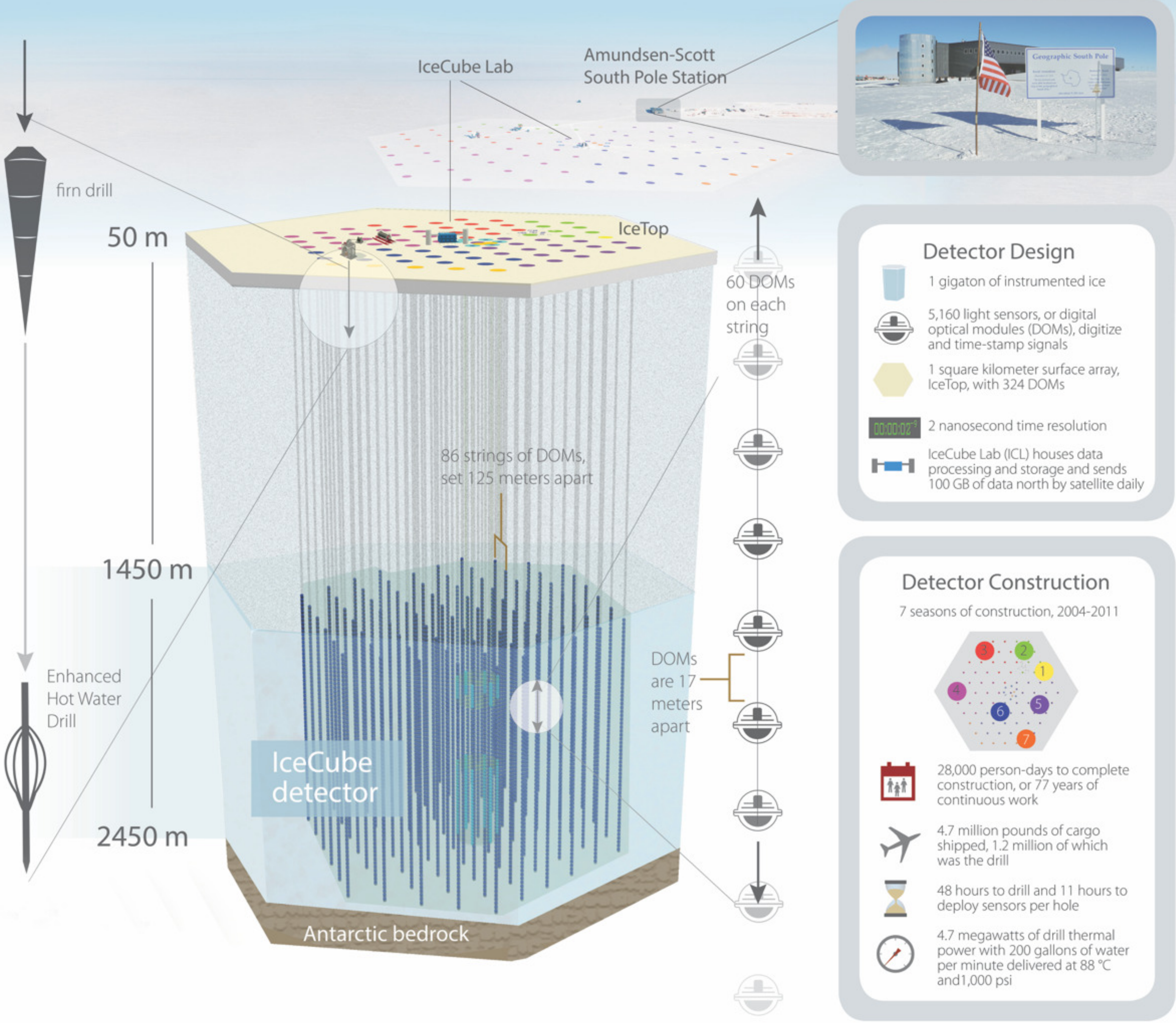
Parallel Photon Simulation for IceCube In C++

Kevin Meagher
University of Wisconsin
Oct 21, 2024
CHEP2024 - Kraków, Poland

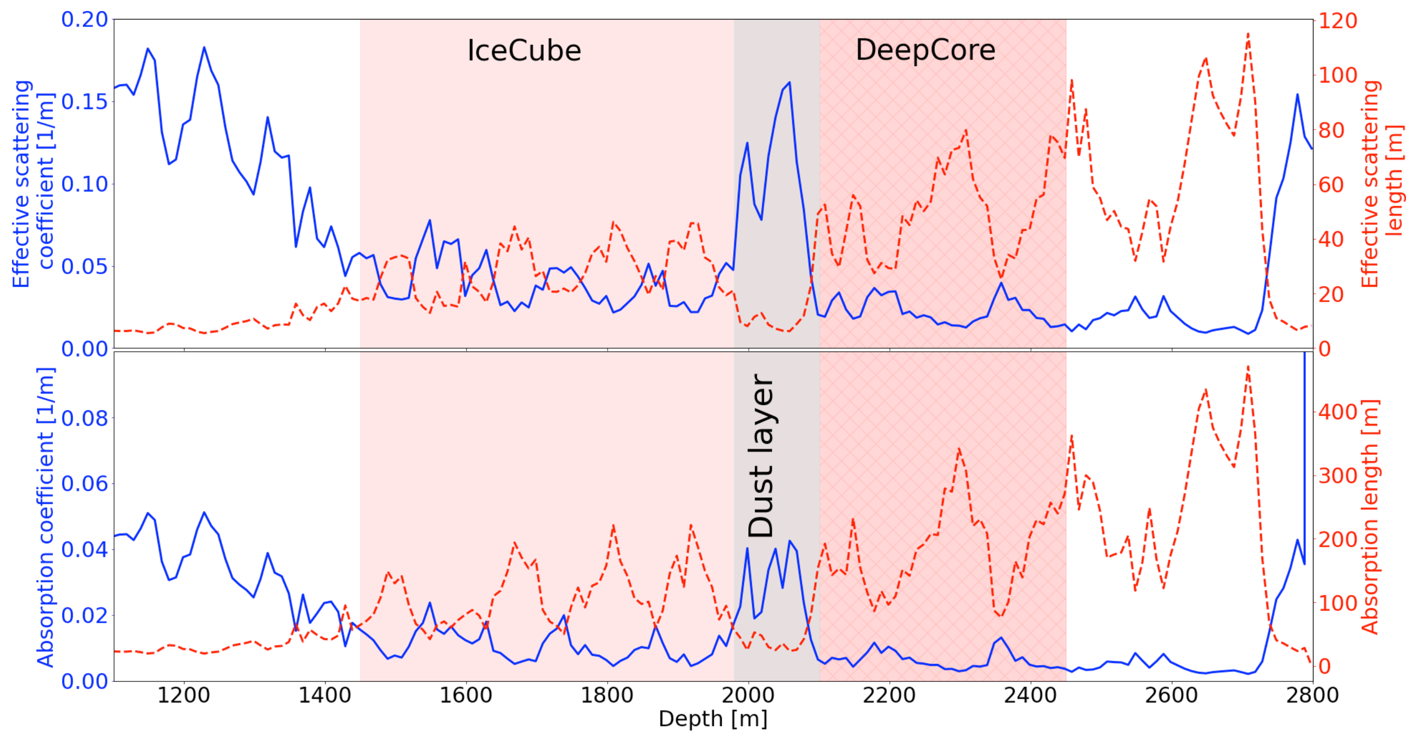
ICECUBE NEUTRINO OBSERVATORY
WISCONSIN ICECUBE PARTICLE ASTROPHYSICS CENTER
WISCONSIN UNIVERSITY OF WISCONSIN–MADISON
NSF

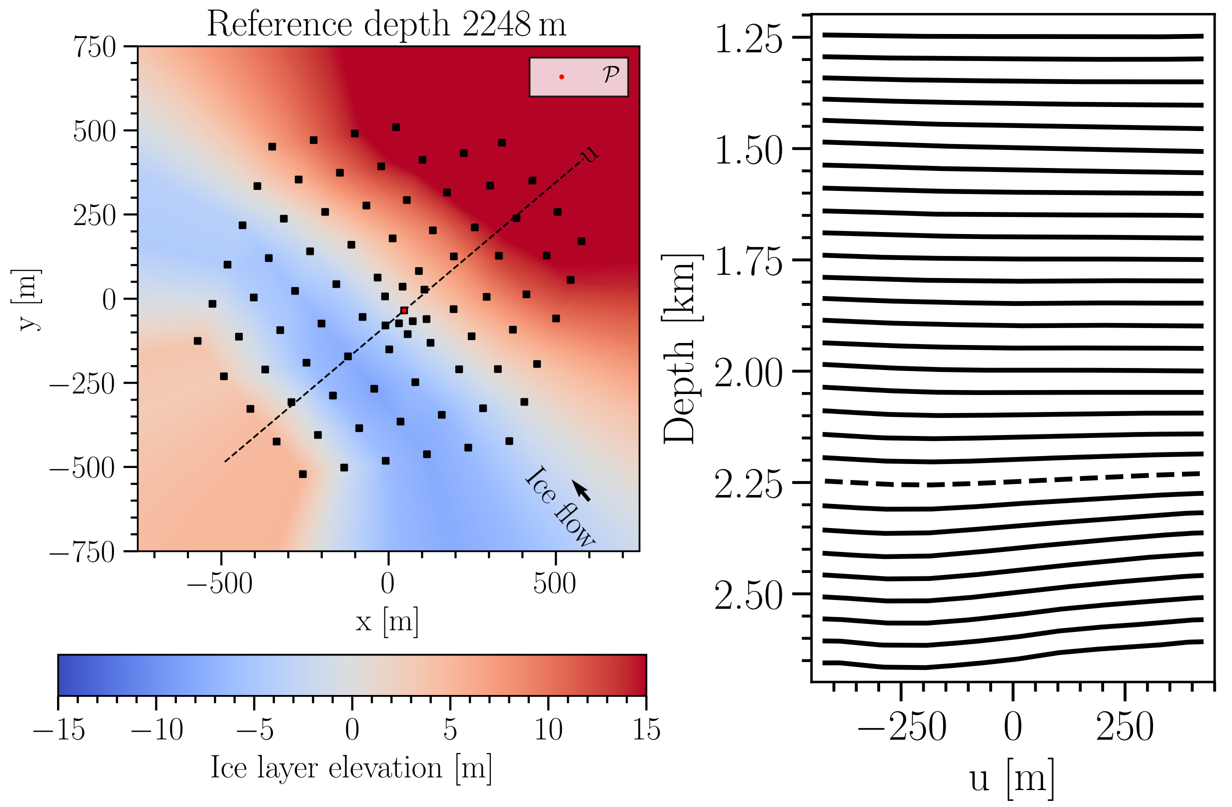
Overview

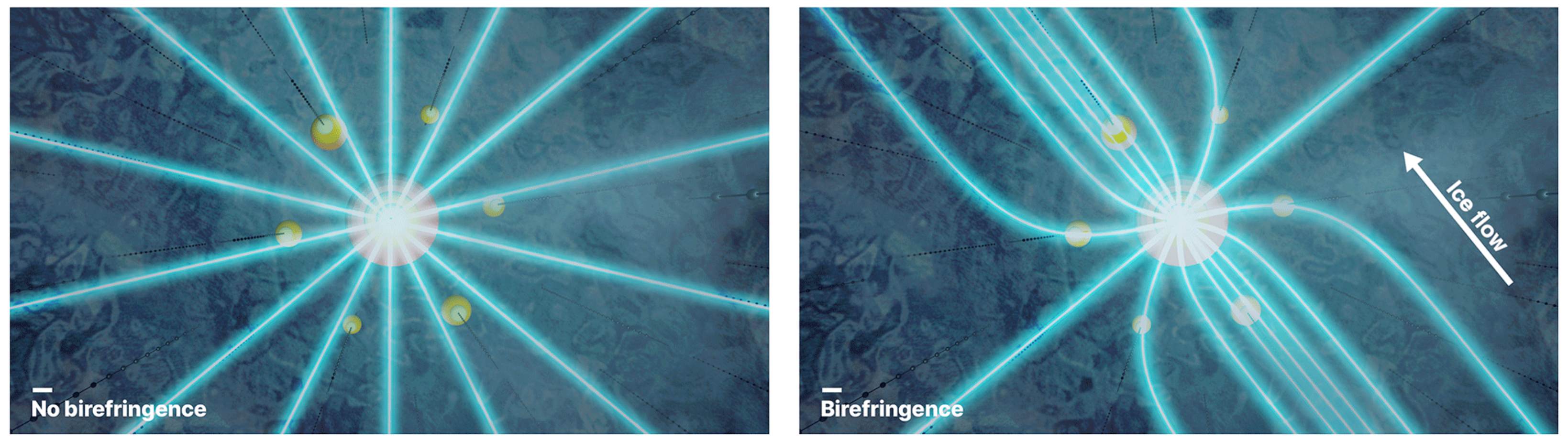
- Introduction to IceCube Neutrino Observatory
- Photon Propagation Simulation in IceCube
- Implementation in C++ with std::par


IceCube Lab
Amundsen-Scott South Pole Station
firn drill
50 m
IceTop
60 DOMs on each string
86 strings of DOMs, set 125 meters apart
1450 m
Enhanced Hot Water Drill
DOMs are 17 meters apart
IceCube detector
2450 m
Antarctic bedrock
Detector Design
1 gigaton of instrumented ice
5,160 light sensors, or digital optical modules (DOMs), digitize and time-stamp signals
1 square kilometer surface array, IceTop, with 324 DOMs
2 nanosecond time resolution
IceCube Lab (ICL) houses data processing and storage and sends 100 GB of data north by satellite daily
Detector Construction
7 seasons of construction, 2004-2011
28,000 person-days to complete construction, or 77 years of continuous work
4.7 million pounds of cargo shipped, 1.2 million of which was the drill
48 hours to drill and 11 hours to deploy sensors per hole
4.7 megawatts of drill thermal power with 200 gallons of water per minute delivered at 88 °C and1,000 psi

Properties of South Pole Ice



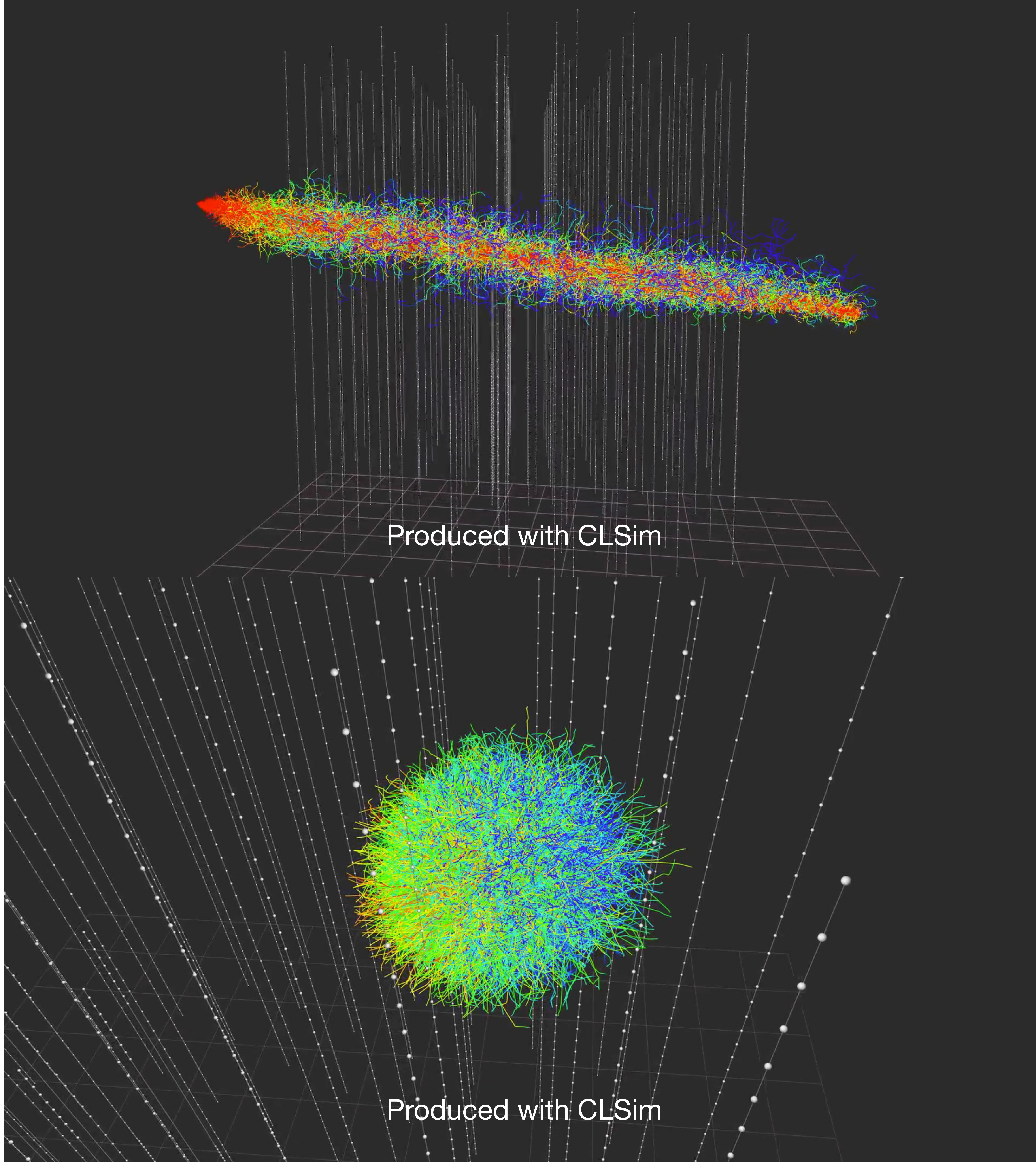




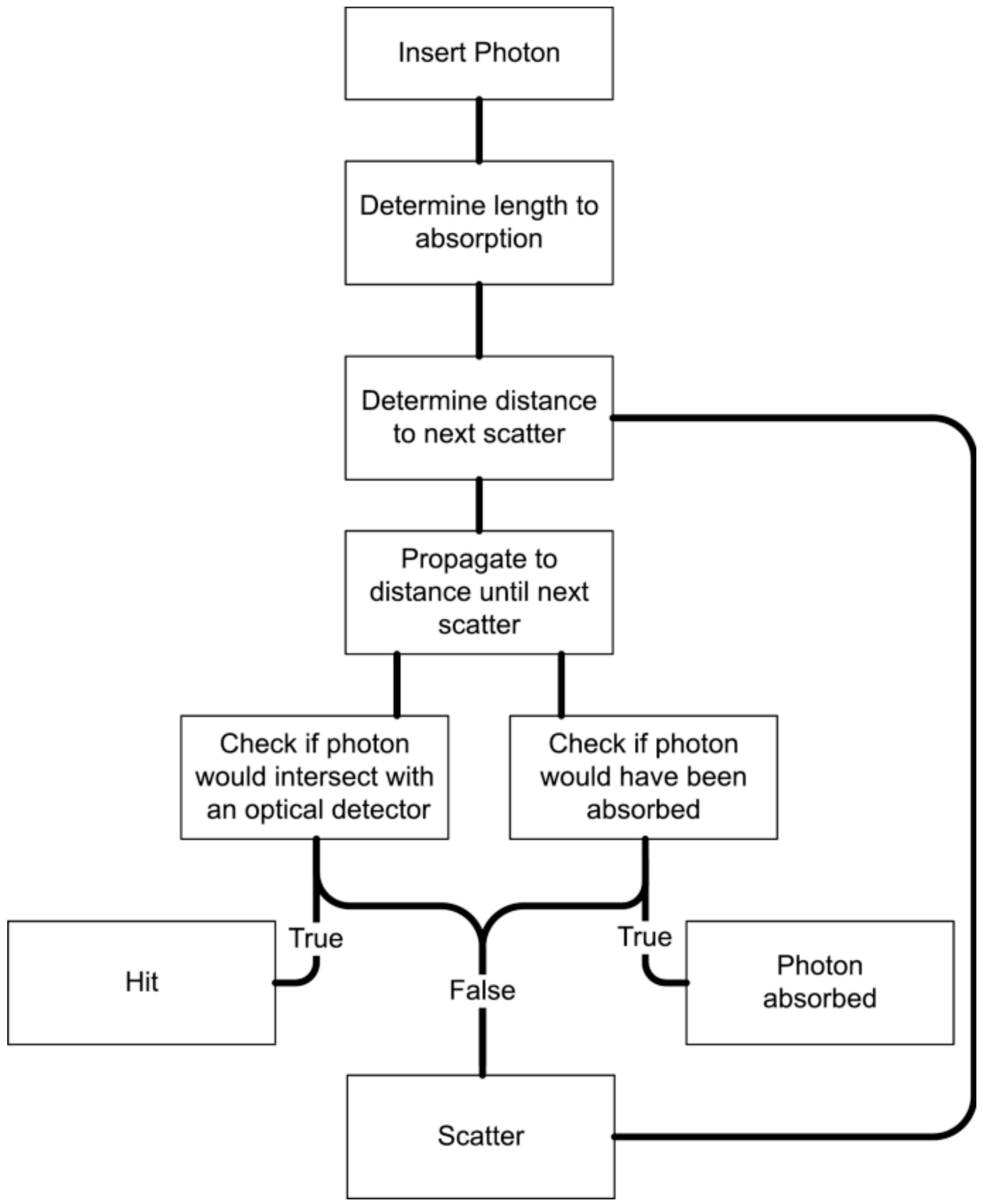
- Detector size ~1 km
- Attenuation length of ~100m Much clearer than any laboratory grown ice
- Scattering length ~20m
- Yearly variations in snow deposition cause the optical properties to vary in horizontal layers
- Ice layers are slightly tilted
- Birefringent effects from ice polycrystals causes light to deflect towards the axis of glacial flow

Challenges of Simulating South Pole Ice

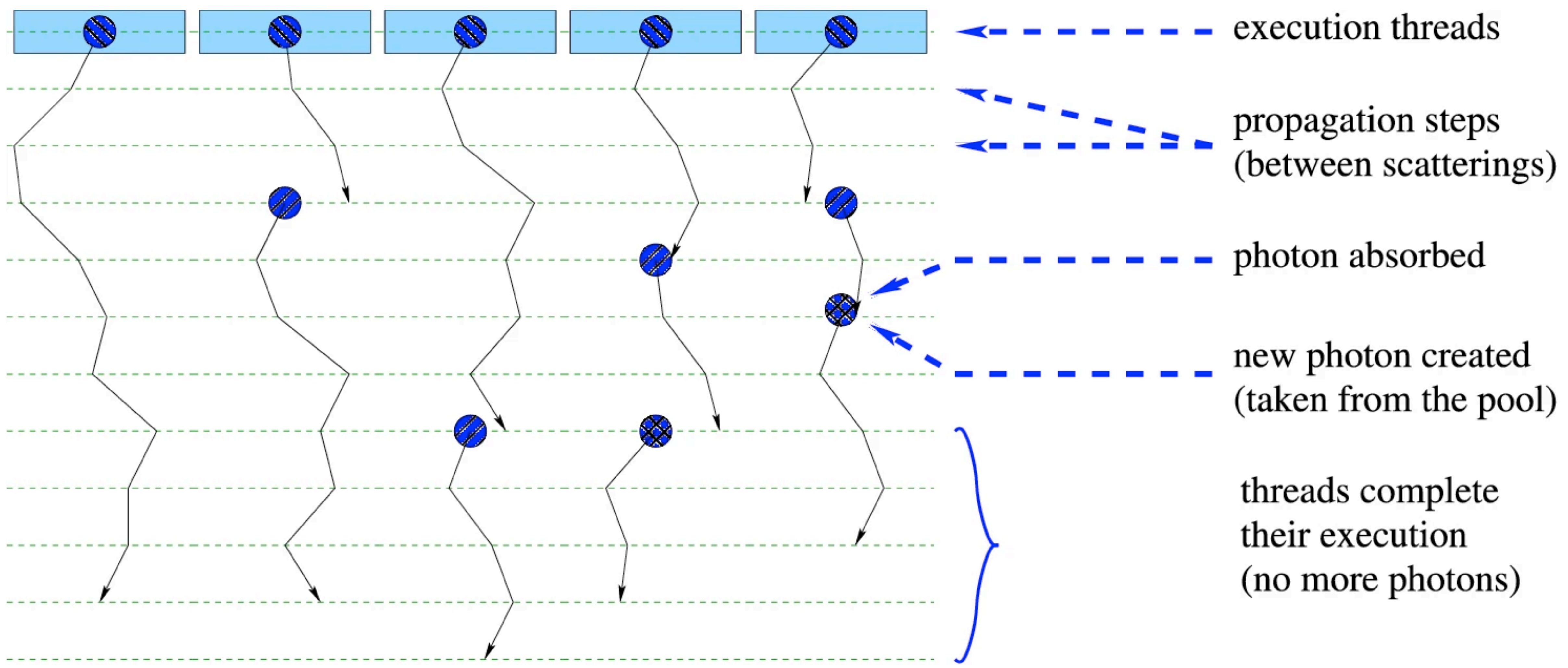
- Photon simulations are necessary for IceModel studies as well as simulation and reconstruction of IceCube events
- Uncertainties in ice properties are the dominant source of uncertainty for many IceCube analyses
- The required quantity of simulation necessitate code highly optimized for our use case
- No off the shelf software is capable of propagating photons with ice properties and the needed optimizations



Photon Propagation Algorithm



- Algorithm is embarrassingly parallel
- Propagation is significantly effected by anisotropic effects and scattering and absorption layers
- Detectors are clustered into layers to speed up collision detection with optical modules
- GPU kernel is ~1.5K lines of code



IceCube currently has two code bases that implement this algorithm

- CLSim
  - Implemented in OpenCL which is seeing declining support from hardware vendors and increasing errors are being seen in simulation production
  - Kernel source code is generated and compiled at runtime
  - Ice model changes require kernel recompile making systematic studies difficult
  - Lots of preprocessor macros and generally hard to understand code
  - Tightly coupled to IceCube's processing software which makes it difficult to use for ice studies
- PPC
  - Multiple implementations of the propagation kernel but CUDA is primarily used
  - Used as a R&D tool for ice model studies
  - Does not have support for systematic variations of ice properties
  - Requires environment variables for configuration and does not integrate well with IceCube's software

It was decided that it was necessary to develop a new library that is more maintainable than either CLSim and PPC

Future Computing Environments

- OpenCL is seeing declining support amongst hardware vendors and we are seeing more and more errors from OpenCL on our current grid infrastructure
- Next generation exoscale computing clusters are being planned with Nvidia, AMD, and Intel we need a solution that will work with all vendors
- All three vendors are supporting a C++ solution to GPU offloading:
  - Nvidia has supported std::par on their HPC compiler nvc++ the successor to the PDI compiler
  - AMD recently added std::par support for ROCm a branch of llvm/clang
  - Intel offers support through OpenMP/OneAPI
- It was decide that a c++ implementation of photon propagation would be the most future proof

Std::par — Parallel directive for C++17

C++17 included parallel execution policies in std::foreach(), where the implementation of parallel execution is left to the compiler

```
std::for_each(
  std::execution::par_unseq,
  photon_steps.begin(),
  photon_steps.end(),
  [](PhotonStep &step) {
    photosim::kernel::propKernel<F>(step);
    }
);
```

Nvidia’s compiler will offload execution to the GPU with a few restrictions:

- No system calls
- All memory is allocated on the heap
- All code comes from the same translation unit
- Mathematical standard library routines are allowed
- No CUDA reserved words [Undocumented]

Advantages of C++ std::par over CUDA

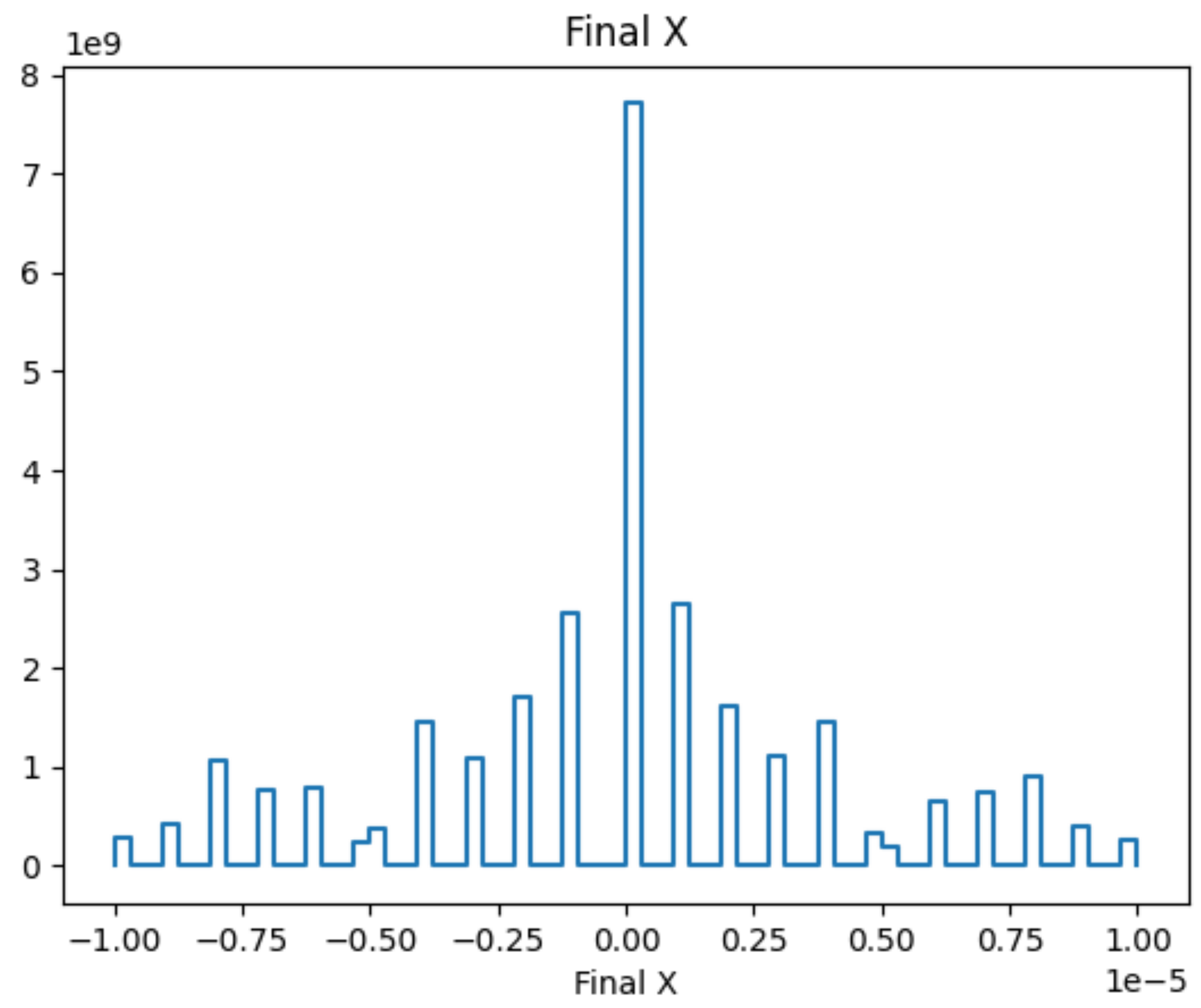
- Easy to develop — C++ has advanced data structures for memory management as opposed to the low level memory management needed for CUDA/OpenCL
- IceCube collaboration has extensive experience with c++
- Same code can be run and debugged with gcc or clang
- All three major vendors support C++ (two of the three support std::par)

Specifications

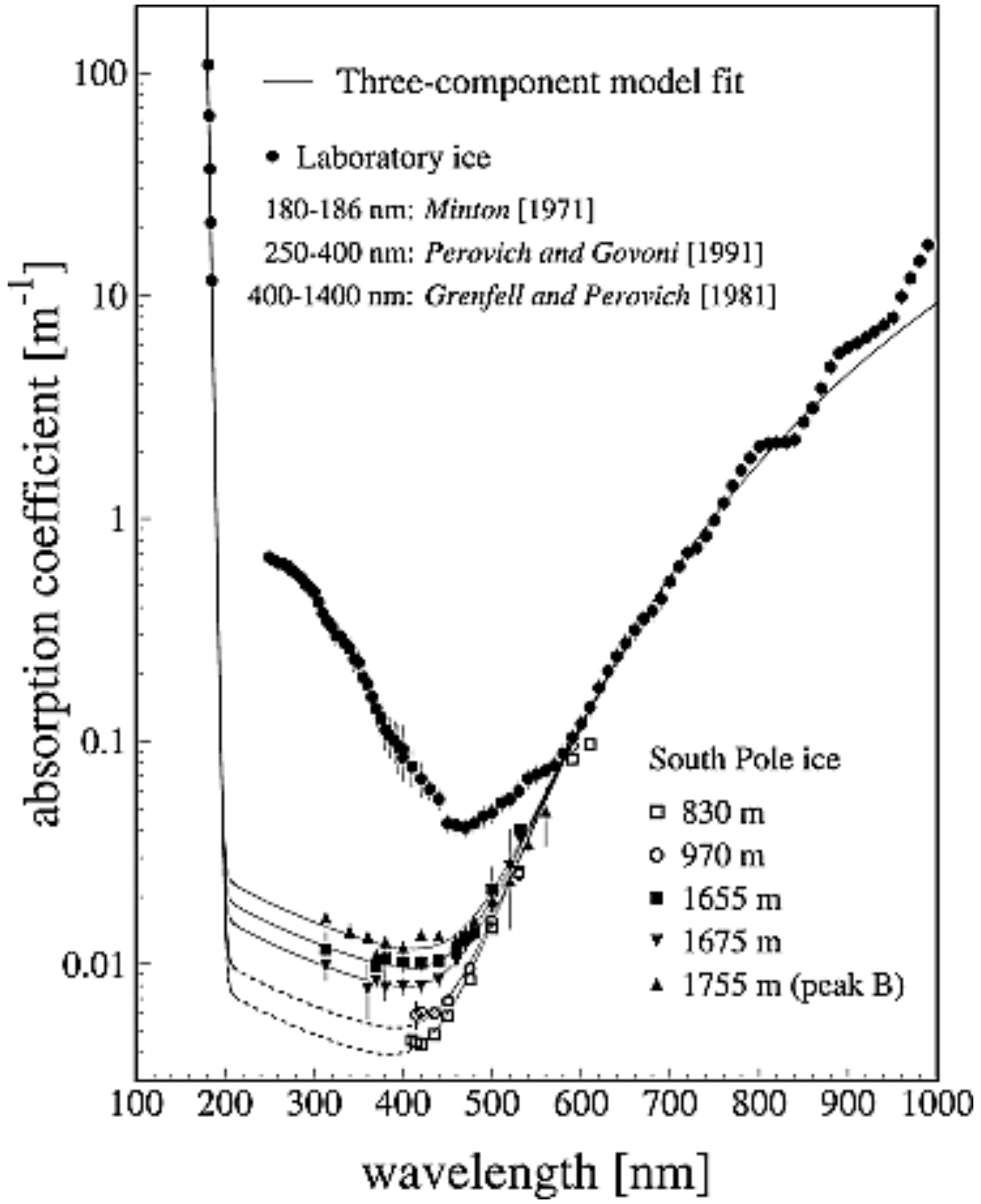
- Implementation of photon propagation in C++ using std::par for parallelization.
- The initial implementation will use Nvidia's HPC compiler: nvc++
- Stand alone library that can be used for ice model studies — It will not depend on IceCube's simulation, but will contain code such as converting Monte Carlo particles into photons, and photon interaction with optical modules.
- Support changing ice model parameters for every event — needed for systematic studies
- An interface with IceCube simulation sorfware will be provided suitable to perform IceCube simulation production.
- Runs on nvc++, g++ 14, and clang++ 18 — AMD and Intel support will be added later
- Environment variables will not be used to configure physics parameters

Progress

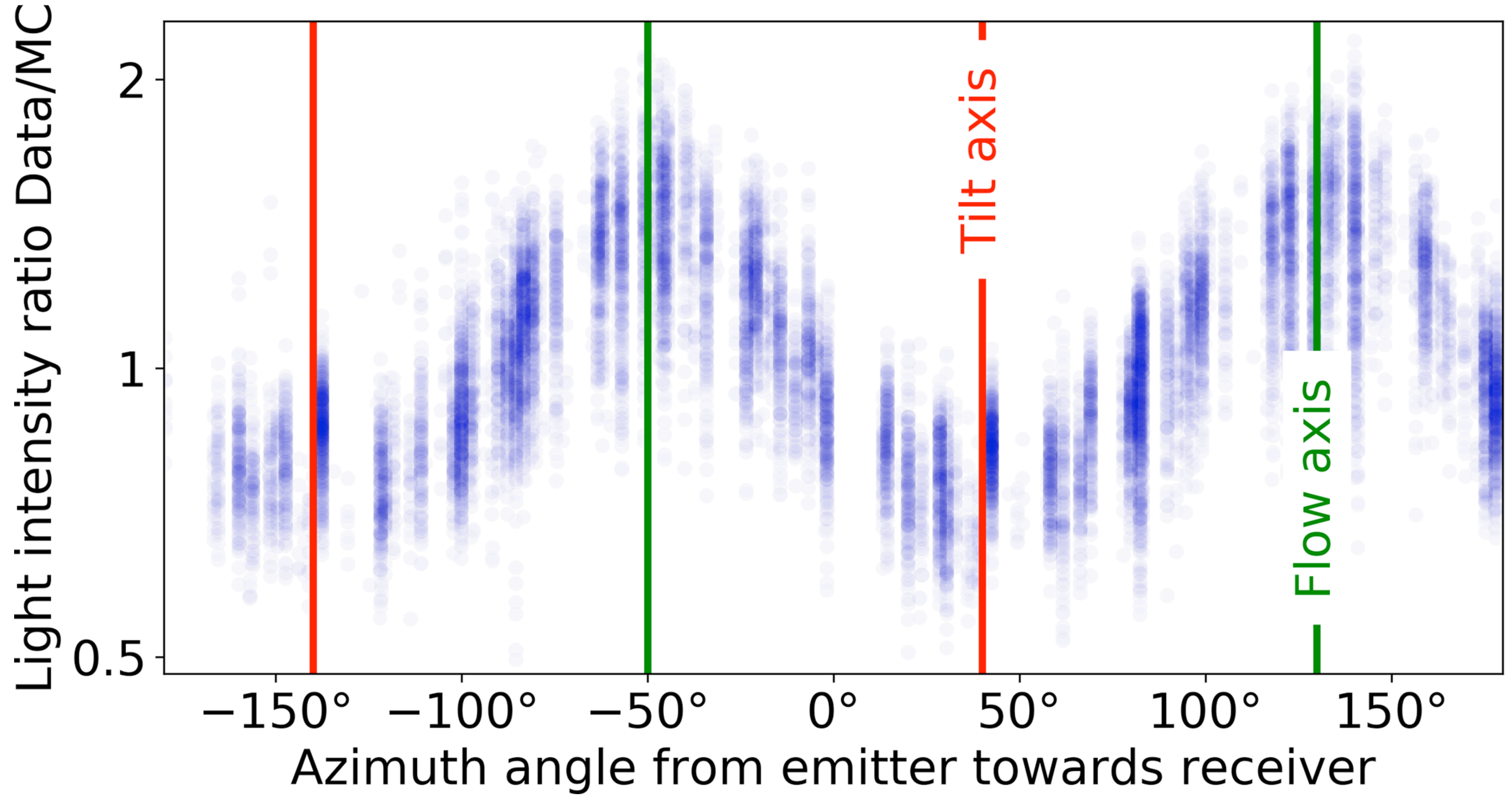
- Completed: Converted CLSim kernel code from OpenCL to C++
- Tracking of individual photons agrees with CLSim within floating point precision
- Code in std::par is successfully offloaded to GPU
- Interface to IceCube's processing framework —works with CPU propagation but causes memory violations with GPU: still working on this
- Benchmarks coming soon
- Lots more to do

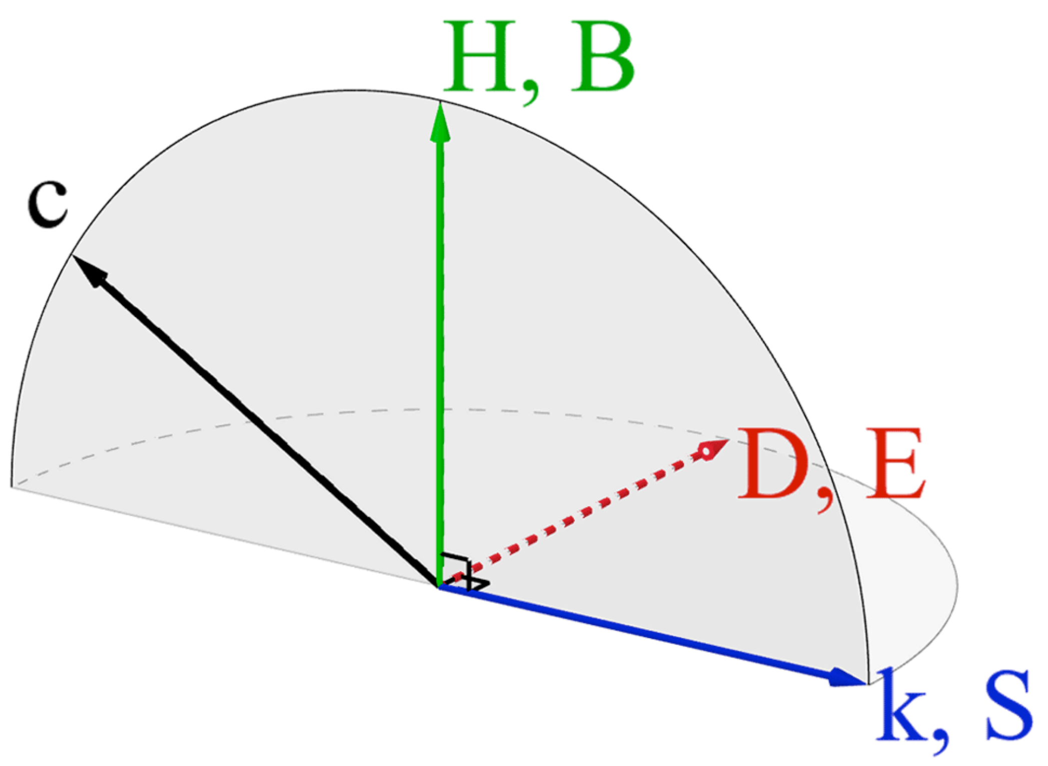


Questions

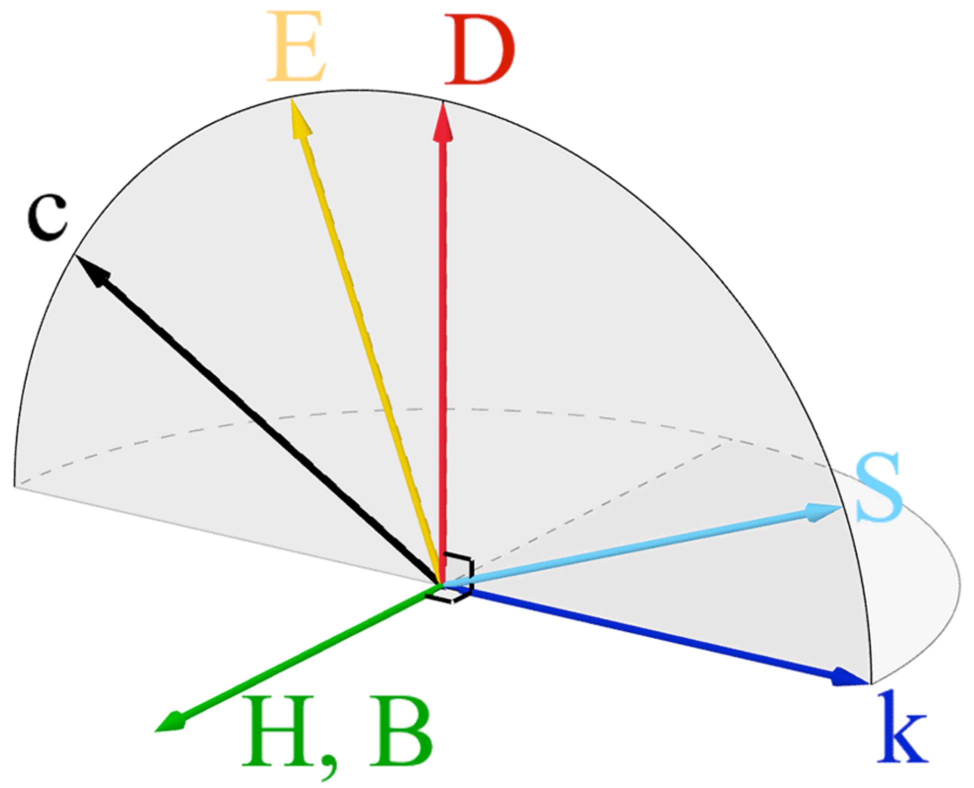


Backup





Ordinary ray



Extraordinary ray