A Study of Photon Propagation Acceleration

Keith Evans[1], James Michael Patrick Brady [1], Olivia Alexandra Borgström[2], Sacha Barré[1], Yangyang Cui[2], Adam Davis[1], Joeseph Rodríguez Fernádez[1], Marco Gersabeck[1], Andrei Mihnea Ghira[2], Zahra Montazeri[2], Antonin Rat[1]

[1]Department of Physics, University of Manchester

[2]Department of Computer Science, University of Manchester

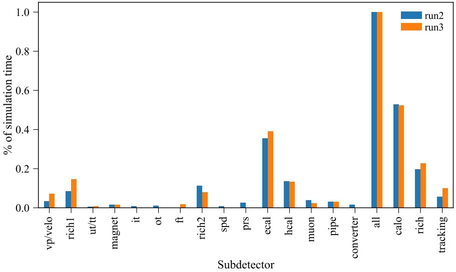
November 21, 2023

Contents

Plan

Background and Motivation

- HEP Simulations are **crucial** due to the size, complexity and sensitivity of detectors
- In LHCb **RICH** simulations represent a large fraction of computational expense
  - Issue: Propagation of Cherenkov Photons
  - This will be a problem for any simulation relying on optical photon propagation
  - General problem inside and outside HEP



Parallelisation and Acceleration

- Particle propagation is a labour intensive problem due to stepping
  - Faster particle = smaller step
- Lends itself well to parallelisation
  - Every particle is independent of all others
- CPU parallelisation relatively easy
  - Imbalanced loads less important
  - Limited potential for speed up
- GPU Parallelisation more difficult
  - Load balancing extremely important
  - Speed up potential very high
    - TFLOPs vs GFLOPs
- Need to consider hetrogeneous hardware
  - Don't want all eggs in one basket

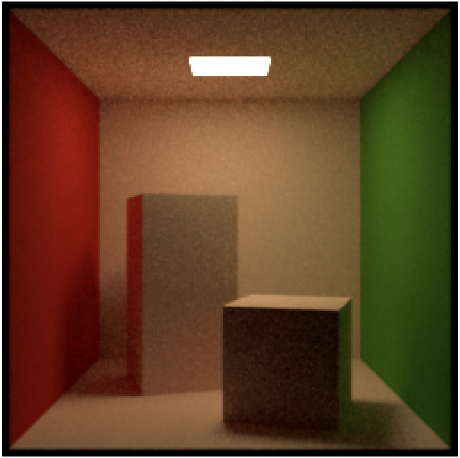
Load Balancing - Binary tree problem

- CPU Case
  - Core 1 propagates particle 1, Core 2 propagates particle 2
  - Core 1 finishes first and moves on to particle 3...n
- GPU Case
  - Thread 1 propagates particle 1, Thread 2 propagates particle 2
  - Thread 1 finishes first and waits for thread 2

Plan

What is Mitsuba3

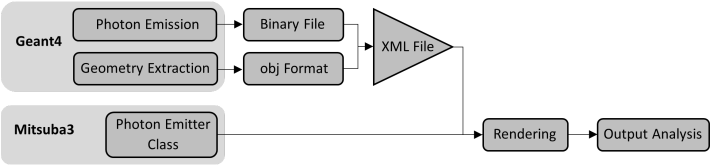
- Physics based rendered
  - Represents light as rays
- Uses MC sampling to probabilistically determine pixel values
- Industrially recognised and widely used
- Open source
  - Under active development
- Accelerated using LLVM and CUDA
  - CUDA uses OptiX
  - LLVM uses Embree
    - Potential for AMD GPUs



- Input
  - XML
  - Python
- Rendering
  - Parses geometry (scene)
  - Compiles optimised kernel (DRJIT)
    - CUDA + OptiX
    - LLVM + Embree
  - Emits and propagates (traces) photons (rays)
  - Constructs images applying camera effects
- Output
  - EXR
  - Numpy

Plan

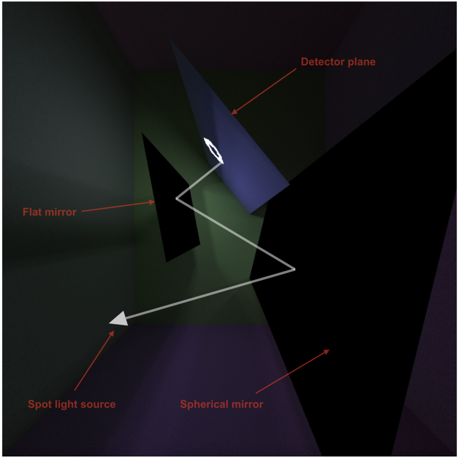
The Pipeline



- Cherenkov emission by Geant4 translated into BIN file
- RICH simplified geometry exported in OBJ file
- XML reads BIN and OBJ file
- New photon emitter class used to emit photons in the scene

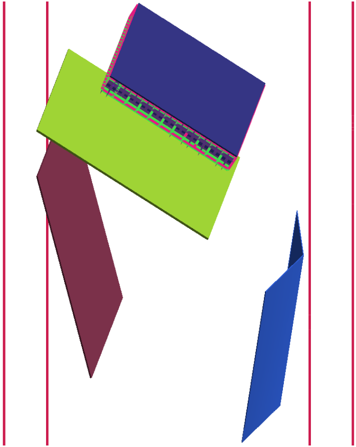
Detector Construction

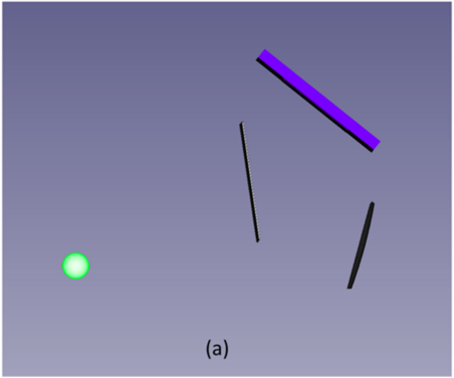
- First iteration created using inbuilt Mitsuba3 tools
  - Emitter : Spot



Detector Construction 2

- Second iteration created using FreeCAD to convert GDML to OBJ
  - Emitter : Photon

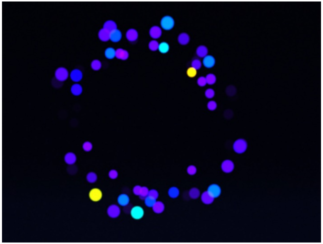




(a)

Photons

- Mitsuba emits light using "emitter" class objects
- Initial tests used "spot" emitter
  - 1 - 2 - 1 mapping of emitter instanciations
  - Caused excessive JIT compilation times
- New custom emitter "photon_emitter"
  - Single instanciation, reads photons from binary file
  - Reduced JIT compilation times to seconds from hours
  - Unintentionally fixed the geometry



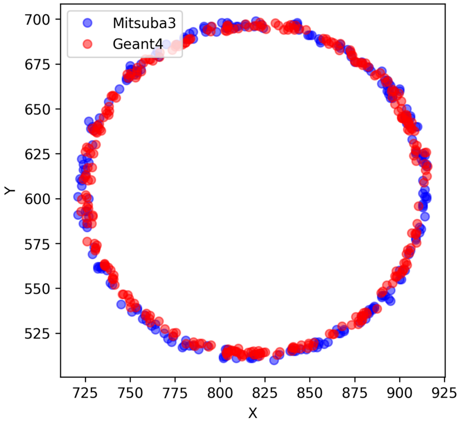
Plan

Setup

- Geant4 simulated 100GeV Muon
- Cherenkov Photons are written to file in binary format
  - (x,y,z,px,py,pz,λ)
- Mitsuba is called
  - Geometry read from OBJ files created earlier
  - Photons read from binary file by photon emitter
  - Output is numpy array
- Convert numpy to sparse matrix
  - (pixel_x, pixel_y)
- Change from Mitsuba detector reference frame to Geant4 reference frame

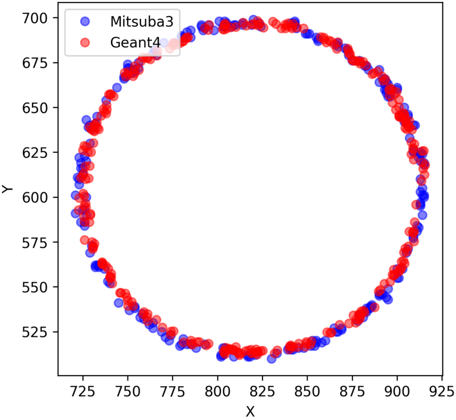
Geant4 vs Mitsuba3 Validation

- Cherenkov rings match well to Geant4
- Differences are due to image processing
  - Mitsuba3 hits are are recorded as pixel values → threshold
- Slight difference because Mitsuba3 hits are projections



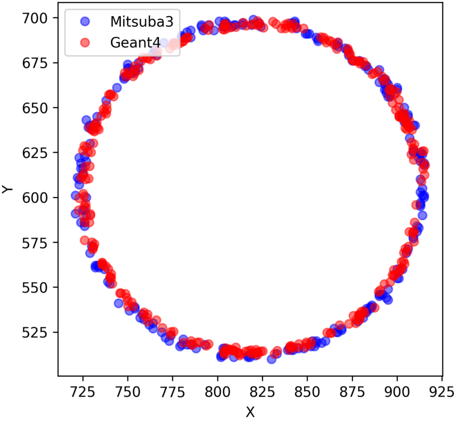
Geant4 vs Mitsuba3 differences

- Thresholding
    - Mitsuba simulates "bloom" like effects creating more hits than photons
    - Set a minimum pixel value such that: $Hits_{Mitsuba} = Hits_{Geant4}$



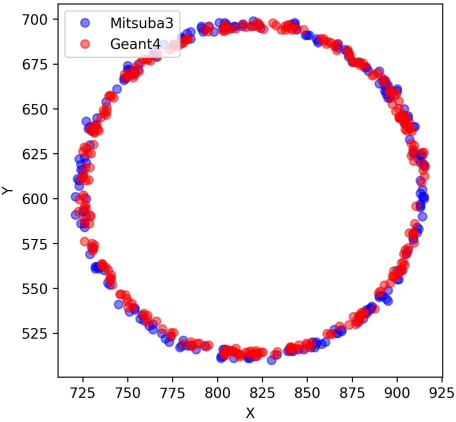
Geant4 vs Mitsuba3 differences

- Projections
  - Mitsuba renders what a camera would see
  - Need to specific distance from film, FOV etc
  - These were calibrated



Geant4 vs Mitsuba3 differences

- Detectors
  - Geant4 used a sensitive detector at the back of the PMTs
  - Mitsuba used the front face of the logical volume



Scaling Study Setup

- Mitsuba
  - Resampled Geant4 photons N times to create X total photons
  - On CPU ran 1,2,4...20 thread renders with repeats
    - Intel Xeon 2.4Ghz with 20 cores
  - On GPU repeated render 3 times
    - NVIDIA Tesla T4
- Geant4
  - Ran N events to create X total photons
  - Timing is for propagation only
  - Single thread Geant4 only

Scaling Study

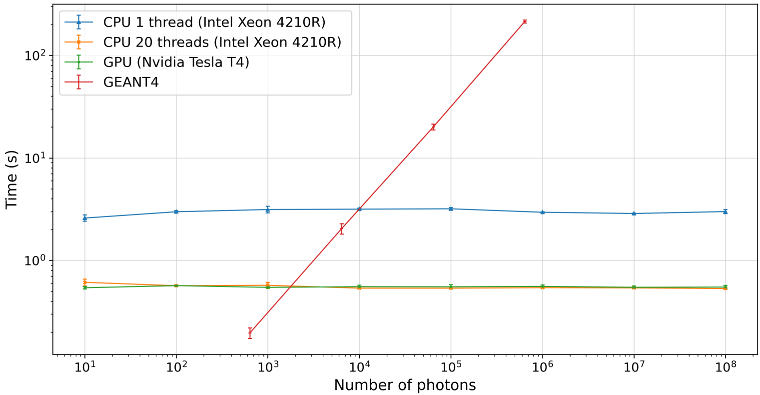


Figure: Photon Propagation time for Geant4 and Mitsuba3 in various configurations

Scaling Study

- Mitsuba3 Render() time does not increase with photon count
- Mitsuba3 outperforms Geant4 (10.7.2)
  - **70 times faster** on CPU (1 core)
  - **400 times faster** on GPU or CPU (20 cores)
- Similar GPU and CPU performance due to compilation overhead

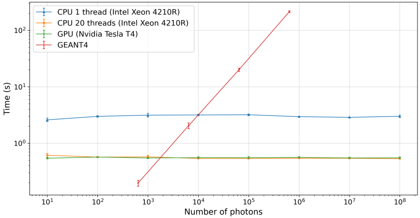


Figure: Photon Propagation time for Geant4 and Mitsuba3 in various configurations

Discussion of Mitsuba3 Timing

- Rendering time not increasing with no. of photons (yet)
- Render() includes JIT, ptracer, image projection and IO
  - Timing seems to be dominated by these overheads
  - Devs have suggested we call the ptracer directly
- Expected that the ptracer will return similar scaling to Geant4 with lower timing
- Mitsuba3 is intersection parallelised
  - Solves the binary tree problem
- Paper : Optical Photon Simulation with Mitsuba3 - https://arxiv.org/abs/2309.12496

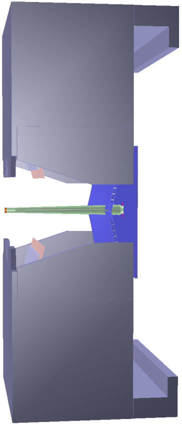
Plan

Next Stage

- Input
  - Automated geometry conversion
  - Direct Geant4 to Mitsuba photon offloading
- Output
  - Multiple detector read out
  - Direct ray output
- Middle Bit
  - Participating mediums (water etc)
  - Scintillation

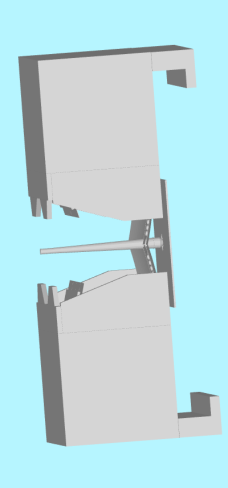
Automated Geometry Conversion

- Ideal solution
  - Direct Geant4 geometry to Mitsuba geometry
  - Requires Geant4-2-Mitsuba data structure conversion
- Practically Ideal Solution
  - Convert GDML to OBJ files
  - Use OBJ files to construct geometry



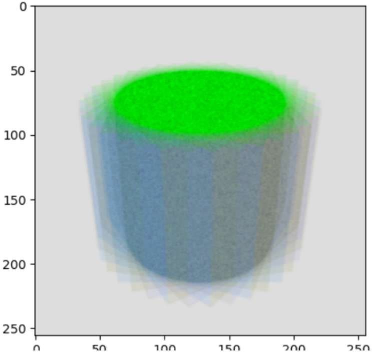
Automated Geometry Conversion

- Hierarchical scanning of GDML
    - Had to learn how to read GDMLs and construct separate world volumes
- Must create separate OBJ files
    - At least each material will need a separate BSDF to control optical properties
    - Bidirectional Scattering Distribution Function (BSDF)
    - Thin film and bulk materials **may** require BSDFs



Multiple Detectors

- Previously we have simplified our renders replacing many PMTs with a single film
  - Will not work for more complex geometries
- Working towards reading multiple films at once
- Pipeline : Image(s)
  $\rightarrow$ Sparse Matrix
  $\rightarrow$ Reference frame change
  $\rightarrow$ Combine



Plan

Summary

- Mitsuba is an open source physics based renderer
- Successfully setup a working testing pipeline
- Confirmed that Mitsuba **can** produce science quality results
- Mitsuba show impressive performance results on both CPU and GPU far exceeding Geant
- Paper has been submitted
  - And rejected
- Currently working on geometry translation and building more complex examples

Plan

Technical Info

- CPU : Intel(R) Xeon(R) Silver 4210R
  - 2.40 GHz
  - 20 Cores
- GPU : NVIDIA Tesla T4
  - 70W Low Power GPU
  - 16GB GDDR6
  - 2560 CUDA Cores
  - 40 RT Cores
  - 585 MHz Base Clock (1590MHz Boost)