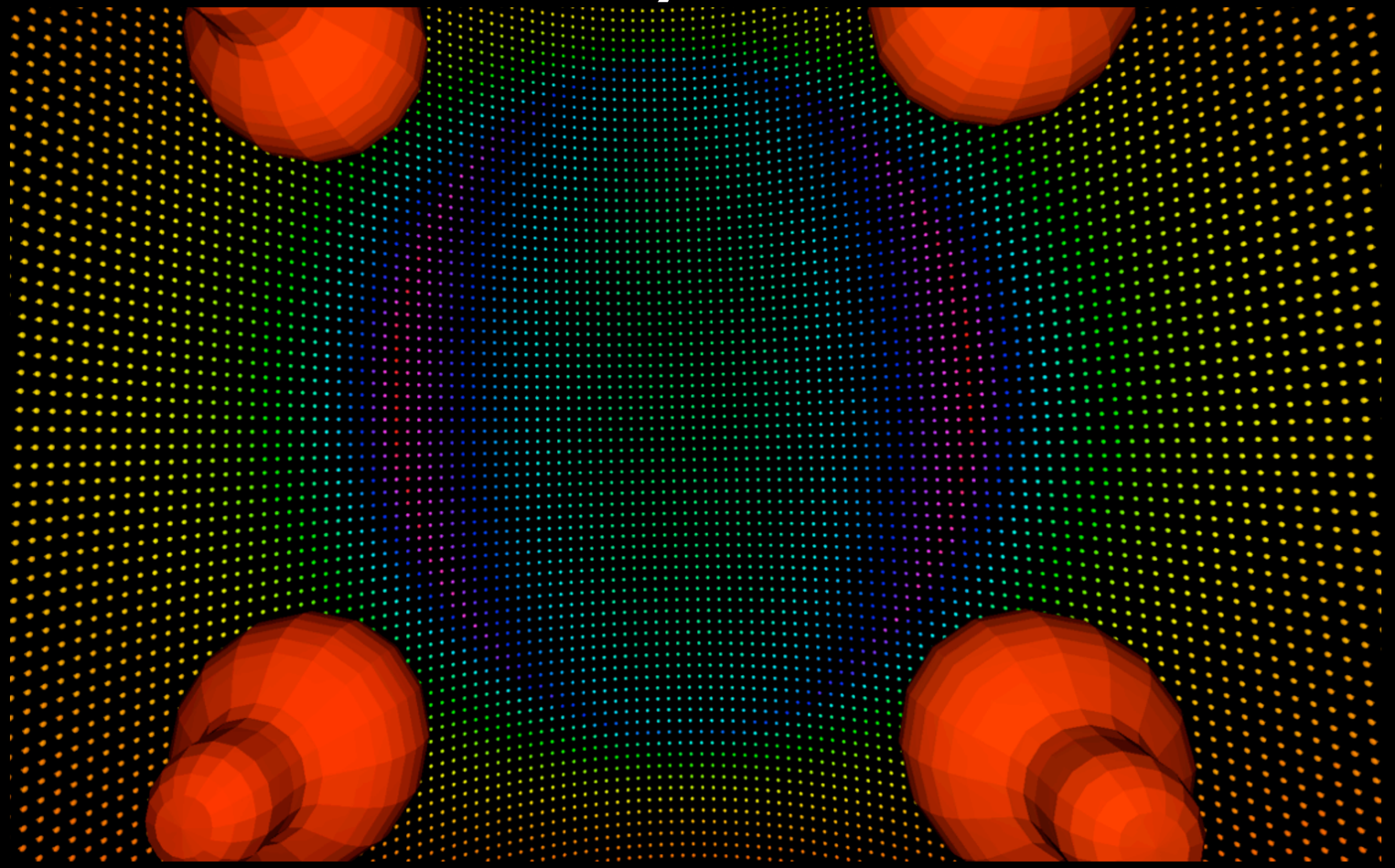
Chroma: *A Fast Photon Simulation for Particle Physics with PyCUDA*



Stan Seibert ■ University of Pennsylvania ■ GTC 2013

Hunting for Rare Interactions

Particle physicists often search for rare interactions:

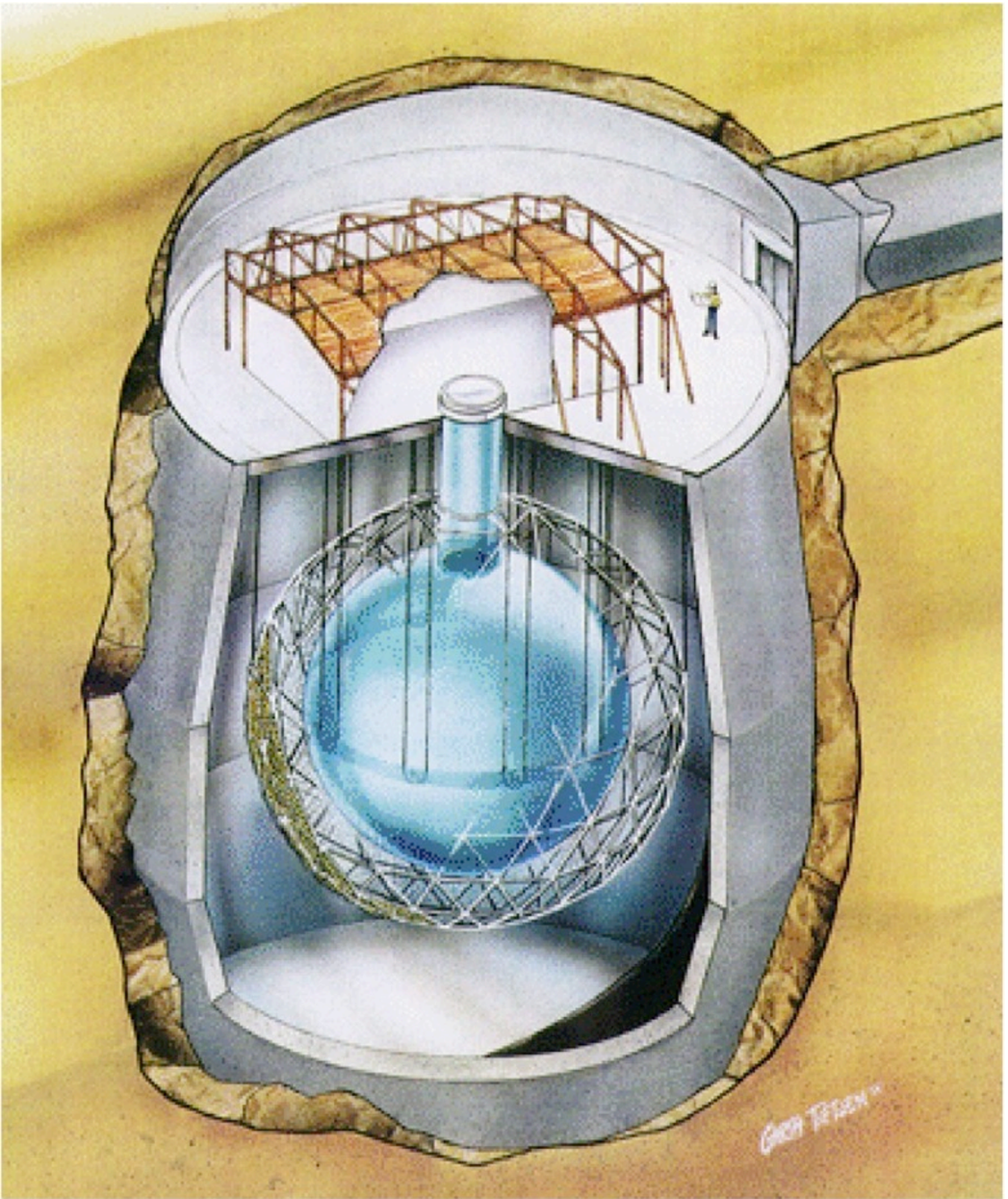
- Neutrinos (Found from many sources!)
- Dark matter (Outlook hazy, ask again later)
- Proton decay (Still looking)

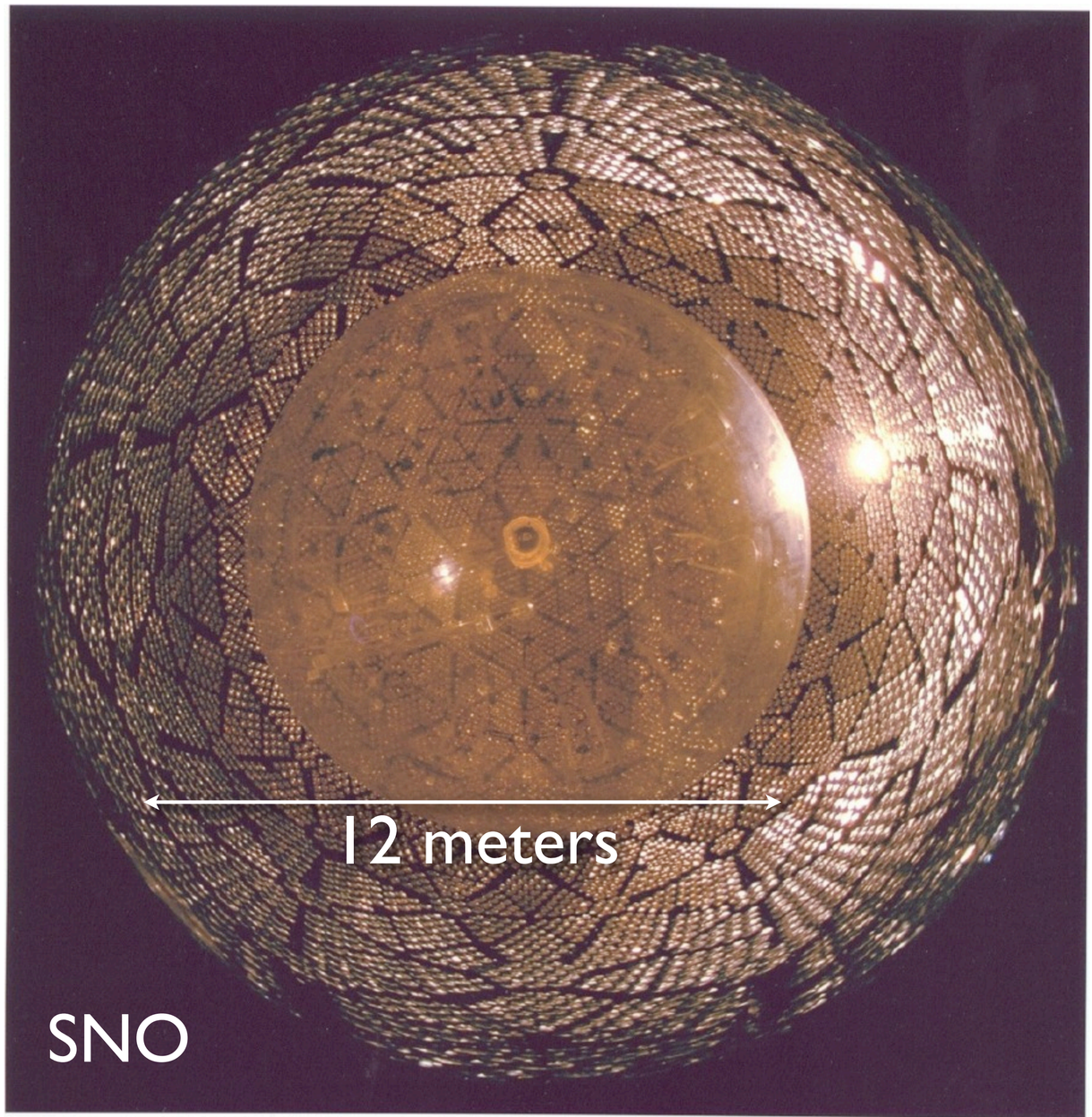
We've found great success in detectors that are:

- Big
- Can see individual photons


IMB Experiment
Big Particle Detectors, Small Amounts of Light

Big Particle Detectors, Small Amounts of Light

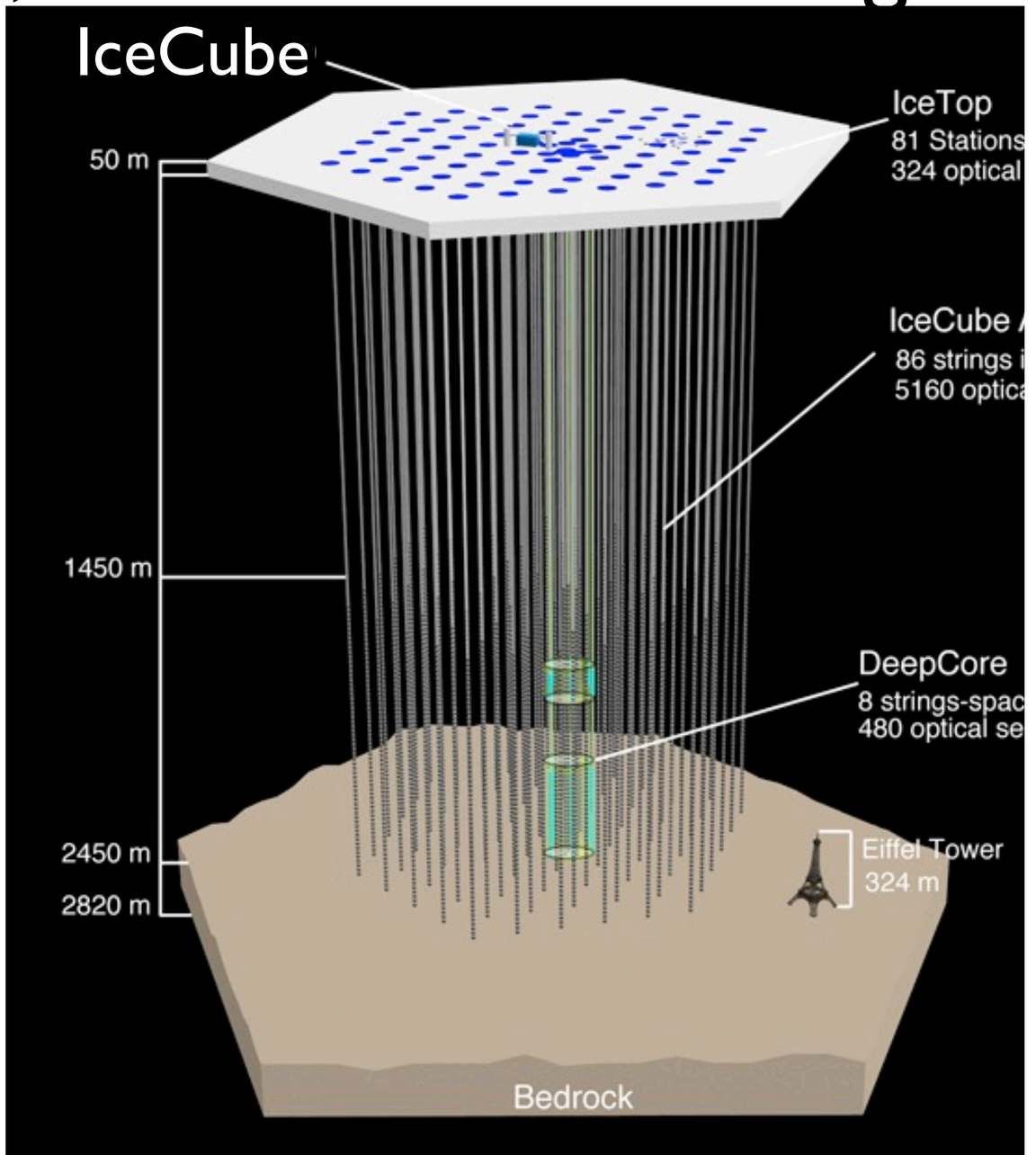




Big Particle Detectors, Small Amounts of Light

A billion tons of natural ice used as a target in Antarctica!

86 strings of detectors that can observe single photons.



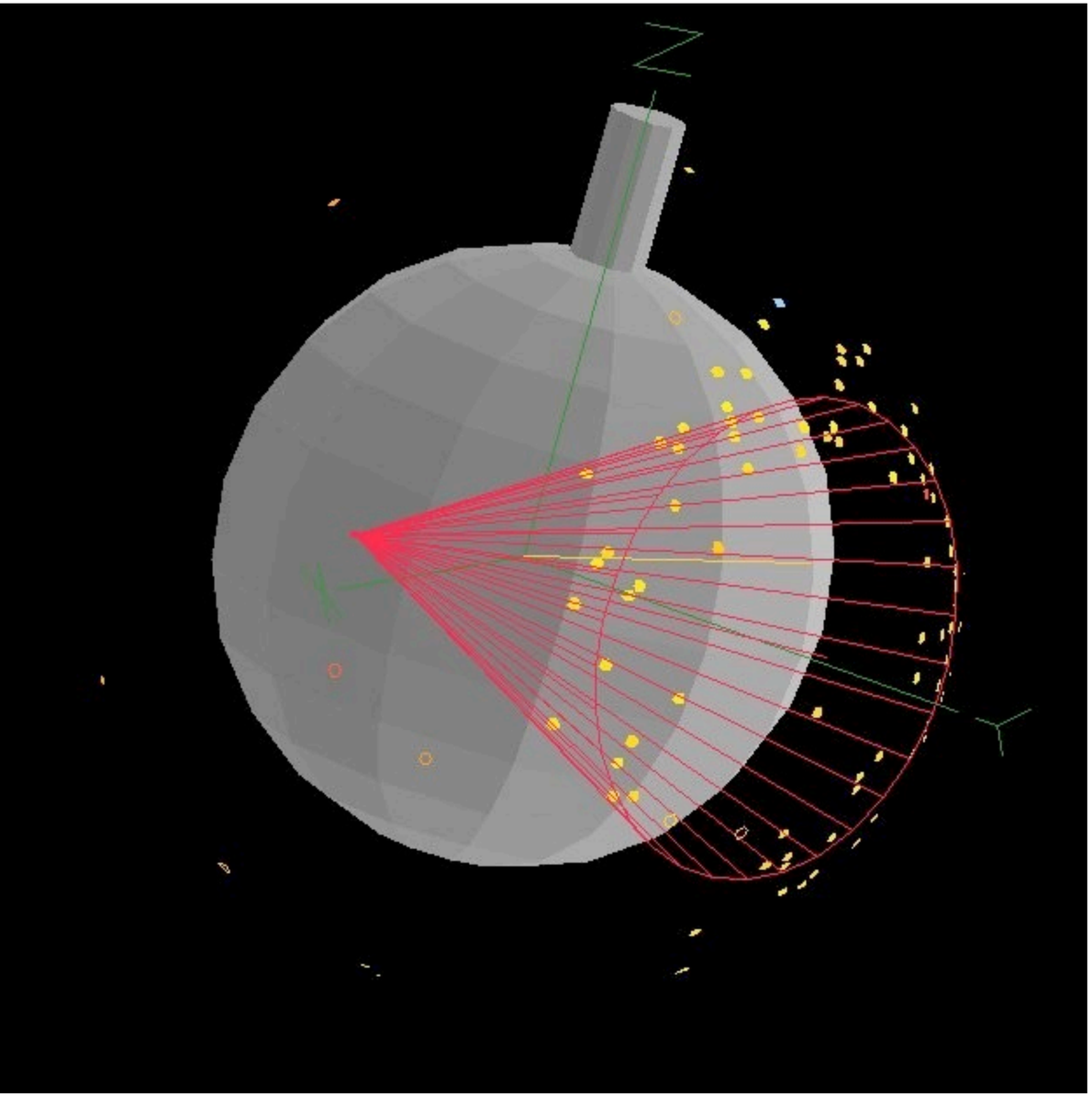
Why do Physicists Care About Ray Tracing?

- We use large detectors that detect interactions using the visible light produced in a transparent target:

  water, mineral oil, ice, etc.

- Simulation of photon propagation in these detectors has achieved percent-level accuracy.
- We rely heavily on simulation for both design optimization and data analysis.
- *The core of every particle physics simulation is a ray tracing engine!*

Tracking Each Photon



A 100W light bulb produces
10^{20} photons per second!

A particle interaction can produce thousands to millions of photons.

But, we only detect tens or hundreds of photons.

Statistical fluctuations matter in our simulations.

Borrowing From Computer Graphics

- Can't use standard ray-tracing packages because we need:
  - Individual photon tracking
  - Realistic statistical fluctuations
  - Control of optical physics to support unusual materials
- But! → We can borrow algorithms from computer graphics to speed up our photon simulations.
- Some cross-pollination of computer graphics into physics has happened already, but not enough!

What is Chroma?

- A Python module using PyCUDA that was initially written by an undergrad and a postdoc (me) working part-time.
- A GPU-accelerated ray-tracer to visualize detector geometries.
- A Monte Carlo photon simulation that includes major optical physics, like dispersion, refraction, reflection, scattering, and fluorescence. (Millions of photon steps per second.)
- Up to 200x faster at photon propagation than the standard library in the field that is used for particle simulation, GEANT4.

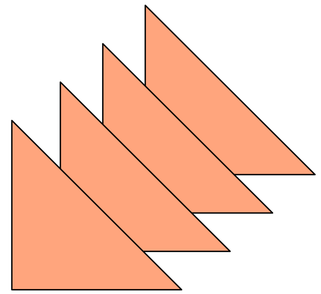
Disclaimer: We didn't know about the GPU-accelerated lattice QCD program with the same name until it was too late...

Why is Chroma so fast?

To win 200x in photon propagation over GEANT4, we had to combine two different bits of technology:

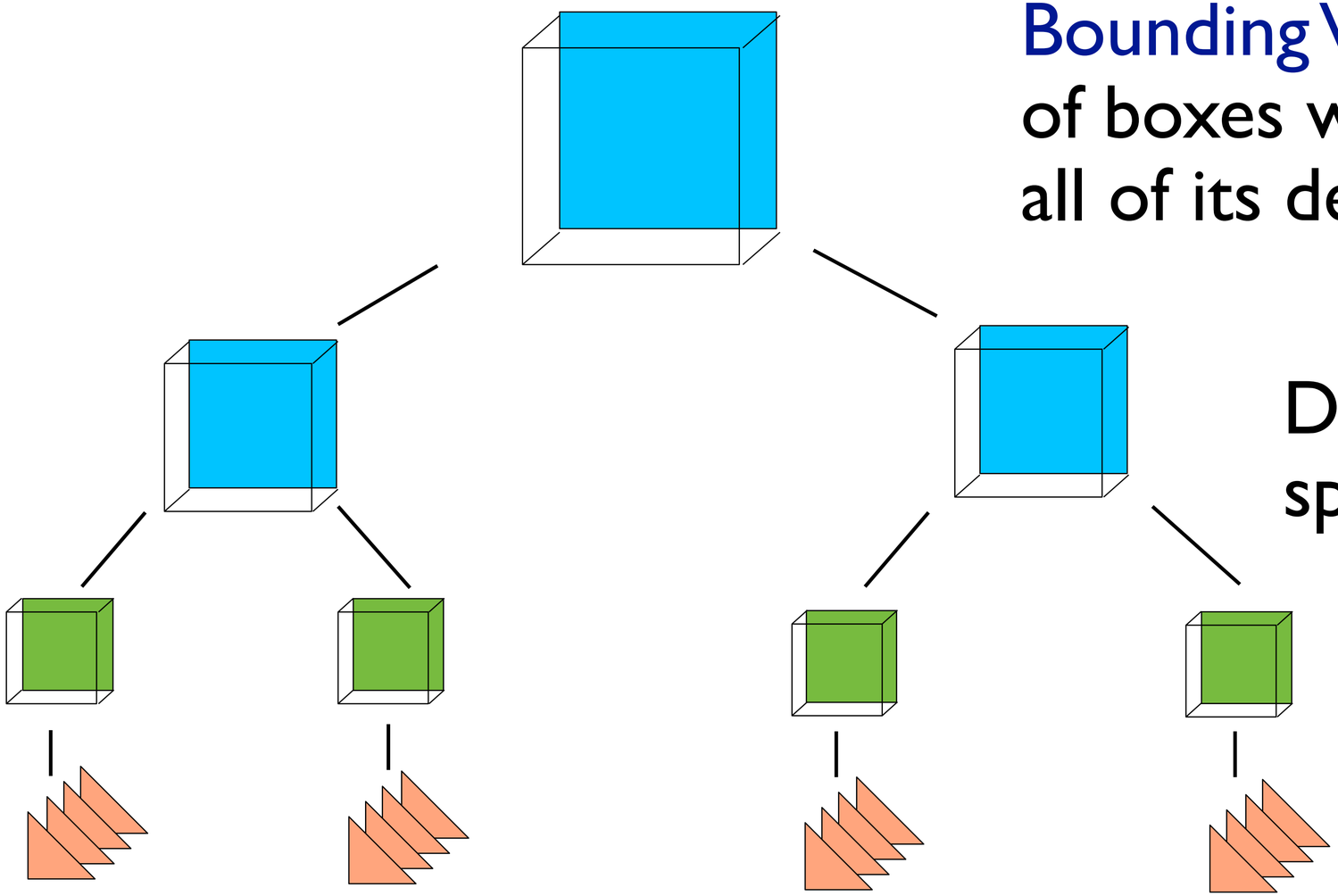
- Better software: Throw away the heavy CSG-based C++ object tree and use a flat triangle mesh. We can generate a bounding volume hierarchy for a triangle mesh that is much more efficient to traverse than a user-provided tree.
- Better hardware: GPUs provide an order-of-magnitude improvement in memory bandwidth and processing power. We propagate many photons in parallel, which can take advantage of the thousands of CUDA cores available in a modern GPU.

Faster Geometries with Triangles



- GEANT4 is slow because of the overhead of delegating geometric operations to polymorphic C++ methods. *Abstraction prevents simplification and optimization, in this case.*
- A triangle mesh can reasonably approximate most surfaces, and is pure data. No geometry-specific code → one code path to optimize.
- Fast mesh techniques are well-studied in the literature.
- Plenty of tools for manipulating triangle meshes exported from CAD files or constructed numerically.

Fast intersection with a mesh

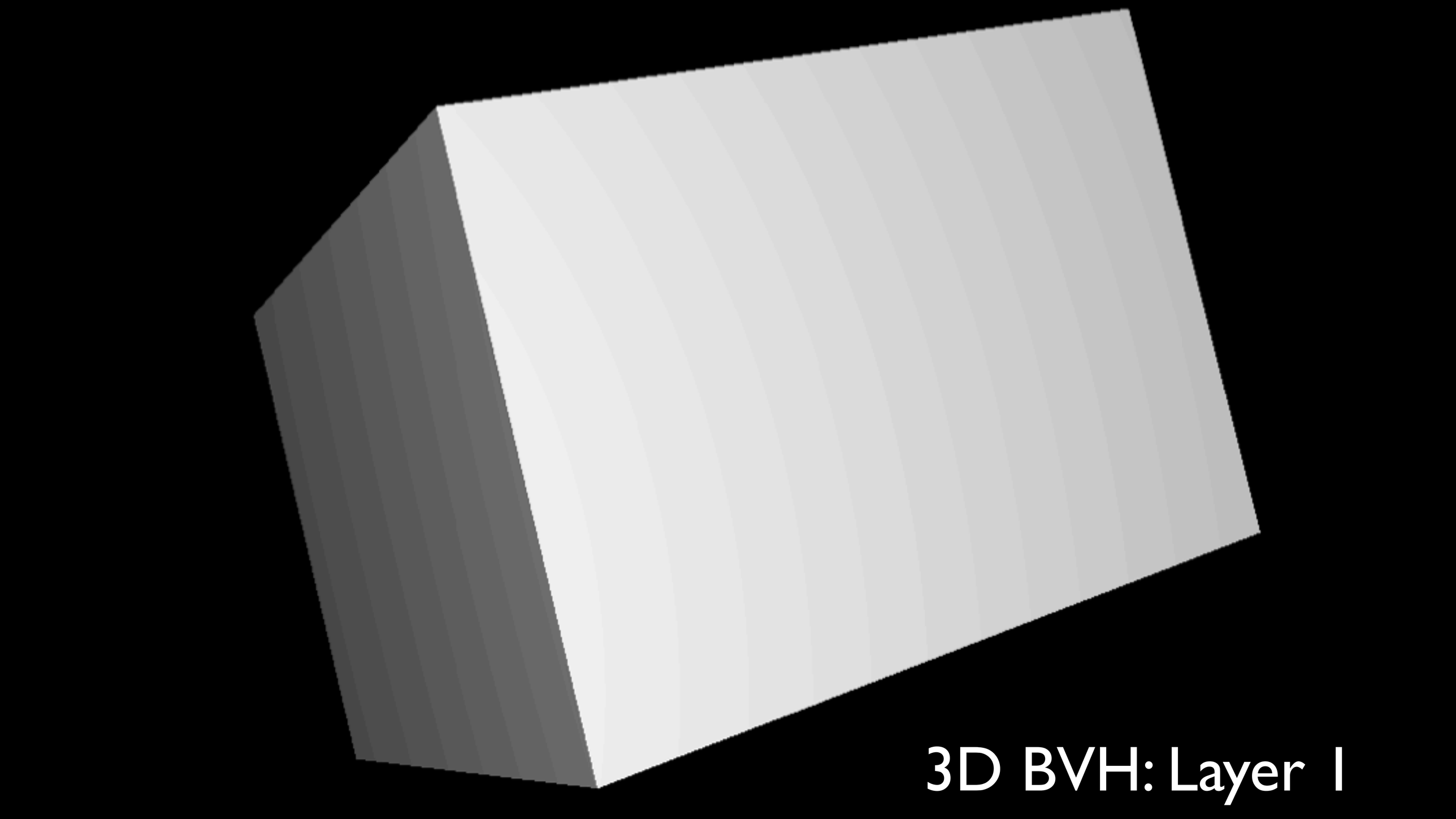


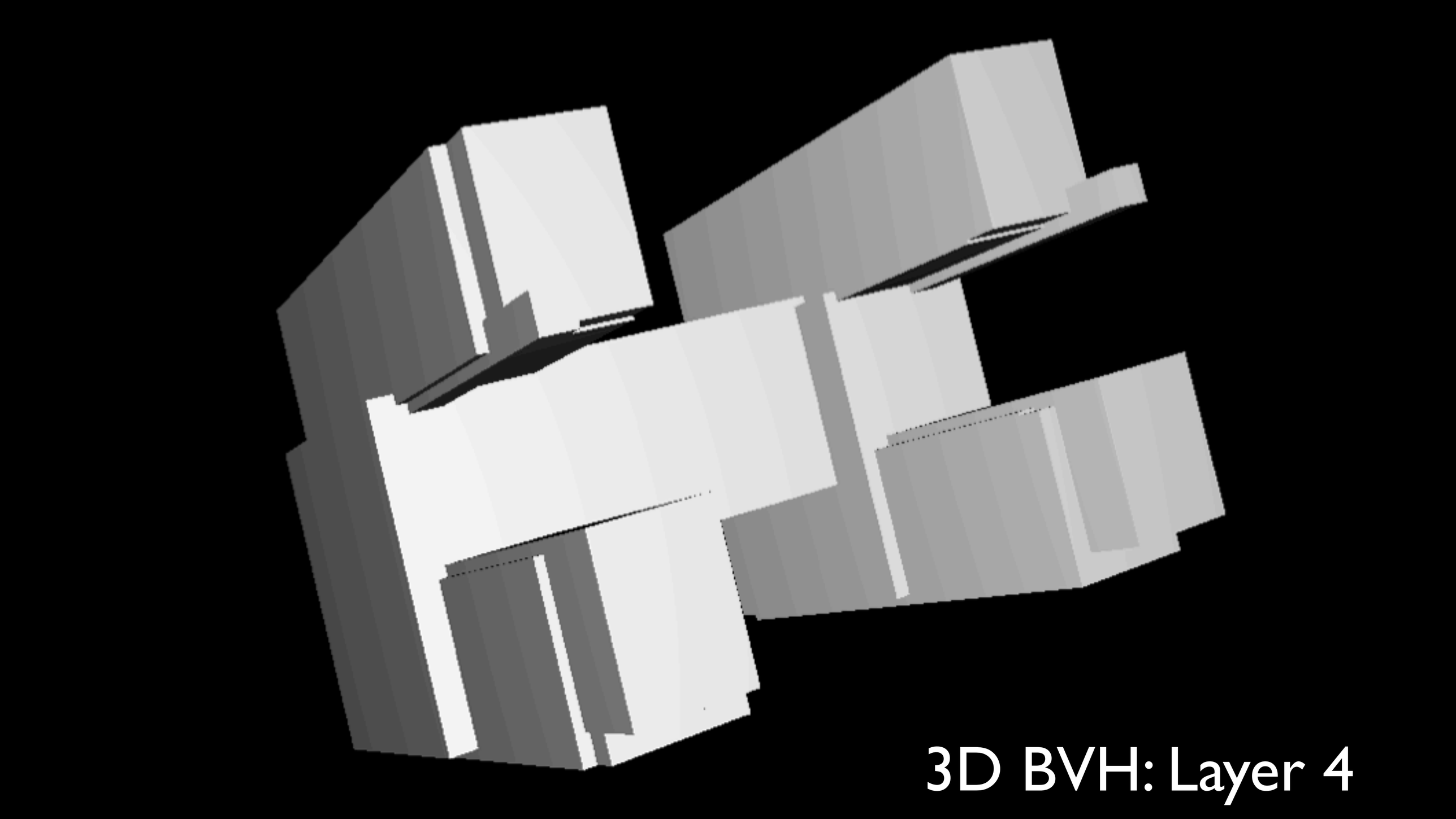
Bounding Volume Hierarchy: A tree of boxes where each node encloses all of its descendants.

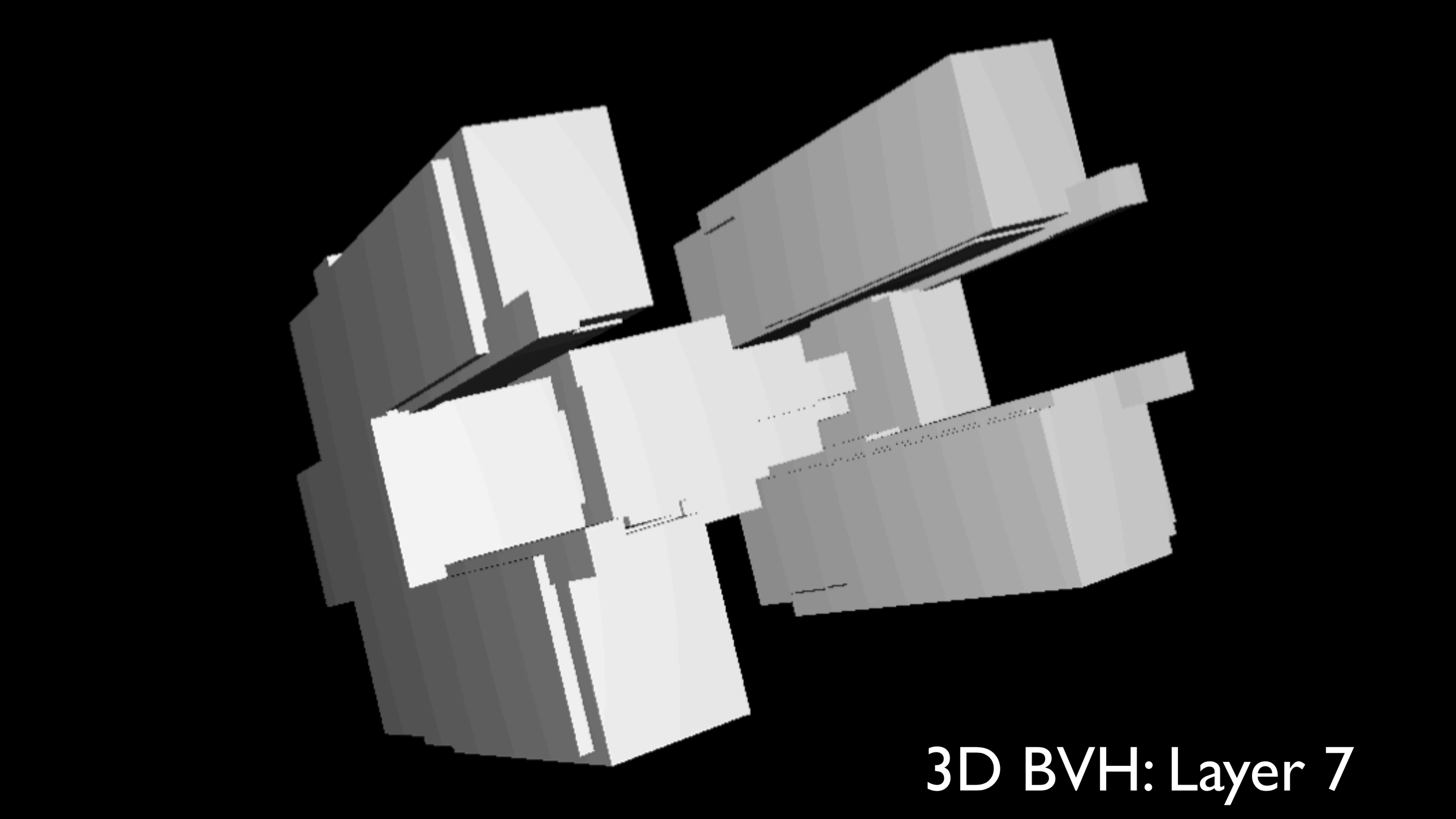
Does not need to partition the space and siblings can overlap!

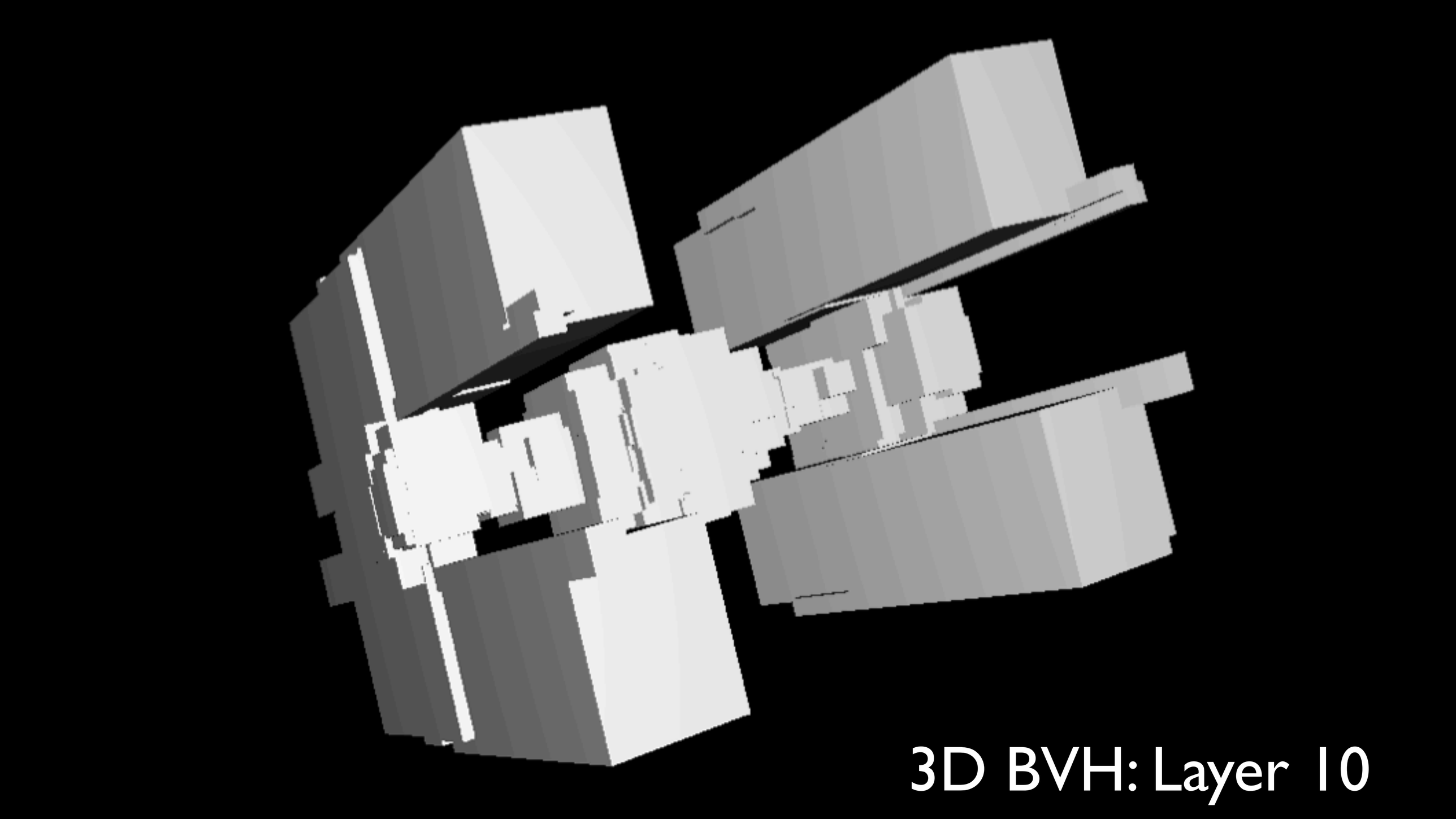
Leaf nodes contain list of triangles

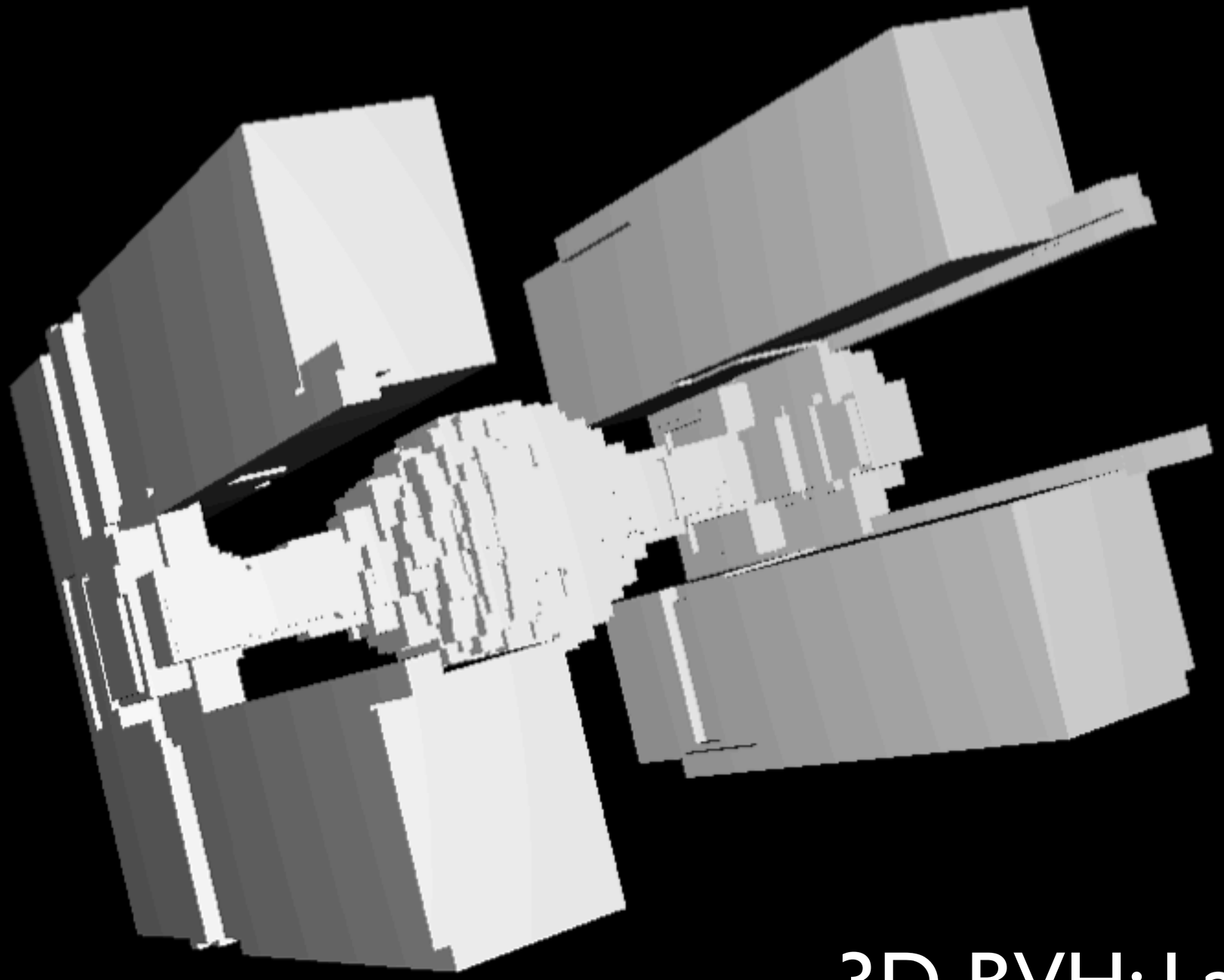
Testing a ray with a mesh of 50 million triangles only requires 130 box intersections and 2 triangle intersection tests.


3D BVH: Layer 1

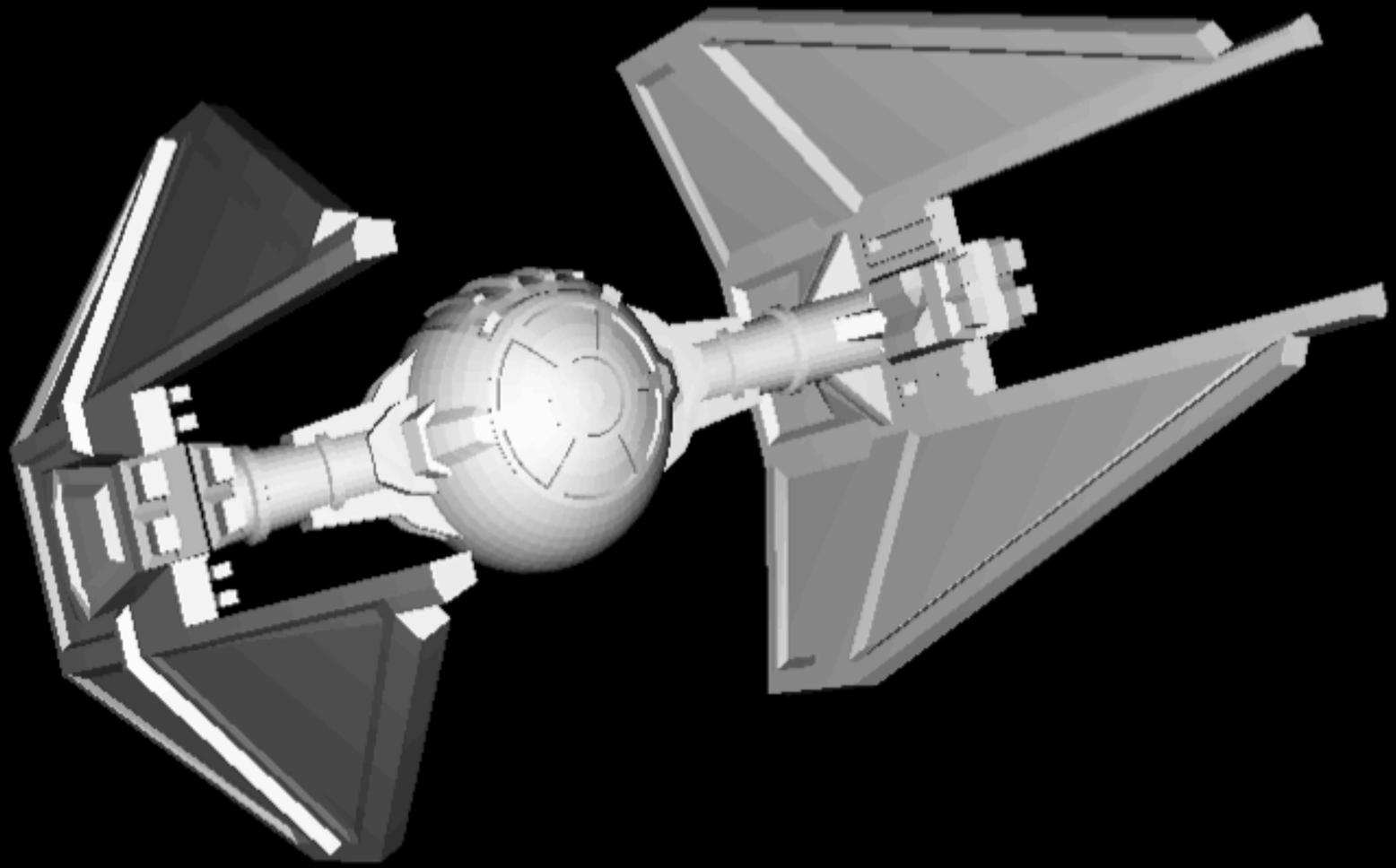

3D BVH: Layer 4


3D BVH: Layer 7


3D BVH: Layer 10



3D BVH: Layer 13



Actual Model

Detector Description

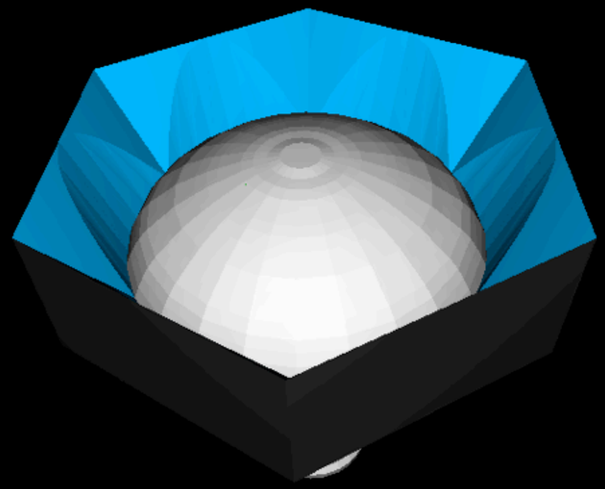
New detectors can be defined rather quickly:

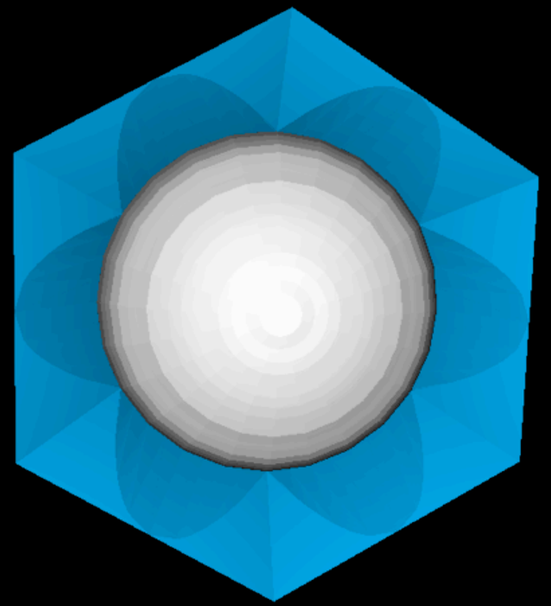
- Basic components are triangle mesh objects which can either be constructed programmatically or exported from your favorite CAD software as STL files.
- Bulk and surface material properties need to be specified, much as in GEANT4.
- *Every triangle can have a different surface material!*

  Simplifies description of objects which have a coating placed only on part of a surface.

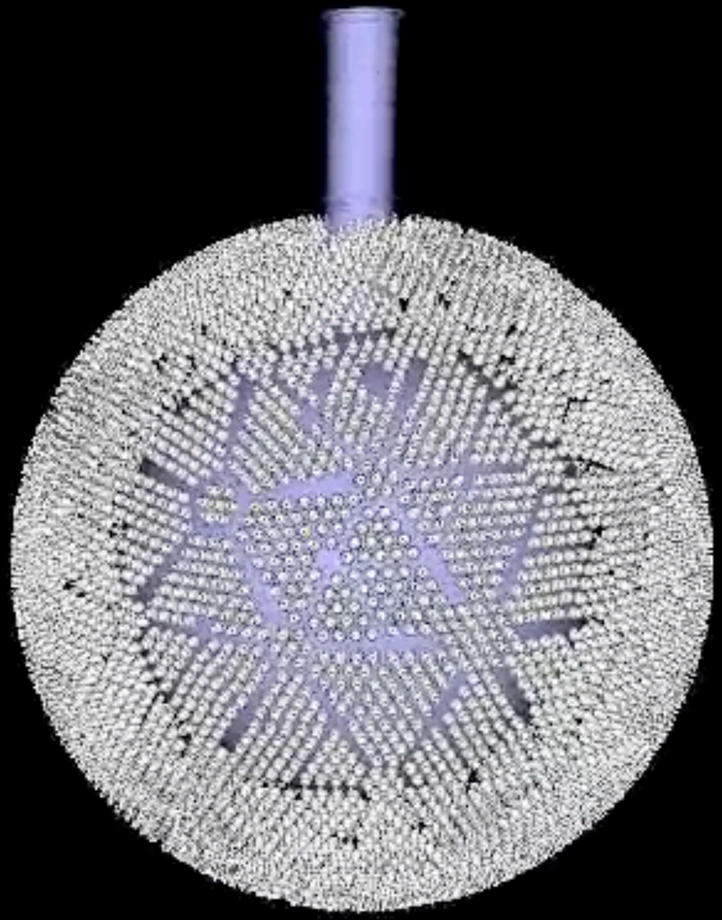
Most detectors can be created with 100-200 lines of Python + externally created solid shapes.

Photon Detector Models

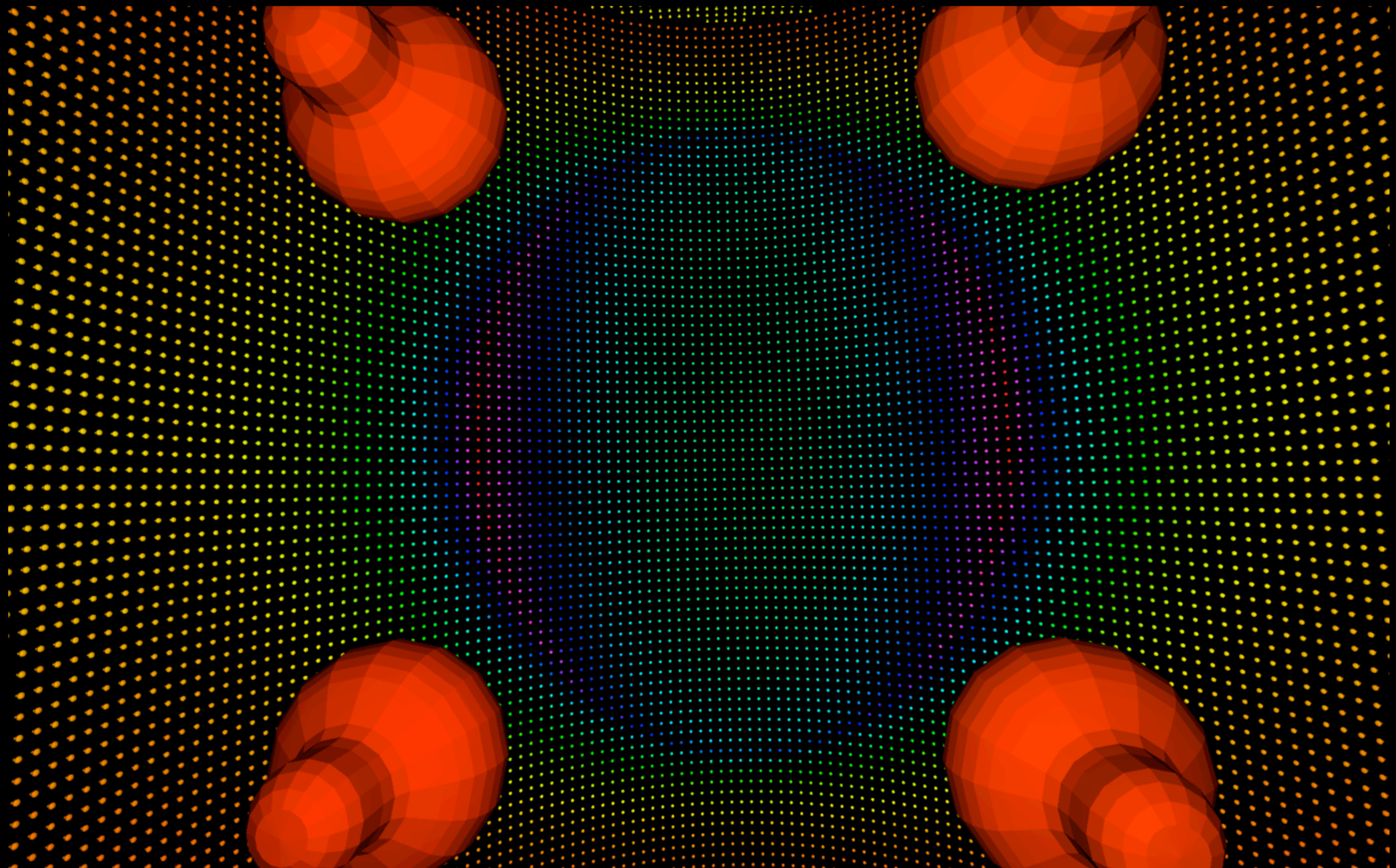




Interactive Geometry Exploration



1 GeV electron: Photon Hit Probability



Why Python?

- Python is a wonderfully productive language.
- Slowly becoming the default "glue" language in physics.
- Relatively easy to interface to existing C++ libraries that are used in particle physics (PyROOT)
- Physicists are not professional programmers, yet we must write small and large programs constantly.
- IMHO, one of the great challenges in particle physics is to reduce the cognitive load of programming. Python helps. (C++ is terrible for this. I am ashamed every time I help a student debug a memory error.)

Isn't Python Slow?

- *Our most valuable resource is programmer time.*
- Only a few programs are performance critical, and they usually have only a few performance critical sections.

- Our Python Performance Strategy:
  - Write simple code.
  - Doing math in a loop? Use numpy instead.
  - Not fast enough? Profile with line_profiler.
  - Move the performance critical section behind an interface that does the calculation on the GPU.
  - *Do not bother to port code to a compiled language to run on the CPU.*
    If it really is the bottleneck, you should move it to the GPU.

PyCUDA

- PyCUDA provides a direct interface to the CUDA toolkit and runtime from Python.
- Relatively thin wrapper around most CUDA functions.
- Compile code at runtime (handy for code generation) or load cubin.
- Provides a gpuarray class which makes it easy to do simple things with arrays on the device and pass them to kernels.
- Integrates well with numpy and numpy arrays.
- Supports the construction of structs with pointer members on the device as well, though this is somewhat tricky.
- http://documen.tician.de/pycuda/

PyCUDA Strategies

- Use a Python class to encapsulate device allocations and CUDA kernels that act on those device allocations.
- Often will result in the creation of a "GPU" mirror for each data structure class. (Ex: Photons and GPUPhotons)
- Whenever possible, represent large data on the CPU with 1D numpy arrays or classes containing 1D numpy arrays. These are easy to conceptually map to equivalent GPU data structures.
- Take advantage of the ability to map host memory into the GPU memory space.
- If you are going to combine PyCUDA with multiprocessing module, *remember to initialize the CUDA device after forking worker processes!*

In Progress Developments

- Allow users of the Chroma module to replace portions of the optical physics with their own model.

Dynamically compiles CUDA source code with device structs patched to include extra fields and with user-supplied physics code inserted into the simulation kernel. (Implemented and being evaluated.)

- Improving the installation experience for new users. Have created a new Python module called “shrinkwrap” to assist with the installation of compiled libraries. Also working on tutorials for new users.

Summary

- Particle physics needs fast photon simulations
- The compute graphics community has developed great techniques and hardware, we just need to use them!
- Chroma was created to explore the application of these techniques to particle physics simulations.
- It works great! Easy to visualize detector designs and realistically simulate photon propagation up to 200x faster than the current tool.
- Python + CUDA provides the best of both worlds. We could have never written Chroma in C++ with such a small team.

http://chroma.bitbucket.org

Special Thanks

- Anthony LaTorre
- Andy Mastbaum
- National Science Foundation
- Department of Energy