

Running Simulations

On workstation:

```
singularity exec --nv ../Chroma.sif python RunSim.py -y Yaml/LoLX/LoLX.yaml
```

- -y path to Yaml file for this simulation
- -v flag at the end for visualizer
- -d debug flag for prints

On clusters:

```
singularity exec --nv -B /project/def-tbrunner/software/ ../Chroma.sif python RunSim.py -y Yaml/LoLX/LoLX.yaml
```

- -B flag binds project directory into Singularity for file system navigation

Position Generators

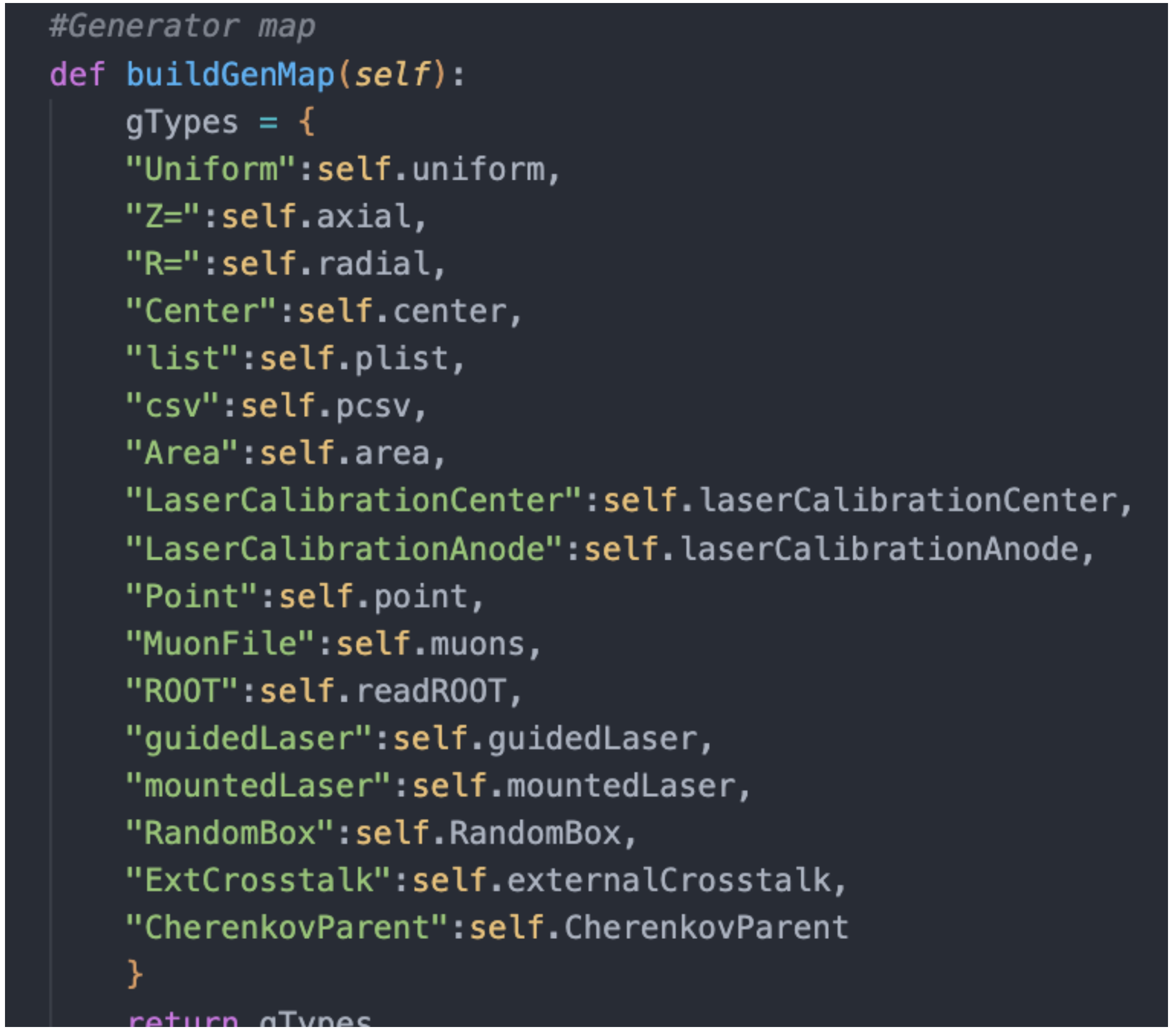
Lots of position generators to choose from (See Generator.py)

Some need extra inputs (Can be separate or in the string)

Examples:

1. PhotonLocation: 'Point = 100.0,0.0,0.0’
2. PhotonLocation: ‘ROOT’

Position generator input strings are sanitized in Event.py, additional Yaml tables are possible for complex generators



```
#Generator map
def buildGenMap(self):
    gTypes = {
    "Uniform":self.uniform,
    "Z=":self.axial,
    "R=":self.radial,
    "Center":self.center,
    "list":self.plist,
    "csv":self.pcsv,
    "Area":self.area,
    "LaserCalibrationCenter":self.laserCalibrationCenter,
    "LaserCalibrationAnode":self.laserCalibrationAnode,
    "Point":self.point,
    "MuonFile":self.muons,
    "ROOT":self.readROOT,
    "guidedLaser":self.guidedLaser,
    "mountedLaser":self.mountedLaser,
    "RandomBox":self.RandomBox,
    "ExtCrosstalk":self.externalCrosstalk,
    "CherenkovParent":self.CherenkovParent
    }
    return gTypes
```

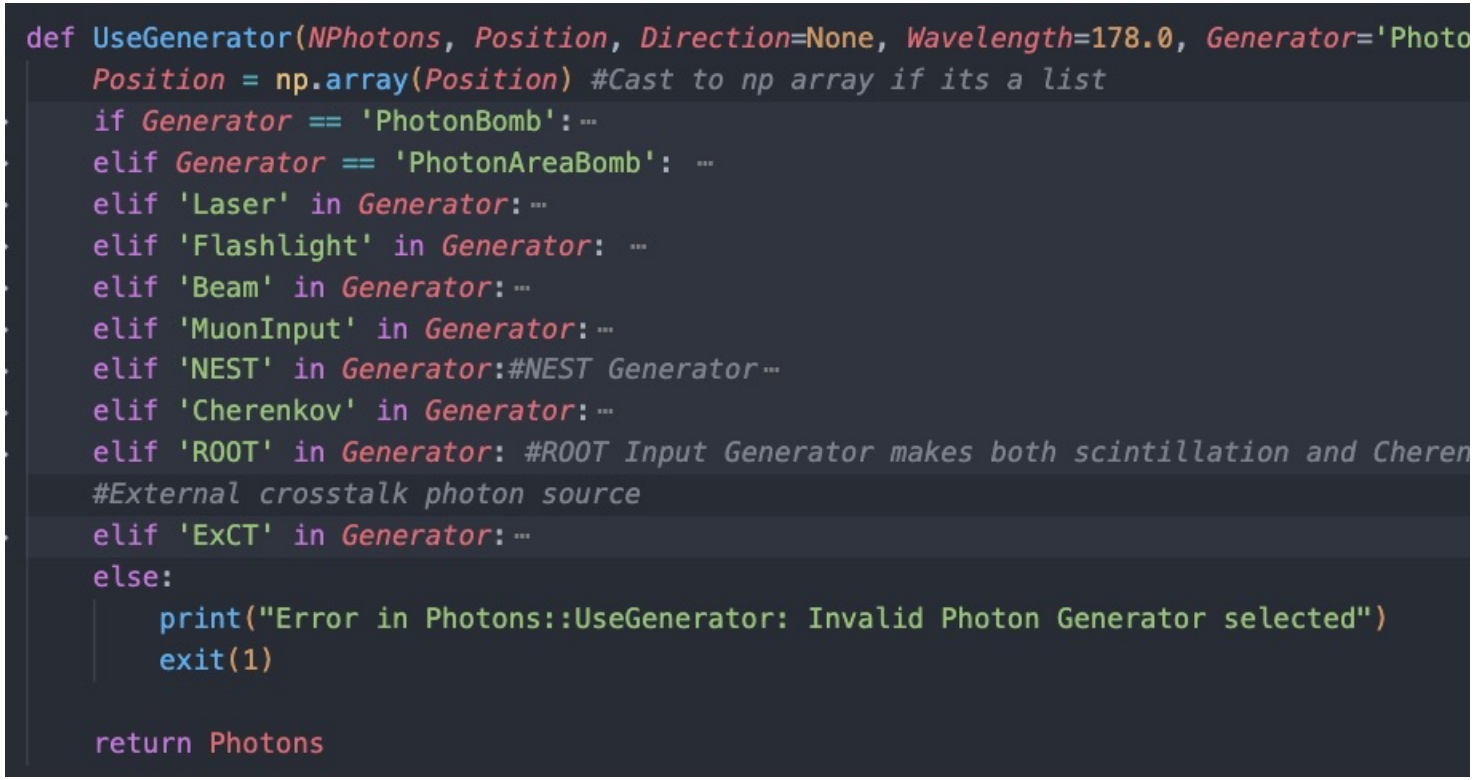
Photon Generators

Fewer Photon generators, Most of our applications are *PhotonBomb-like* or *Laser-like*

Custom/More complex generators added recently for direct scintillation/Cherenkov simulations and SiPM External Crosstalk

Defined in Simulation Table:

`Generator: ‘Beam’`



```
def UseGenerator(NPhotons, Position, Direction=None, Wavelength=178.0, Generator='Photo
    Position = np.array(Position) #Cast to np array if its a list
    if Generator == 'PhotonBomb':⋯
    elif Generator == 'PhotonAreaBomb': ⋯
    elif 'Laser' in Generator:⋯
    elif 'Flashlight' in Generator: ⋯
    elif 'Beam' in Generator:⋯
    elif 'MuonInput' in Generator:⋯
    elif 'NEST' in Generator:#NEST Generator⋯
    elif 'Cherenkov' in Generator:⋯
    elif 'ROOT' in Generator: #ROOT Input Generator makes both scintillation and Cheren
    #External crosstalk photon source
    elif 'ExCT' in Generator:⋯
    else:
        print("Error in Photons::UseGenerator: Invalid Photon Generator selected")
        exit(1)

    return Photons
```

Extending Position and Photon Generators

- For many applications, its cleaner and better practice to define custom generators
- When you should do this:
  - Other people will run your code
  - Other people will need to understand/refer to your simulation output
  - You need write a custom python script anyways to create a CSV of “Point” sources already
- When you shouldn’t do this:
  - Undefined parameters or algorithms
  - Just experimenting

Adding a new position generator

Needs:

- Algorithm for position generation, lots of examples of random sampling in Generator.py already for reference
  - All Meshes and ”Detector” info available in Generator class
- Input parameters in string (Need to sanitize input in Event.py)
  - Or Input parameters hardcoded (Bad)
  - Or Input parameters in custom table in Simulation Yaml (Good, but don’t hardcode it as required, check if it exists and throw a warning if not!)
- Add function to Generator.py class Generator
  - Add function name to dictionary of functions “buildGenMap”
    - Binds input Yaml PhotonLocation name to Generator class function name

Must output (N sources):

- Position (Nx3)
- Direction (Nx3) (Can be None if we don’t care but must be Position-shaped)
- ExtraData (NxM) (optional, can be None)

Adding a new photon generator

Needs

- Photons are made from position generators in Event.py::EventReader::CreateEvents

-
  ```
  def UseGenerator(NPhotons, Position, Direction=None, Wavelength=178.0, Generator='PhotonBomb', Diameter=1.0, ExtraData=None):
  ```
- Define function in Photons.py module file
- Add to chained if-elses in UseGenerator
- Can override WL, or direction, apply time-offsets and polarization
- UseGenerator is called once per source location
    - Can have multiple source locations per “event” through coincidence or ROOT input

Must output

-
  ```
  from chroma.event import Photons
  ```
- Chroma photons object, takes in np.arrays (one entry per photon)

-
  ```
  return Photons(Position, Direction, Polarization, Wavelengths)
  ```

Running your first simulation

Demo – 2 - Running Chroma Simulations

(David demonstrates on his laptop)

Output File Components

Default is Hdf5 file format, was testing UpROOT Ttree writing but very buggy currently for vector branches.

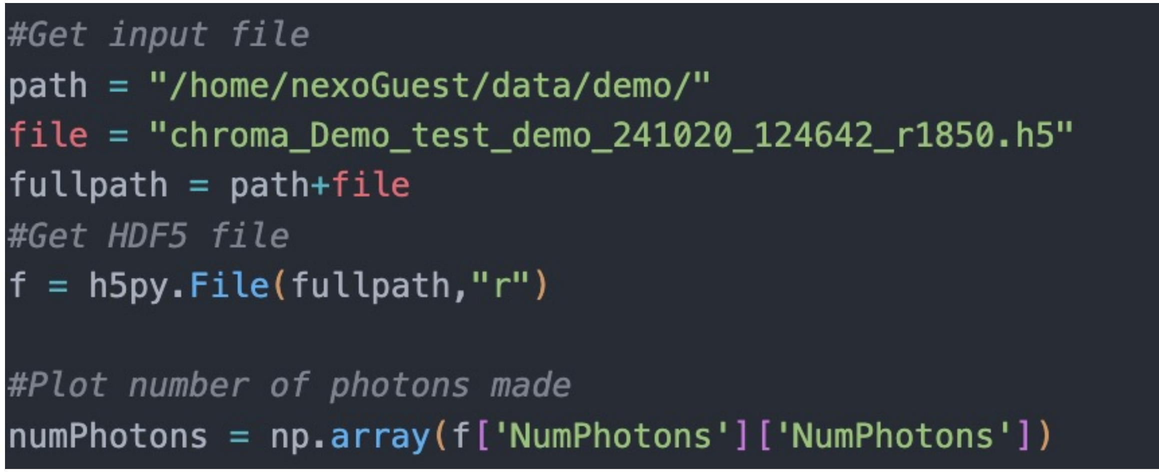
- Chroma container source code has pyROOT and has option to write ROOT files using pyROOT

Five main components to output file:

1. Metadata
    - One entry per file
    - Dumps Simulation Yaml used and other info, option to dump optical parameters to h5 file
2. Event Group
    - One entry per simulation event
    - NumHitChannels
    - NumDetected
    - NumPhotons
3. Photon Group
    - One entry per created photon
    - Flags (bit-map of history of photon interactions from Chroma)
    - PhotonWavelength (nm)
    - StartPosition (x,y,z [mm])
    - EndPosition (x,y,z [mm])
    - LastHitTriangle (name)
    - PhotonTime (ns)
4. Channel Group
    - One entry per event, array-like
    - ChannelIDs (Map of Channel index to ID)
    - ChannelTimes (Array of hit time of last photon on channel ’n’)
    - ChannelCharges (Array of total number of detected photons on channel ‘n’)
5. Detected Photon Group
    - One entry per detected photon
    - DetectedPos (EndPosition for this detected photon)
    - DetectorHit (x,y,z center position of hit detector)
    - DetectorHitID (ID of hit detector)
    - IncidentAngles (Angle of incidence when detected)

Analyzing Output Files

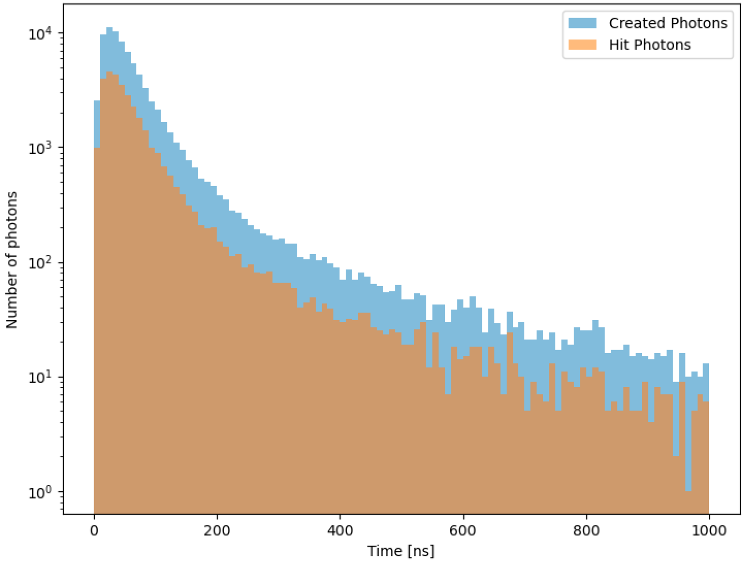
- Look in Utilities/ and Analysis/ for inspiration, don't be afraid to ask on #chroma channel on nEXO slack!
- Property and Group is duplicated currently
  - See NumPhotons on right for reference
- Use Utilities/printSimulationOutput.py to get a feel!
- Once you've identified your analysis needs, trim the SaveVariables input specification to speed up simulation time!

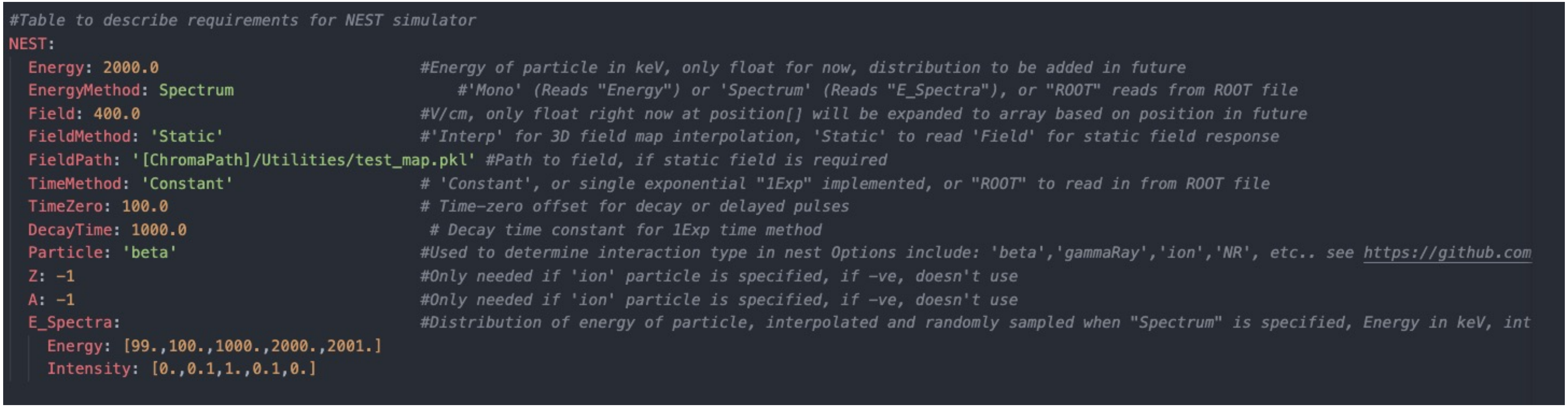


```
#Get input file
path = "/home/nexoGuest/data/demo/"
file = "chroma_Demo_test_demo_241020_124642_r1850.h5"
fullpath = path+file
#Get HDF5 file
f = h5py.File(fullpath,"r")

#Plot number of photons made
numPhotons = np.array(f['NumPhotons']['NumPhotons'])
```

Additional Info – Standalone NEST

- Incorporate NEST into chroma-simulation using NESTpy bindings (NEST 2.0) - https://pypi.org/project/nestpy/
- This uses NEST models to produce the mean number of photons for a given particle type, energy, and E field
- Can define static field or field map (3d look-up-table)

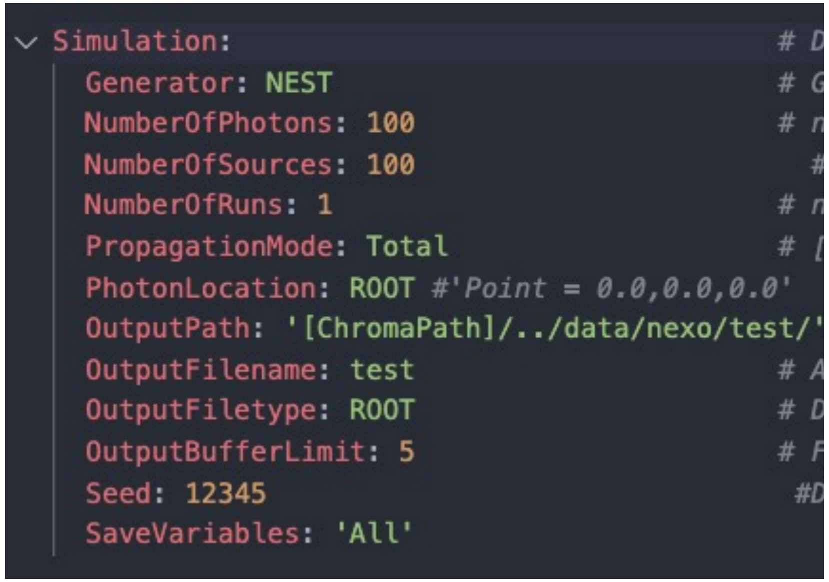




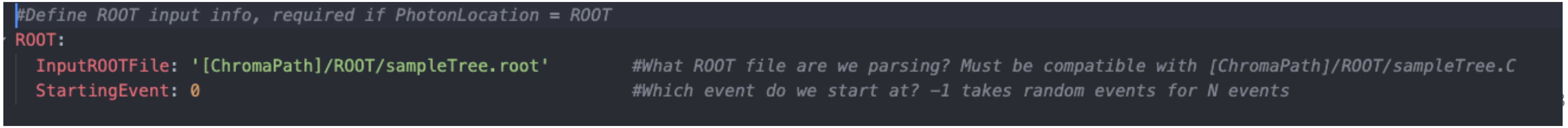
```
#Table to describe requirements for NEST simulator
NEST:
  Energy: 2000.0                            #Energy of particle in keV, only float for now, distribution to be added in future
  EnergyMethod: Spectrum                        #'Mono' (Reads "Energy") or 'Spectrum' (Reads "E_Spectra"), or "ROOT" reads from ROOT file
  Field: 400.0                              #V/cm, only float right now at position[] will be expanded to array based on position in future
  FieldMethod: 'Static'                     #'Interp' for 3D field map interpolation, 'Static' to read 'Field' for static field response
  FieldPath: '[ChromaPath]/Utilities/test_map.pkl' #Path to field, if static field is required
  TimeMethod: 'Constant'                    # 'Constant', or single exponential "1Exp" implemented, or "ROOT" to read in from ROOT file
  TimeZero: 100.0                           # Time-zero offset for decay or delayed pulses
  DecayTime: 1000.0                          # Decay time constant for 1Exp time method
  Particle: 'beta'                          #Used to determine interaction type in nest Options include: 'beta','gammaRay','ion','NR', etc.. see https://github.com
  Z: -1                                     #Only needed if 'ion' particle is specified, if -ve, doesn't use
  A: -1                                     #Only needed if 'ion' particle is specified, if -ve, doesn't use
  E_Spectra:                                #Distribution of energy of particle, interpolated and randomly sampled when "Spectrum" is specified, Energy in keV, int
    Energy: [99.,100.,1000.,2000.,2001.]
    Intensity: [0.,0.1,1.,0.1,0.]
```

Additional Info – GEANT4 -> ROOT + NEST

- To integrate with G4 (as a 2-stage simulation), we can output some info from the G4 simulation to a ROOT File to be read into Chroma-simulation
- The TTree format and an example of TTree building required is in :
  - [ChromaPath]/ROOT/sampleTree.C
- We need a few things:
  - Energy deposit number for given 'event' (ndep)
  - Event ID (For counting ndeps per event to simulate)
  - Energy deposited at each deposit (eDep)
  - Position of each deposit (X,Y,Z)
  - Particle ID for each deposit (PDG Code)
  - Time of each deposit (tDep)
  - Momentum direction (x,y,z vectors)
  - Energy of the particle when depositing the energy at this step
- This can be stored in a simple TTree which is read into a new position generator "ROOT"
  - [Simulation][PhotonLocation] = ROOT, Read into a pandas DF using UpROOT



```
Simulation:                                              # D
  Generator: NEST                                        # G
  NumberOfPhotons: 100                                   # n
  NumberOfSources: 100                                     #
  NumberOfRuns: 1                                        # n
  PropagationMode: Total                                 # [
  PhotonLocation: ROOT #'Point = 0.0,0.0,0.0'
  OutputPath: '[ChromaPath]/../data/nexo/test/'
  OutputFilename: test                                   # A
  OutputFiletype: ROOT                                   # D
  OutputBufferLimit: 5                                   # F
  Seed: 12345                                             #D
  SaveVariables: 'All'
```



```
#Define ROOT input info, required if PhotonLocation = ROOT
ROOT:
  InputROOTFile: '[ChromaPath]/ROOT/sampleTree.root'       #What ROOT file are we parsing? Must be compatible with [ChromaPath]/ROOT/sampleTree.C
  StartingEvent: 0                                         #Which event do we start at? -1 takes random events for N events
```

Additional Info – Coincidence Generation

In order for ROOT input to work, I needed to abstract the concept of “coincidence”

Included options for “manual” coincidence

Define this “Coincidence” table in your Simulation table of Simulation yaml.

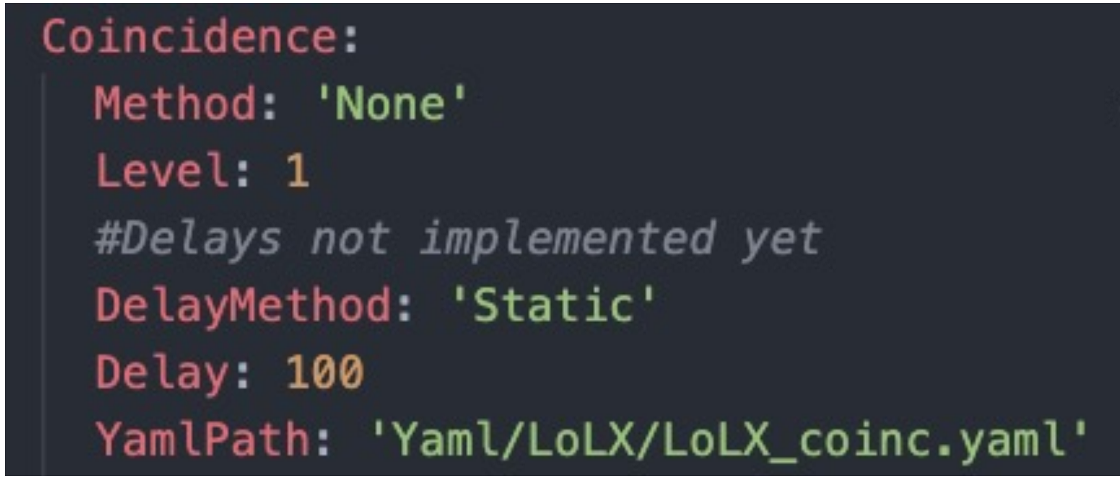
Level = Number of coincidences per source

Methods:

- 'SamePos’, different photon type, same position as this primary source
- 'PileUp’, different positions and same photon gen. type
- 'PileUpDiff’for different positions and different types
- 'None' for no coincidence (or don’t define this table)

DelayMethod: Option to include a time delay between coincidence events

YamlPath: Path to yaml file that contains the Simulation table for the coincidence generation (uses whats needed for a given Method)



```
Coincidence:
  Method: 'None'
  Level: 1
  #Delays not implemented yet
  DelayMethod: 'Static'
  Delay: 100
  YamlPath: 'Yaml/LoLX/LoLX_coinc.yaml'
```

Recap – Part 2

- First debug check should always be for spelling of names
  - Check Components table keys to STL names
  - Check Components table entries to OpticalProperties table keys
  - Check OpticalProperties table entry spelling against search terms in Detector.py
- Easy to extend Chroma-simulation generators, code that does nEXO analysis should be part of Chroma-simulation for review and reproducibility
- Can produce scintillation light directly in Chroma-simulation using NEST model, more complex generators are possible

nEXO-Offline Into Chroma

Demo - 3 – nEXO Offline Simulation output into Chroma

Time-permitting

Thank you!

Questions?

Guest Access to Chroma Workstation

Host: 132.206.126.37

User: nexoGuest

Pass: nEXO2024

Only a few people can run on the GPU simultaneously, be considerate!

Feel free to make more directories and clone chroma-simulation for your needs