# Modular C++ Frameworks for Ray-Propagation

J. C. Hanson (CCAPP, The Ohio State University)

May 12, 2017

CCAPP @ OSU

# Modular C++ Frameworks for Ray-Propagation

I. Collaboration with Dave Seckel (ARA), Robert Lahmann (ARIANNA)
II. Problem Setup
III. The Code
IV. Some Graphs

# Modular C++ Frameworks for Ray-Propagation

Challenge: Speed up ray-tracing, while incorporating what we know about horizontal propagation.

I. There are several approaches
   A. Use MATLAB to code up simple models to understand the physics
   B. AraSim RayTrace.h: 900 lines of code using Boost C++, index fits only
   C. A simple set of interdependent C++ classes (modular approach)

II. Modular approach allows addition of new physics
   A. Index data versus depth
   B. Diffuse scattering vs. specular
   C. Reflective layers

# Modular C++ Frameworks for Ray-Propagation, Ice, Reflections, RF emitters

Problem setup: (dots refer to $d/dz$, $\alpha$ with respect to horizontal)

$$n(z)\cos(\alpha(z)) = const \tag{1}$$

$$c_0 = nc \tag{2}$$

$$\dot{\alpha} = \frac{\cos\alpha}{\sin\alpha}\frac{\dot{n}}{n} \tag{3}$$

$$dz = cdt\sin\alpha \tag{4}$$

$$\frac{d\alpha}{dt} = \cos\alpha\frac{c_0}{n^2}\frac{dn}{dz} \tag{5}$$

Knowing how the angle depends on time, we know how to increment it, and (x,z). Also, Robert Lahmann has shown how this approach falls out of my Fermat's Principle approach, which is informative.

# The code

clone from github.com:
`https://github.com/918particle/Ray_Propagation`

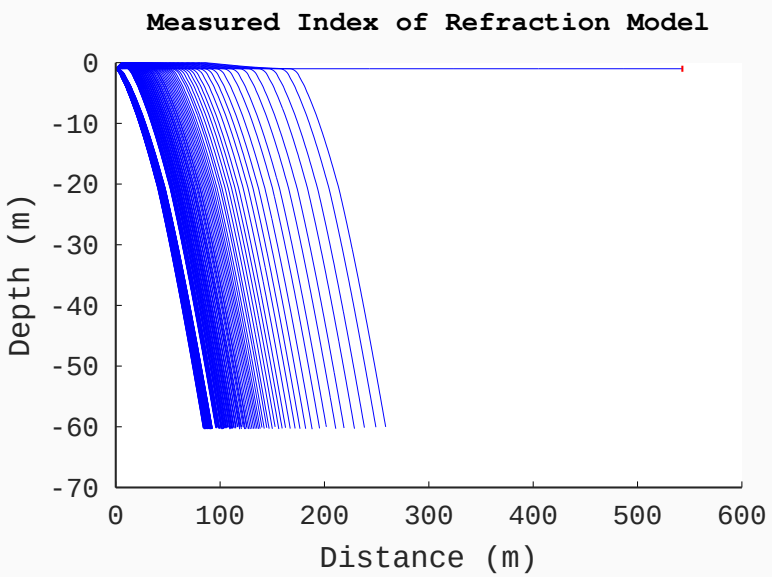
- Minimal octave example:
  `Ray_Propagation/code/RayProp_jch.m`
- With diffuse scattering and reflective surfaces: coming soon
- C++ modules:
  `Ray_Propagation/code/cpp_propagation`

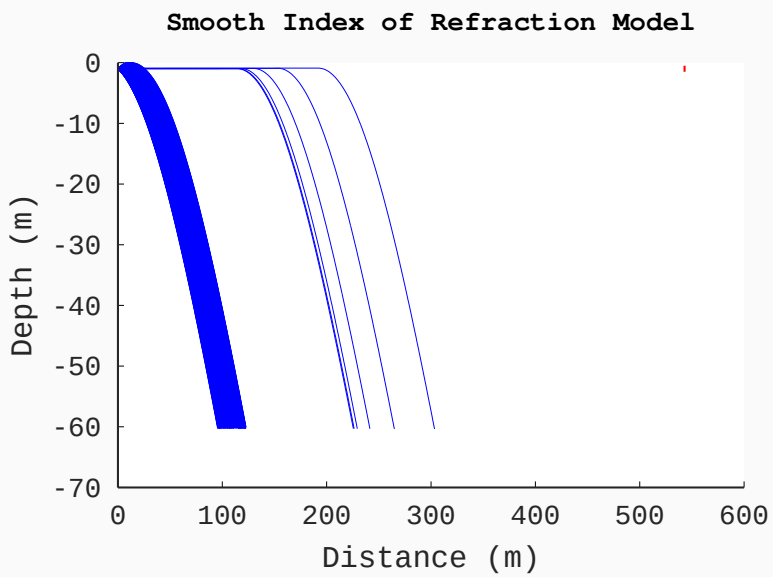
# The code

The C++ module has four classes:

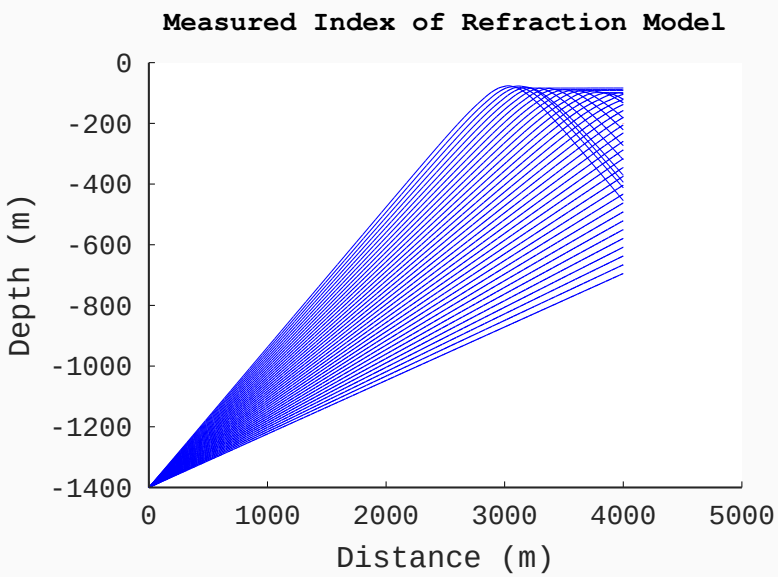
- Emitter.h: Handles the position and angle of the ray within (x,z) coordinate system
- Ice.h: Handles index of refraction profile, attenuation length profile, cylindrical coordinates
- Reflector.h: Defines depths at which reflectors exist (will get this from Ice.h soon)
- Propagator.h: Handles propagation using differential approach, obtains $n(z)$ from Ice.h

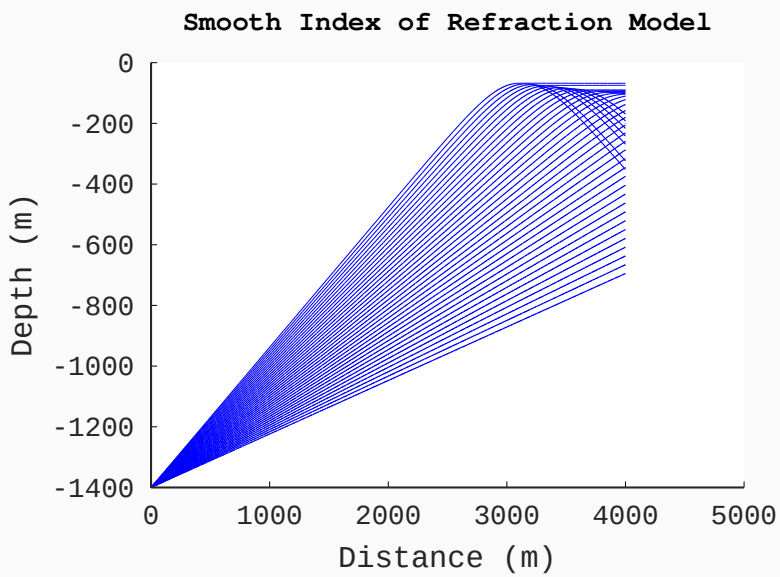
# Octave implementation (no reflections or diffuse scattering)



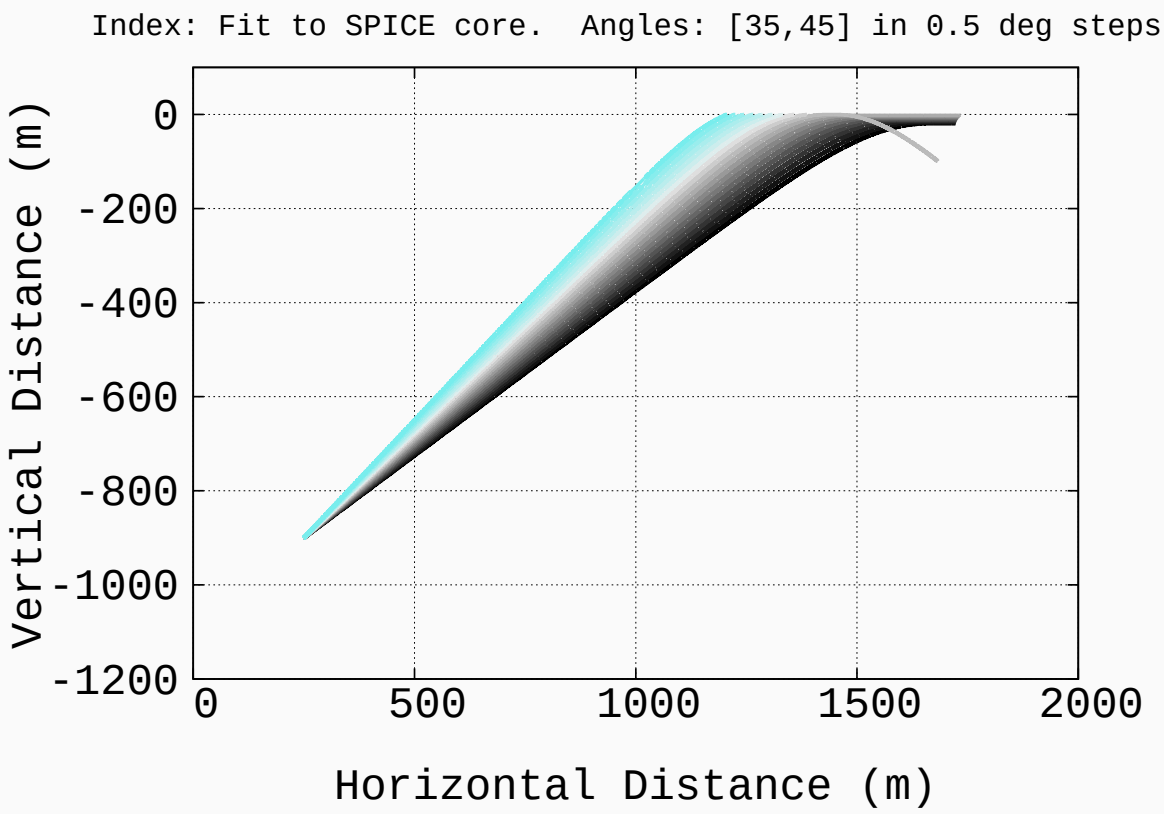



# Octave implementation (no reflections or diffuse scattering)



# C++ modular implementation (no reflections or diffuse scattering)



# Coming soon: Spoorti

Challenge: Speed up ray-tracing, while incorporating what we know about horizontal propagation.

I. There are several approaches
   A. Use MATLAB to code up simple models to understand the physics
   B. AraSim RayTrace.h: 900 lines of code using Boost C++, index fits only
   C. A simple set of interdependent C++ classes (modular approach)

II. Modular approach allows addition of new physics
   A. Index data versus depth
   B. Diffuse scattering vs. specular
   C. Reflective layers