INICEMC MEETING - C++ MODULE FOR RF PROPAGATION THROUGH ICE AND FIRN

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

May 19, 2017

CCAPP @ OSU

OUTLINE

I. A C++ Module for RF propagation in ice - Why?
   A. Class structure and functions
   B. How Propagator.h works

II. Physics questions
   A. Measured firn profiles and channeling
   B. Reaching the surface
   C. Air to firn propagation (new)
   D. RFRay.h distance and loss tracking (new)

III. What's next?
   A. Diffuse reflection (Geoffrey)
   B. Verify with Mathematica (Spoorti)
   C. Channelling with no explicit reflection layer

THE CODE

CLASS STRUCTURE AND FUNCTIONS - SEE GITHUB, 918PARTICLE, RAY PROPAGATION

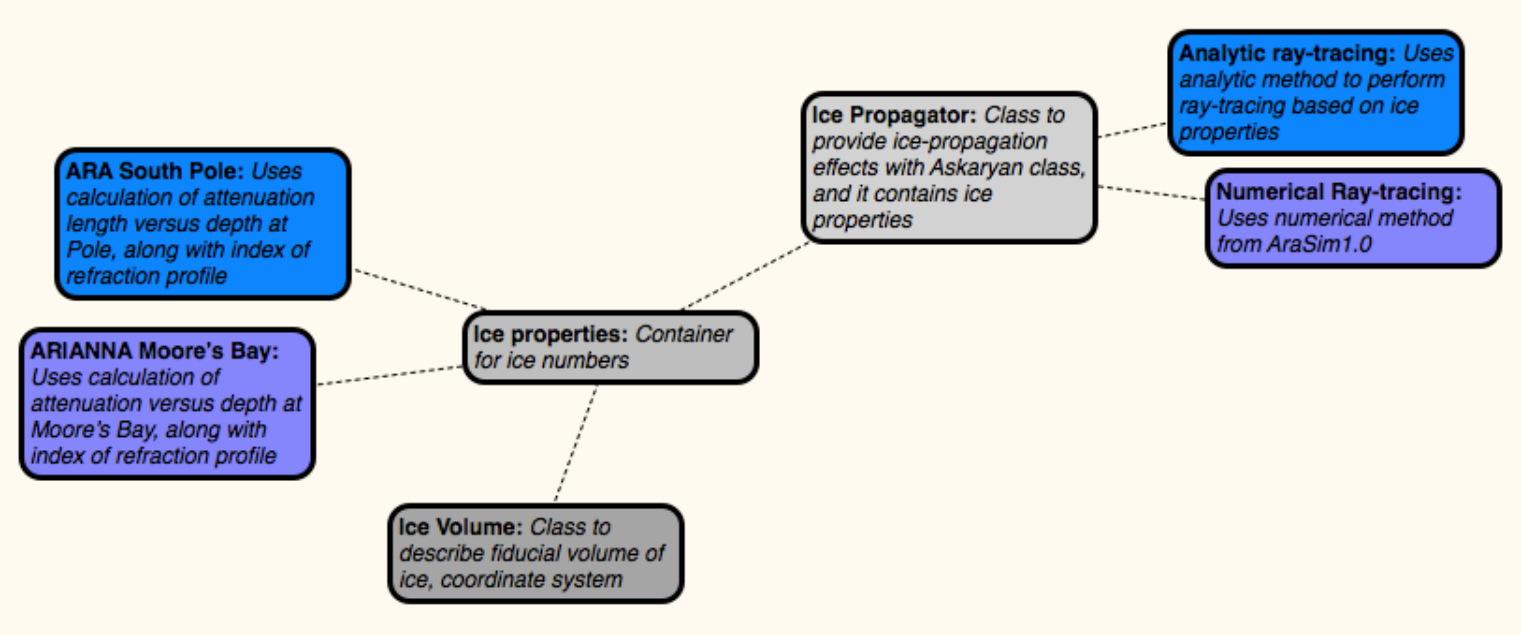

Figure 1: The original RF propagation class structure from AraSim2 outline.

CLASS STRUCTURE AND FUNCTIONS

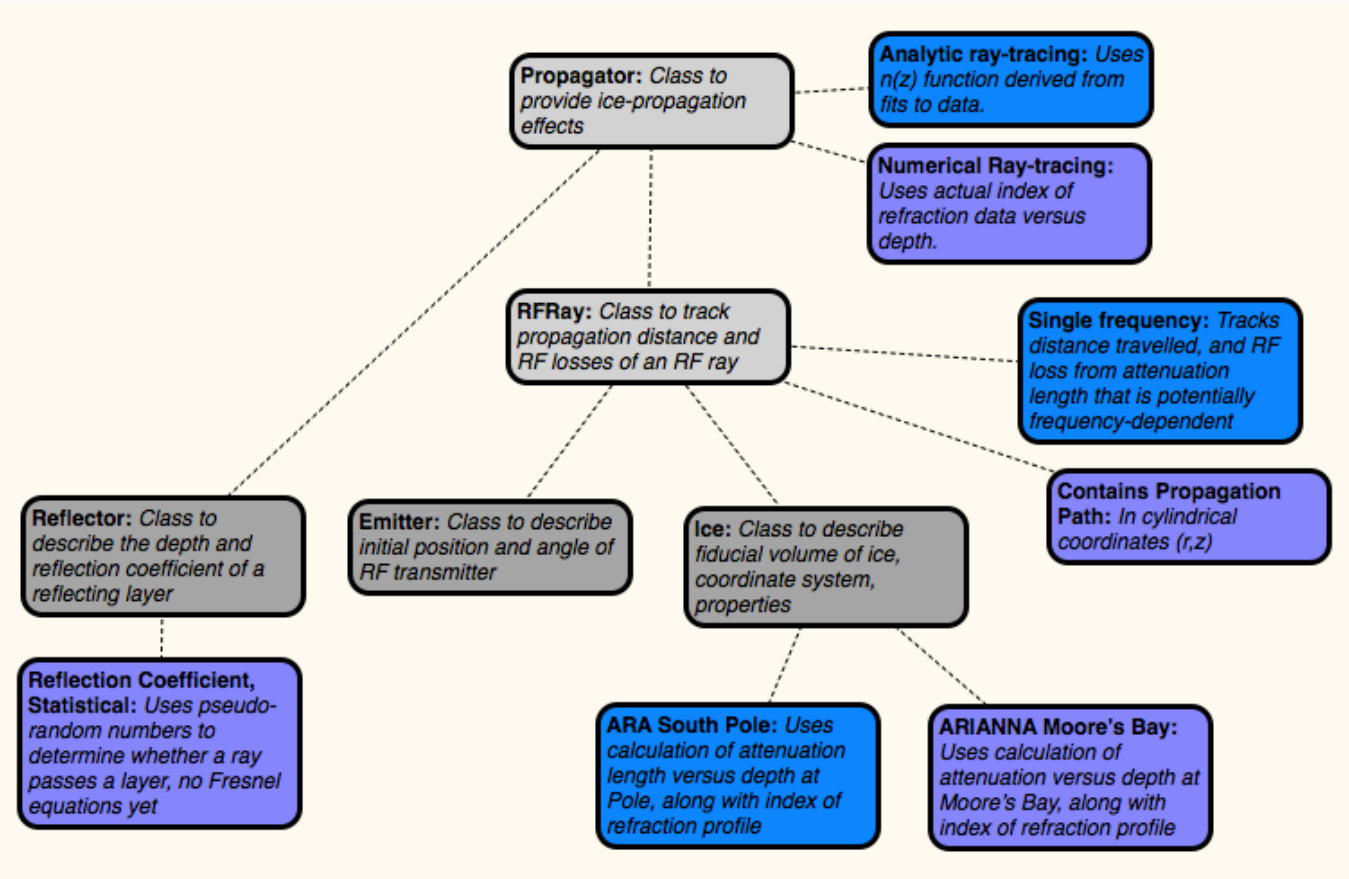

Figure 2: The current RF propagation class structure.

HOW PROPAGATOR.H WORKS

```
class Propagator : public Reflector, public RFRay
{
public:
float _globalTime; //nanoseconds
float _timeStep; //nanoseconds
std::pair<float,float> _currentPosition;
bool _isInitialized;
...
void InitializePropagator(float,...,float);
void AddReflector(float,float);
void Propagate(); //Propagate ray through medium
void ReadoutPath(std::string);
};
```

How Propagator.h works - check for reflections (issue no. 1)

```
void Propagator::Propagate()
{
float c0 = 0.299792458; //speed of light in vacuum, meters per nanosecond
float dz = 1.0e-4; //units: meters
float dndz = 0.0; //units: meters^(-1)
float theTime = 0.0;
this->_path.push_back(_emitterPosition);
bool flag = true;
while(theTime<_globalTime)
{
 float n = GetIndex(_emitterPosition.second);
 theTime+=_timeStep;
 std::pair<float,float> old_pos = _emitterPosition;
 _emitterPosition.first+=cos(_initialAngle)*_timeStep*c0/n;
 _emitterPosition.second+=sin(_initialAngle)*_timeStep*c0/n;
 this->_path.push_back(_emitterPosition);
 CheckForAReflection(_initialAngle,_emitterPosition.second);
...
```

HOW PROPAGATOR.H WORKS - EVALUATE dn/dz (ISSUE NO. 2)

```
...
 if(std::abs(old_pos.second-_emitterPosition.second)>dz)
 {
  dndz = (GetIndex(_emitterPosition.second)-GetIndex(old_pos.second))
   /(_emitterPosition.second-old_pos.second);
  }
  else
  {
   dndz = (GetIndex(_emitterPosition.second)-GetIndex(_emitterPosition.second-dz))
   /(_emitterPosition.second-dz);
  }
  float dTheta = _timeStep*cos(_initialAngle)*dndz*c0/(n*n);
  std::cout<<dTheta*180.0/3.14159<<" ";
  if(dTheta>=3.14159/2.0 && dTheta<3.14159) dTheta-=3.14159/2.0;
  else if(dTheta>=3.14159) dTheta-=3.14159;
  _initialAngle+=dTheta;
  this->_currentAngle = _initialAngle; //Change this after today, May 19th, 2017.
  }
}
```

PHYSICS QUESTIONS

MEASURED FIRN PROFILES AND CHANNELLING

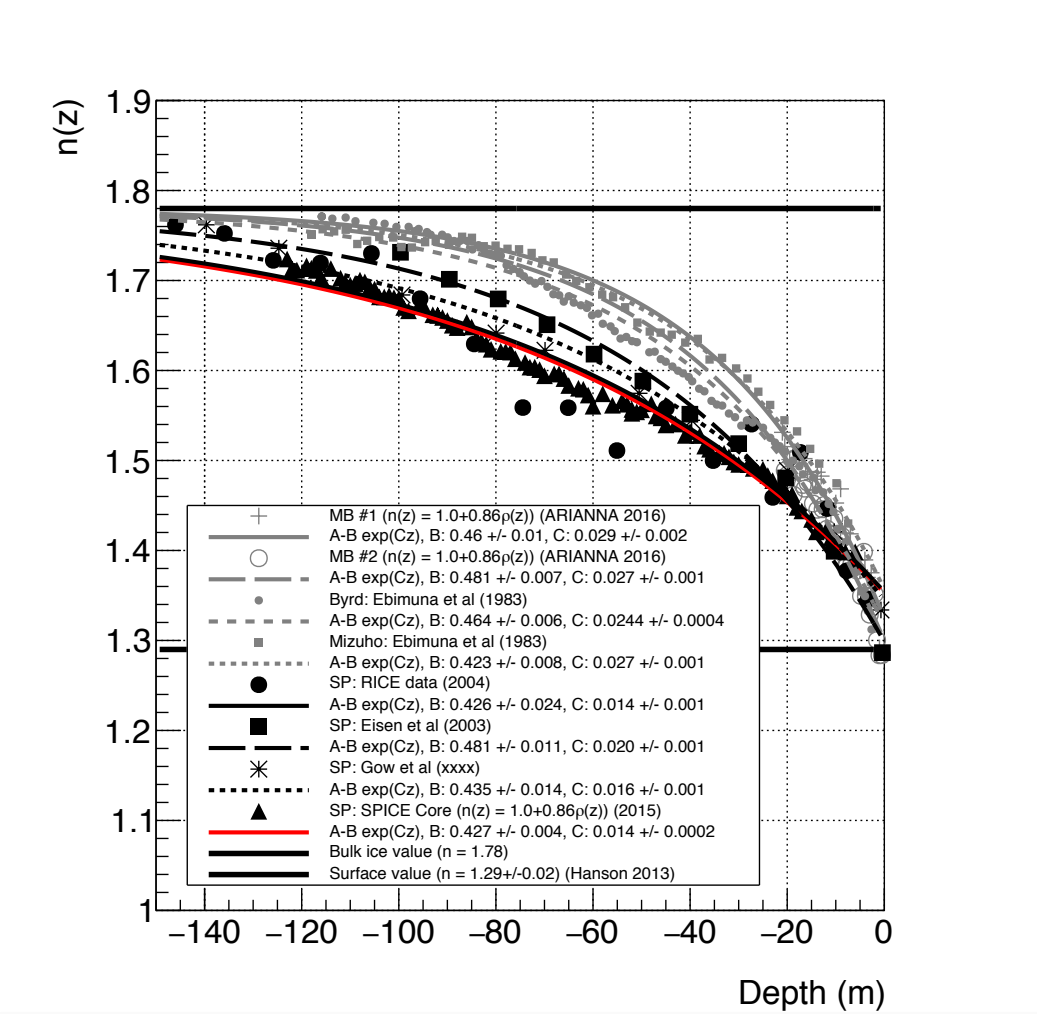

Figure 3: Summary of index of refraction data and fits.

```
else if(modelName==”Gow”)
{
_A = 1.78;
_B = 0.435;
_C = 0.016;
std::ifstream in(”/home/jordan/ANewHope/Gow_withOnePlus8
float depth,index;
while(in.good() && ~in.eof())
{
in>>depth;
in>>index;
_indexVsDepth.push_back(std::pair<float,float>(depth,ind
}
in.close();
}
```

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

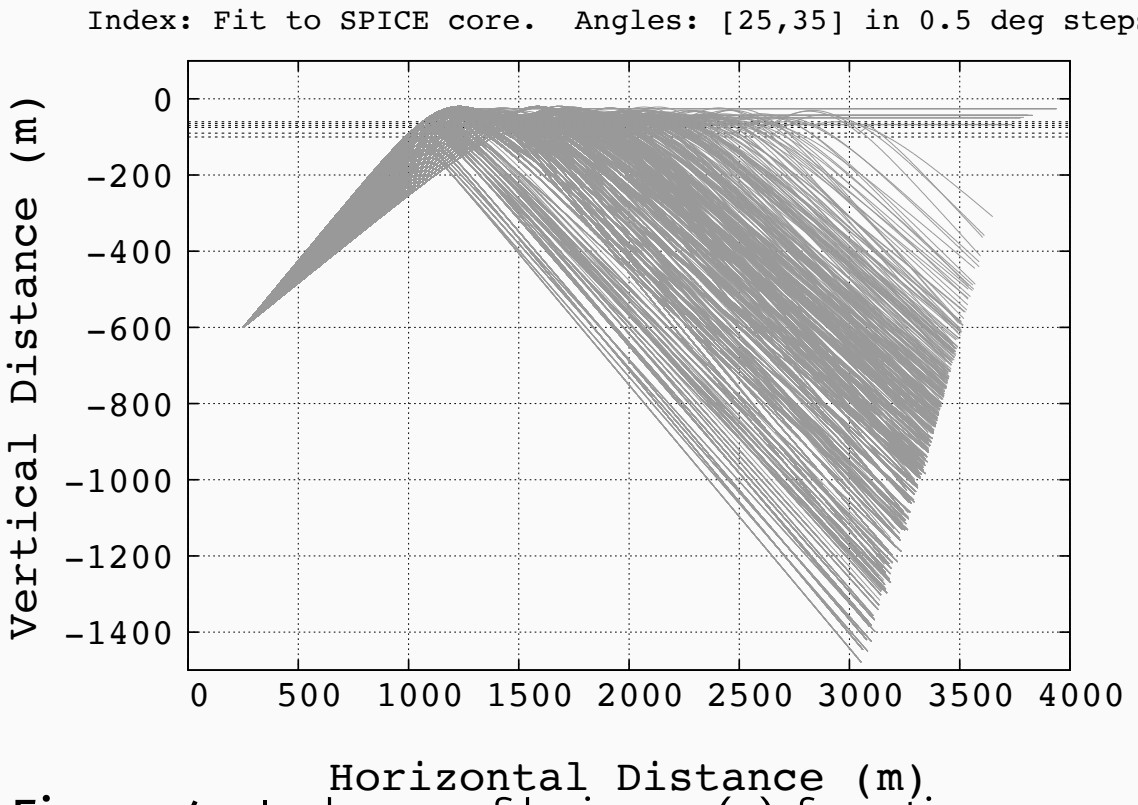

Figure 4: Index profile is a $n(z)$ function.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

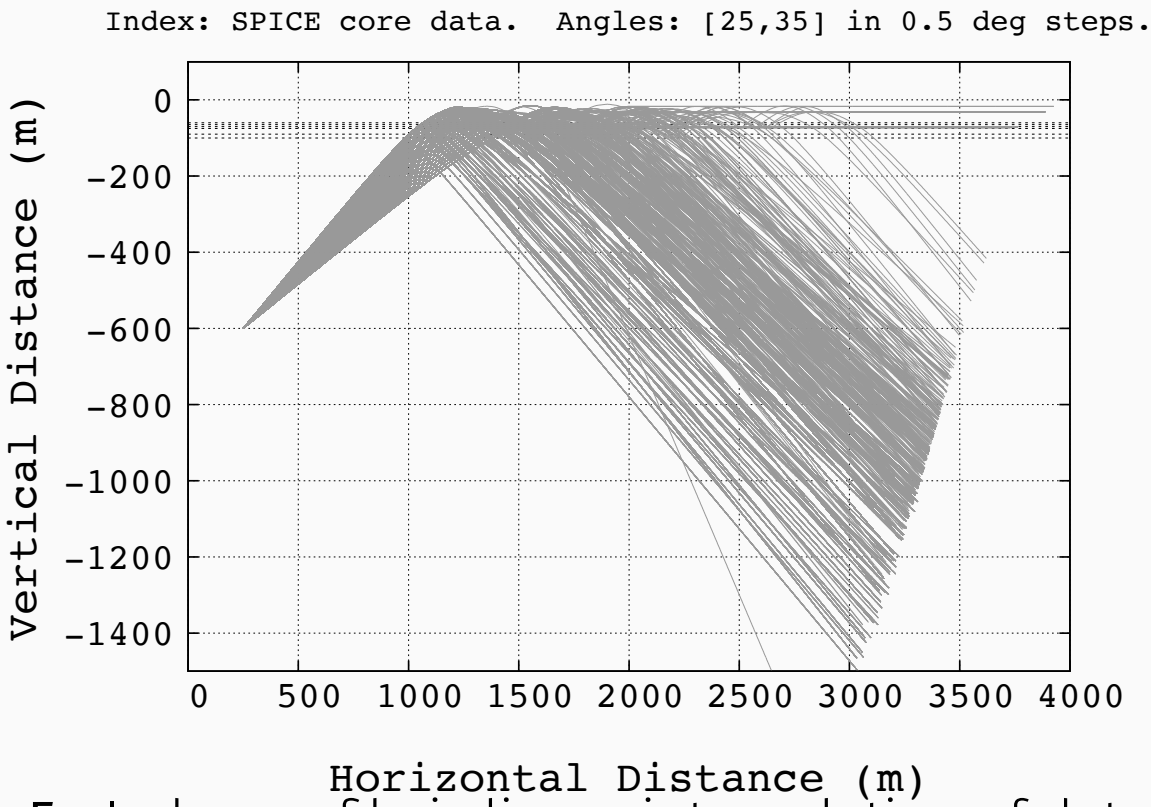

Figure 5: Index profile is linear interpolation of data.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

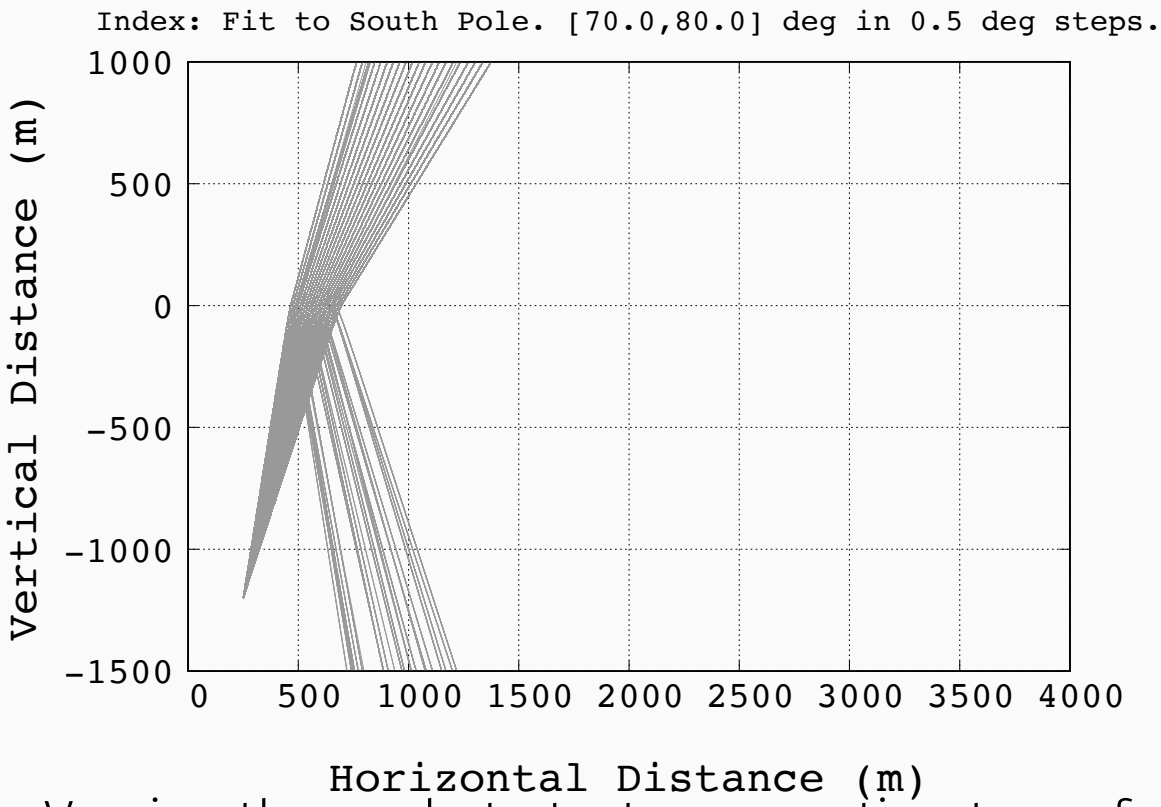

Figure 6: Varying the angle to test propagation to surface.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

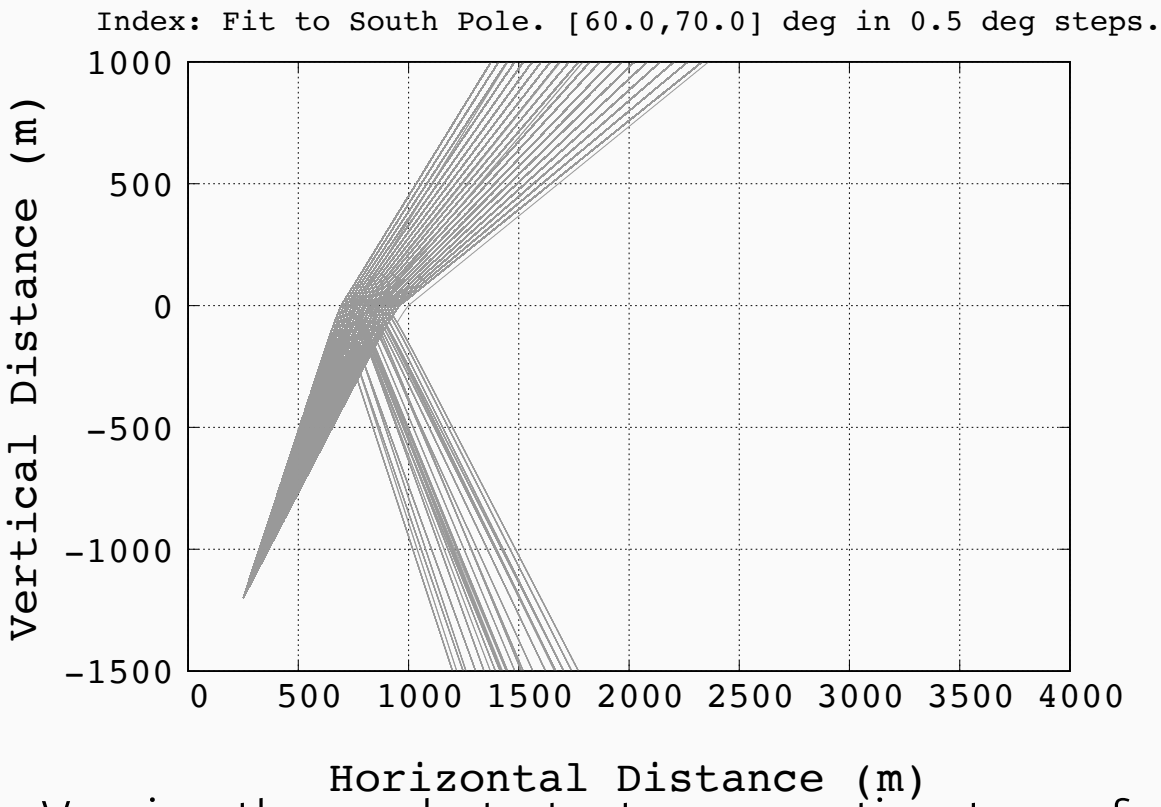

Figure 7: Varying the angle to test propagation to surface.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

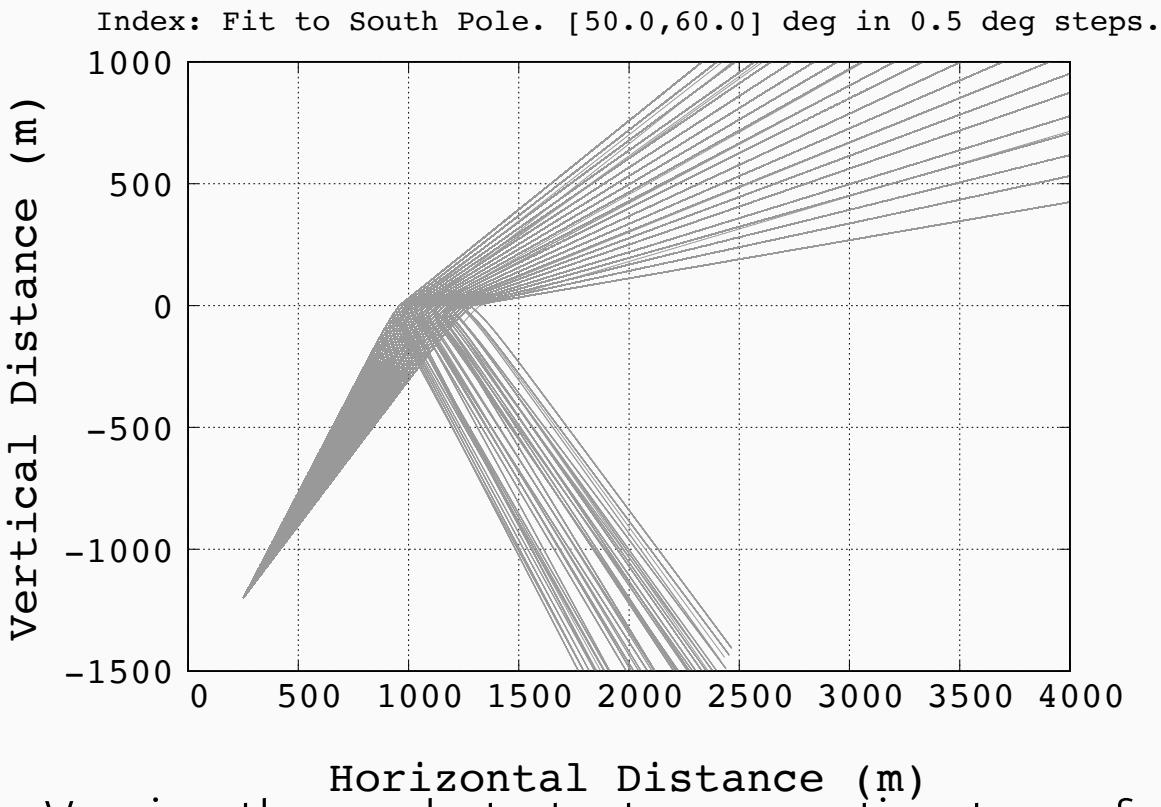

Figure 8: Varying the angle to test propagation to surface.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

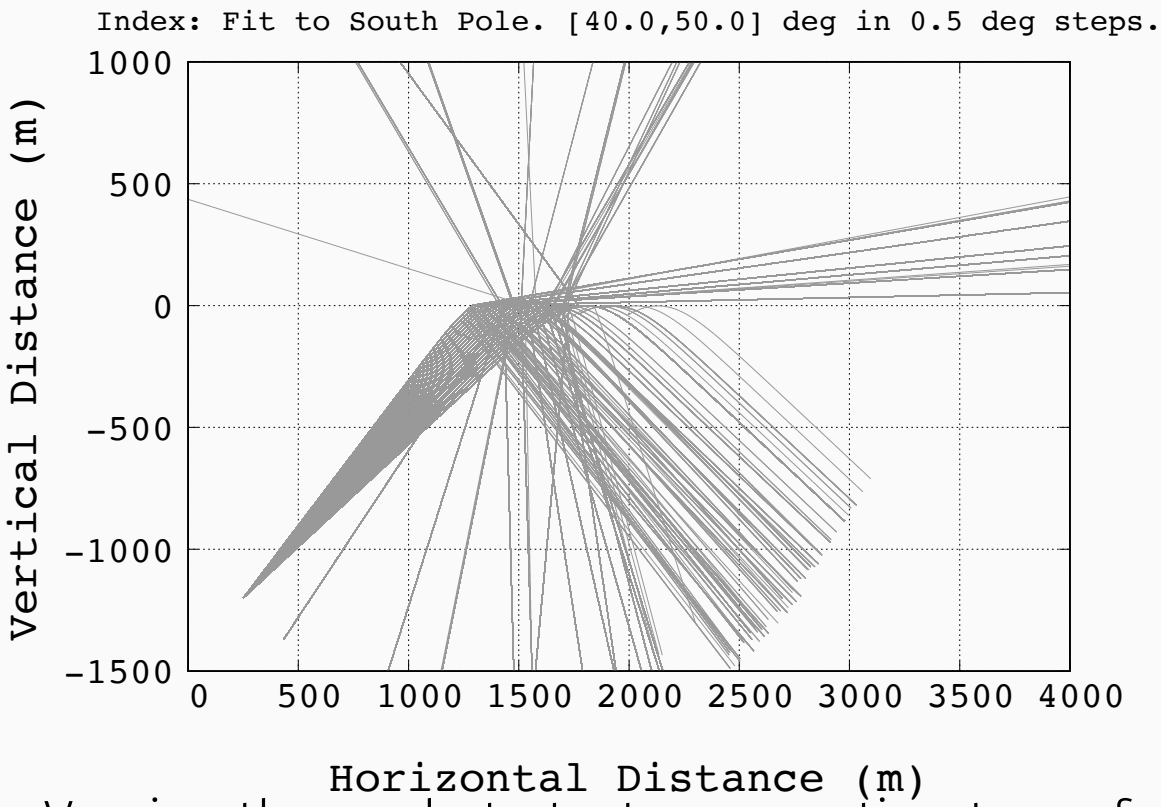

Figure 9: Varying the angle to test propagation to surface.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

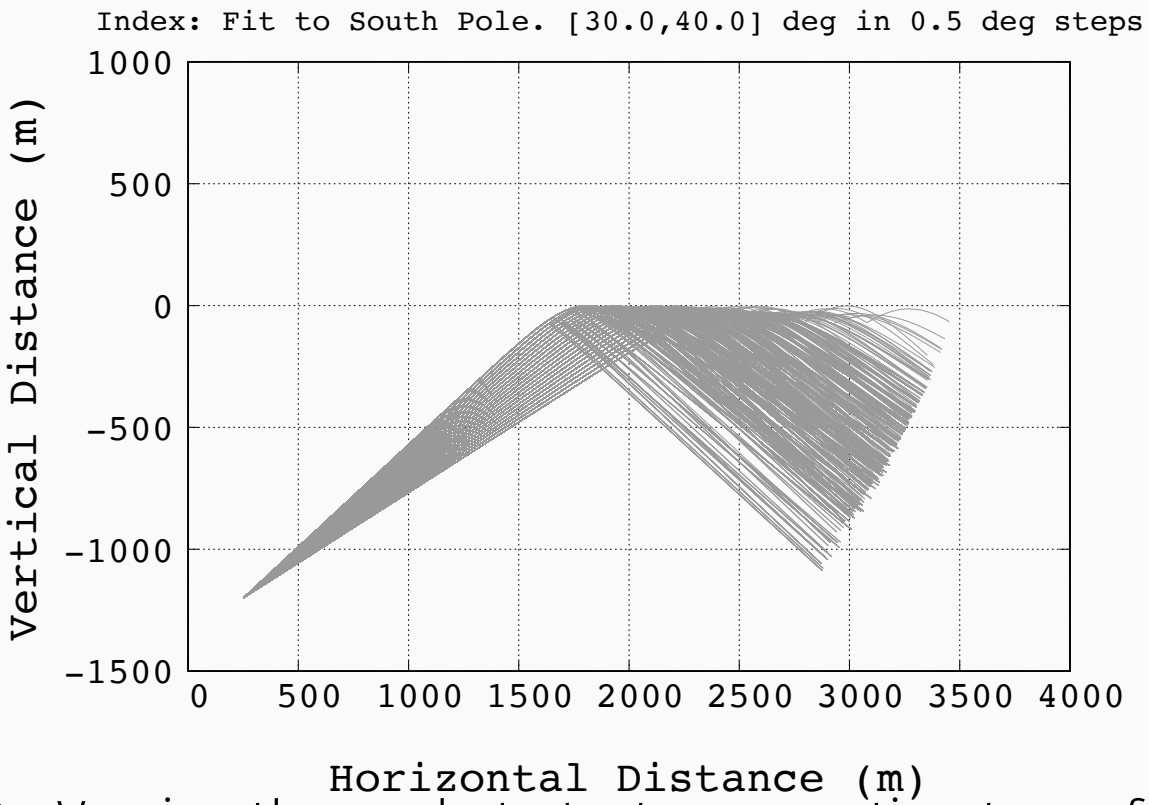

Figure 10: Varying the angle to test propagation to surface.

REACHING THE SURFACE WITH REFRACTION AND CHANELLING

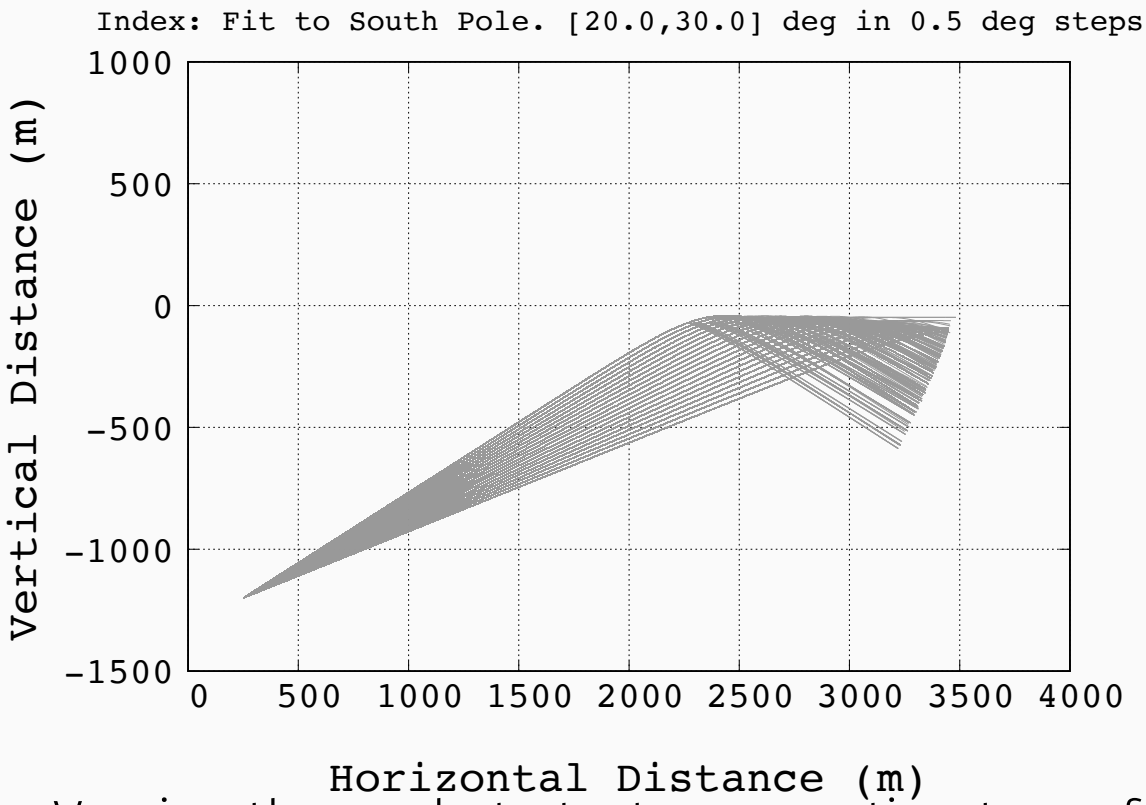

Figure 11: Varying the angle to test propagation to surface.

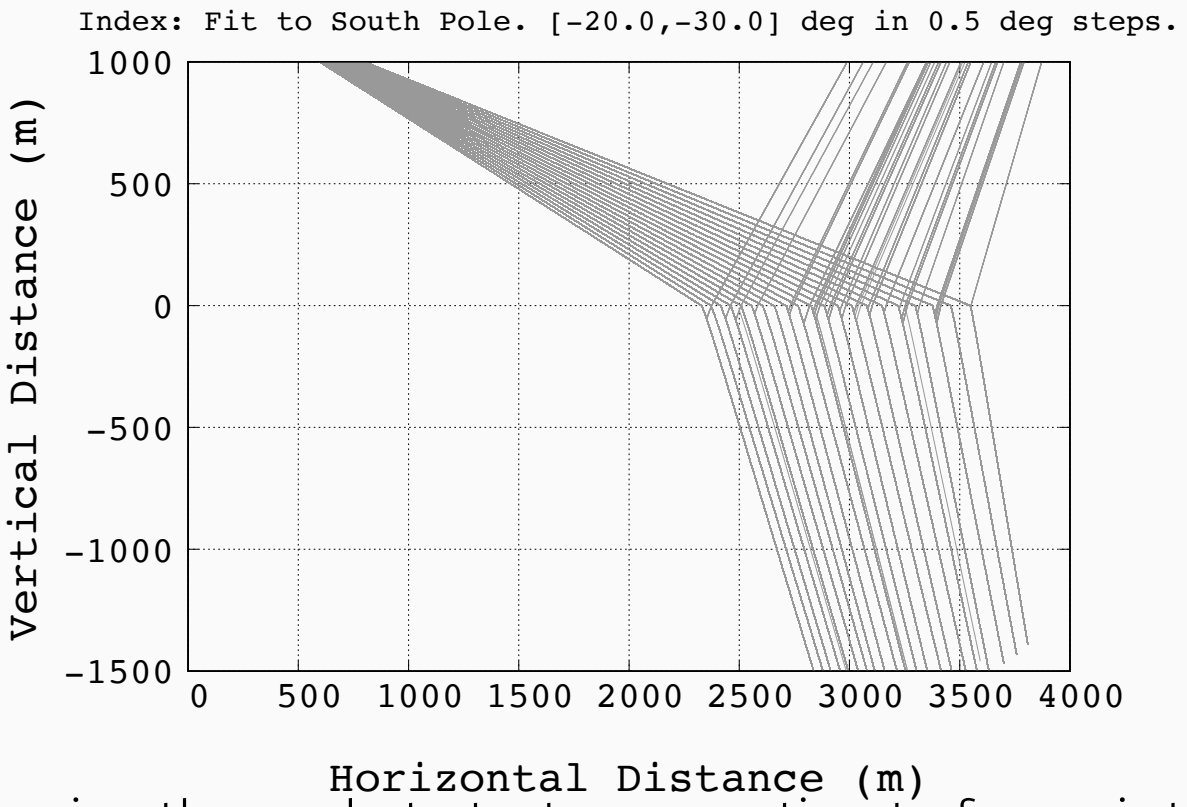

Figure 12: Varying the angle to test propagation to from air to firn.

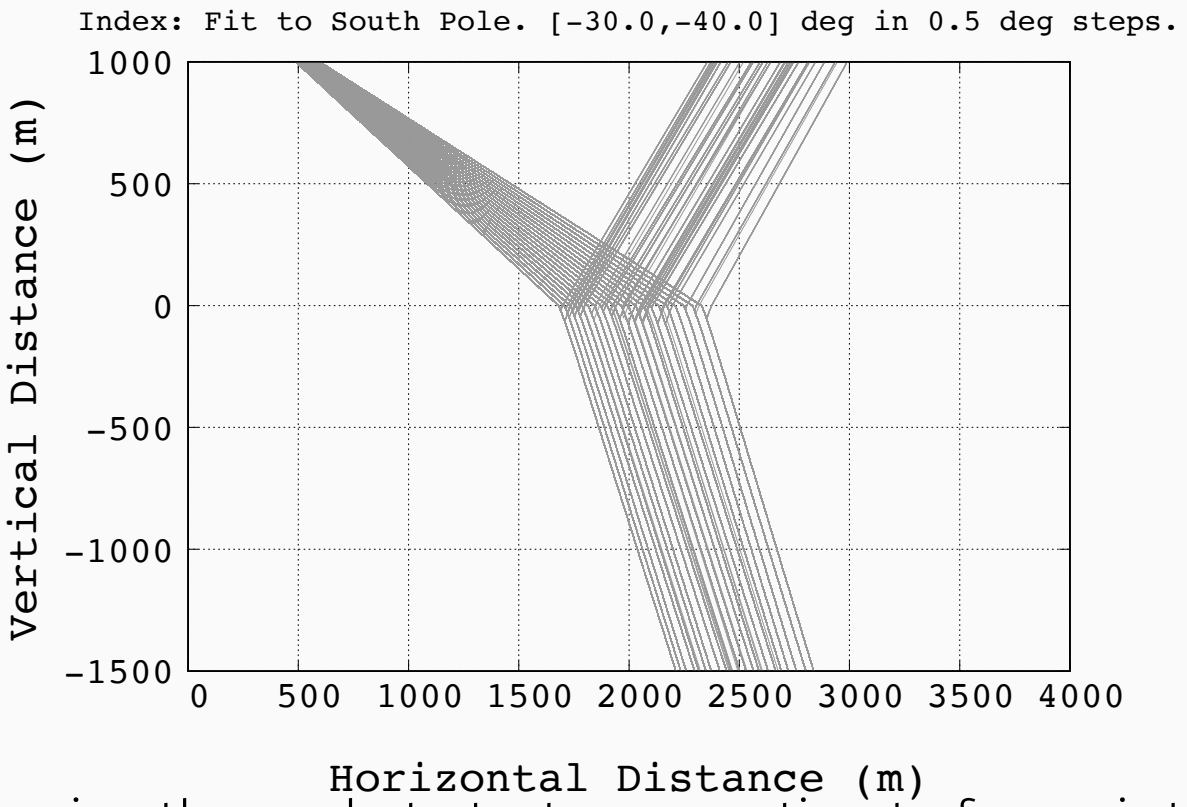

Figure 13: Varying the angle to test propagation to from air to firn.

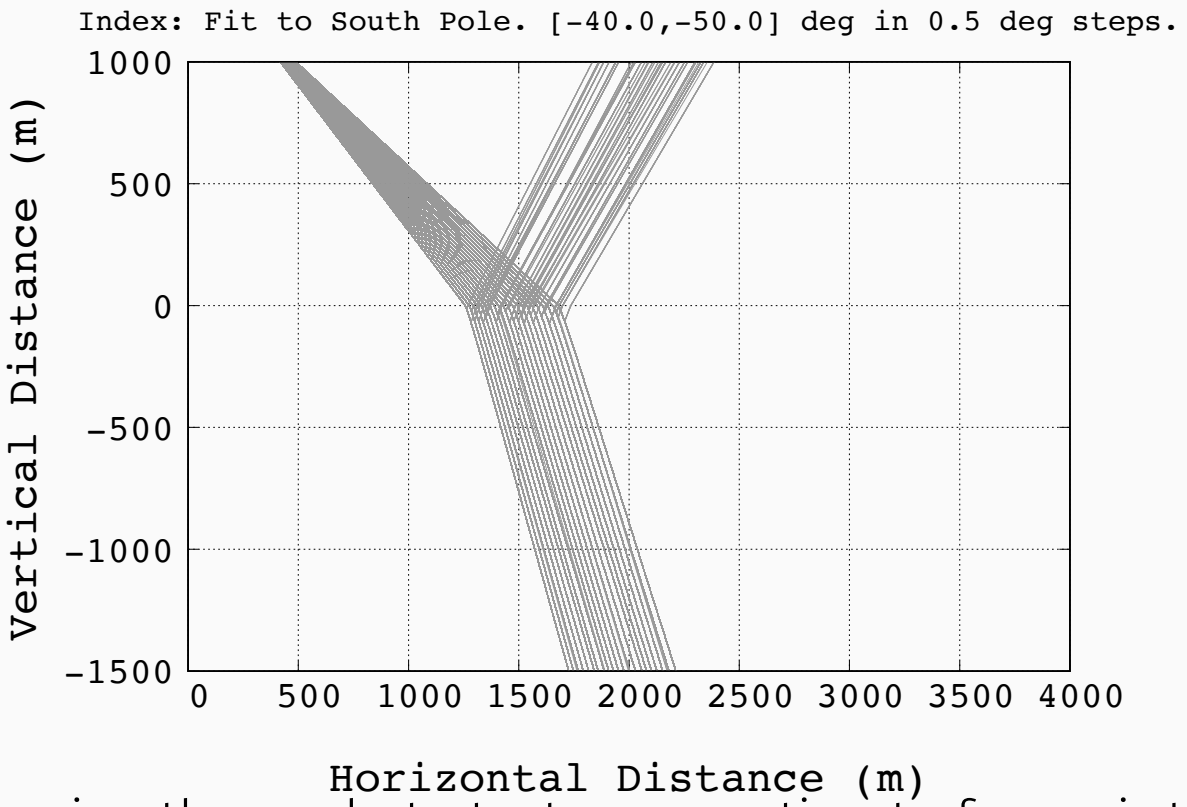

Figure 14: Varying the angle to test propagation to from air to firn.

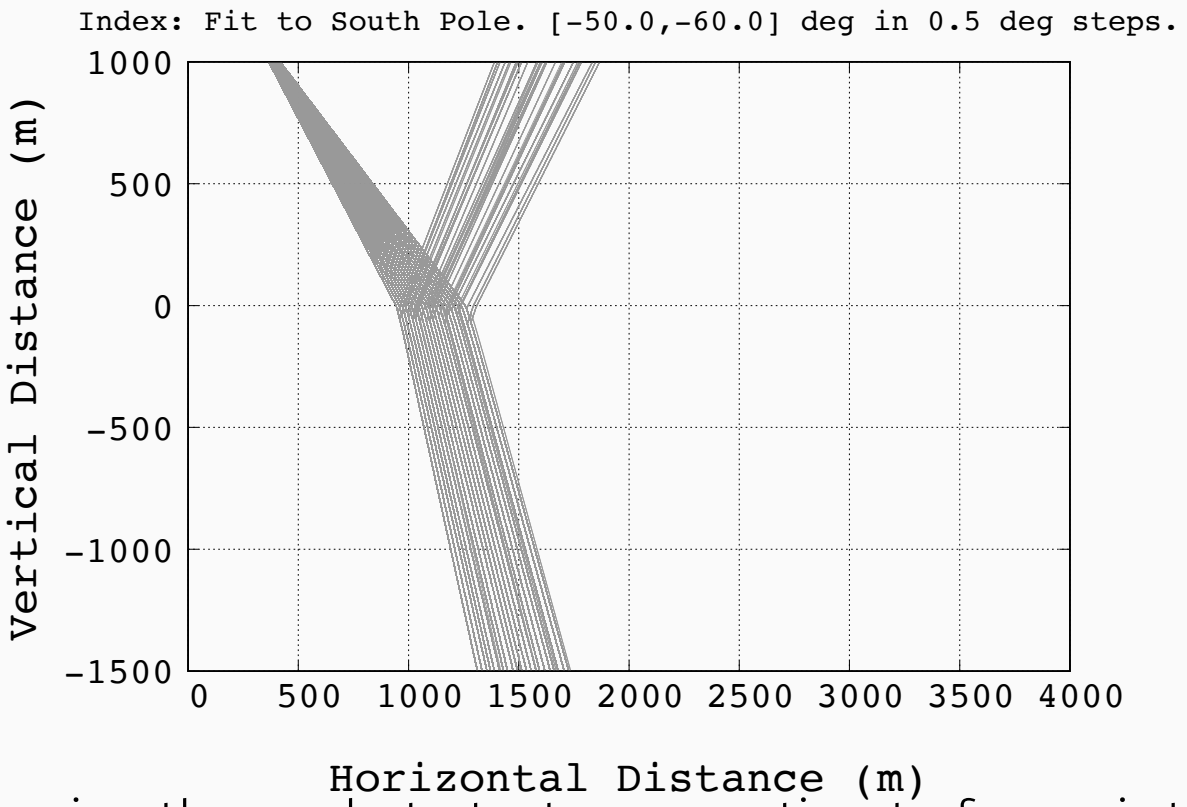

Figure 15: Varying the angle to test propagation to from air to firn.

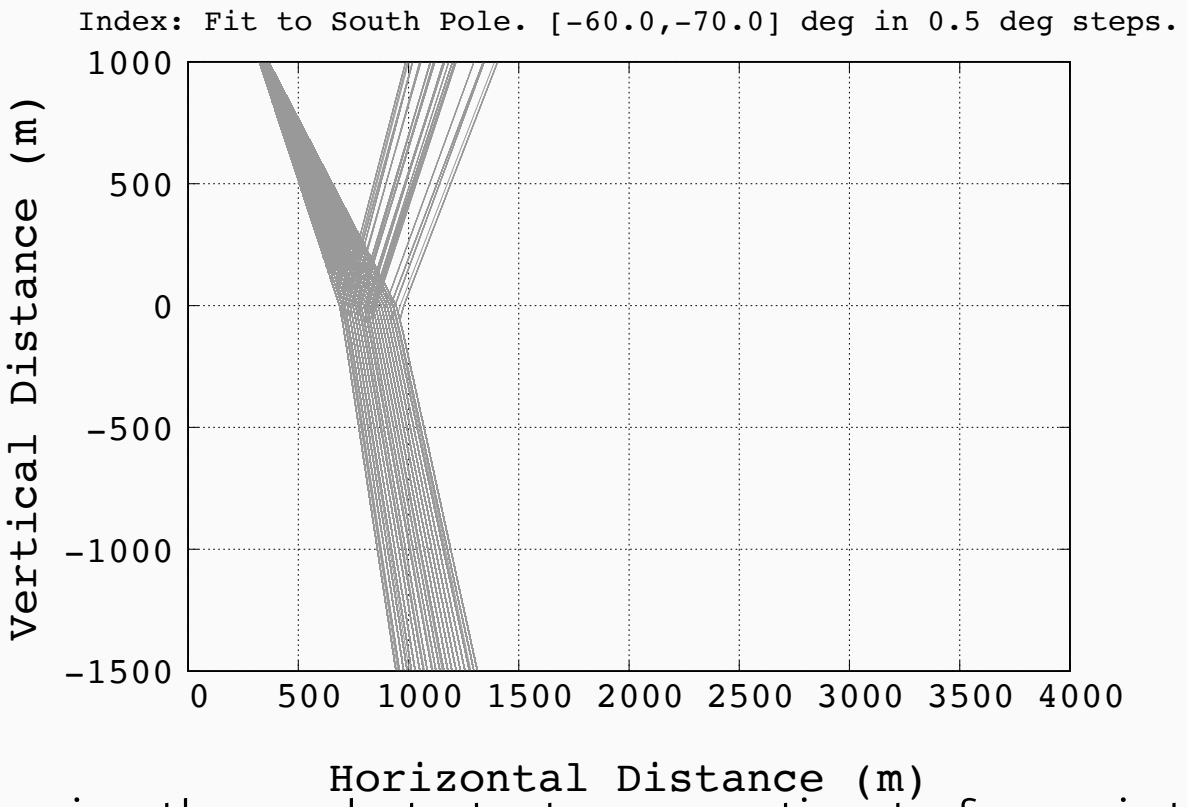

Figure 16: Varying the angle to test propagation to from air to firn.

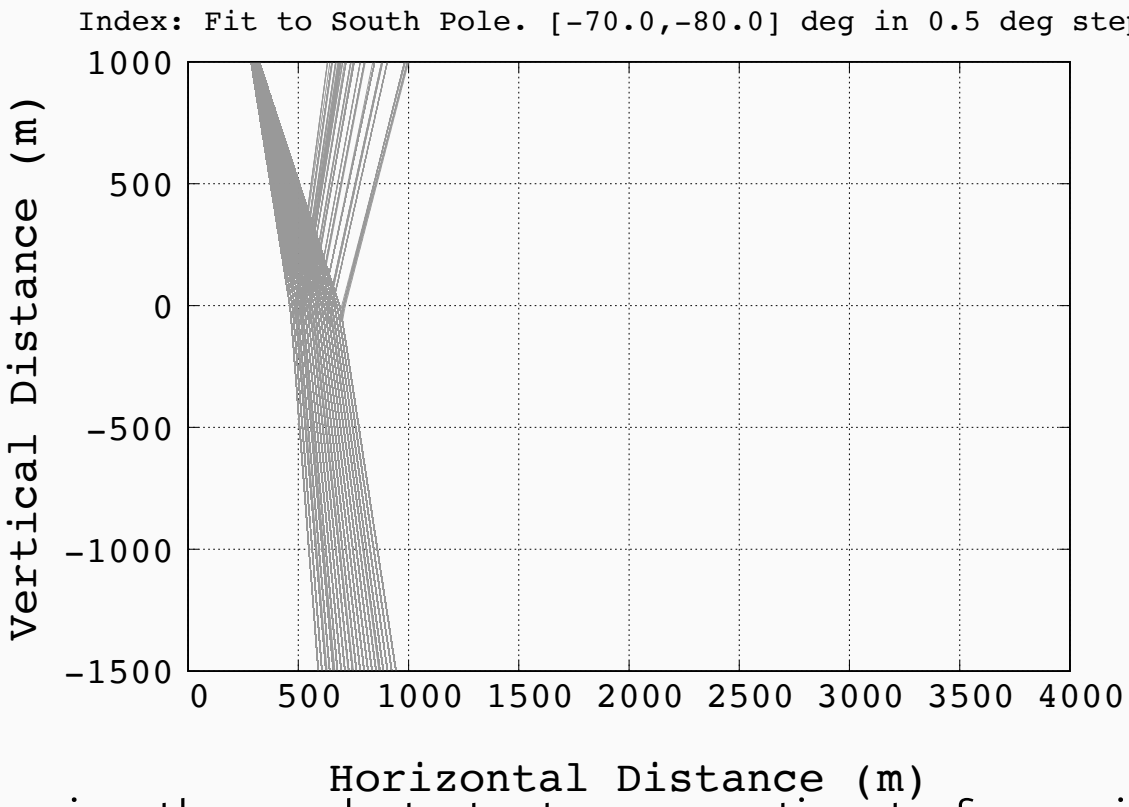

Figure 17: Varying the angle to test propagation to from air to firn.

```
class RFRay : public Emitter, public Ice
{
public:
RFRay() : _distanceTravelled(0.0), _rfLoss(0.0), _freq(0
void SetFreq(float);
void Update(std::pair<float,float>,float);
std::pair<float,float> _currentPosition; //Current posit
float _currentAngle; //Current angle with respect to hor
std::pair<float,float> _priorPosition; //Current positio
float _distanceTravelled; //Keeps track of the propagati
float _rfLoss; //Amount of attenuation (apart from dista
float _freq; //Frequency of the ray
std::vector<std::pair<float,float> > _path;
};
```

OUTLINE

I. A C++ Module for RF propagation in ice - Why?
   A. Class structure and functions
   B. How Propagator.h works

II. Physics questions
   A. Measured firn profiles and channeling
   B. Reaching the surface
   C. Air to firn propagation (new)
   D. RFRay.h distance and loss tracking (new)

III. **What's next?**
   A. **Diffuse reflection (Geoffrey)**
   B. **Verify with Mathematica (Spoorti)**
   C. **Channelling with no explicit reflection layer**