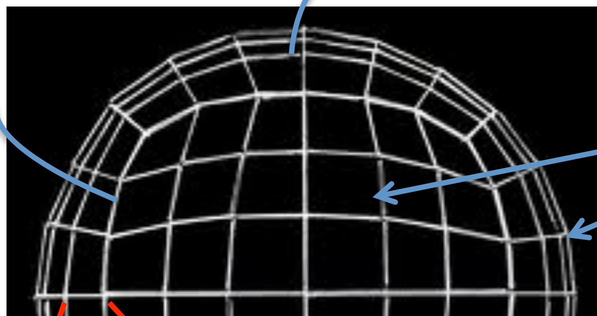
Spatial Interpolation on a Sphere Tiled with Quadrilateral Cells

Kevin Raeder[1], Jeffrey L. Anderson[1]

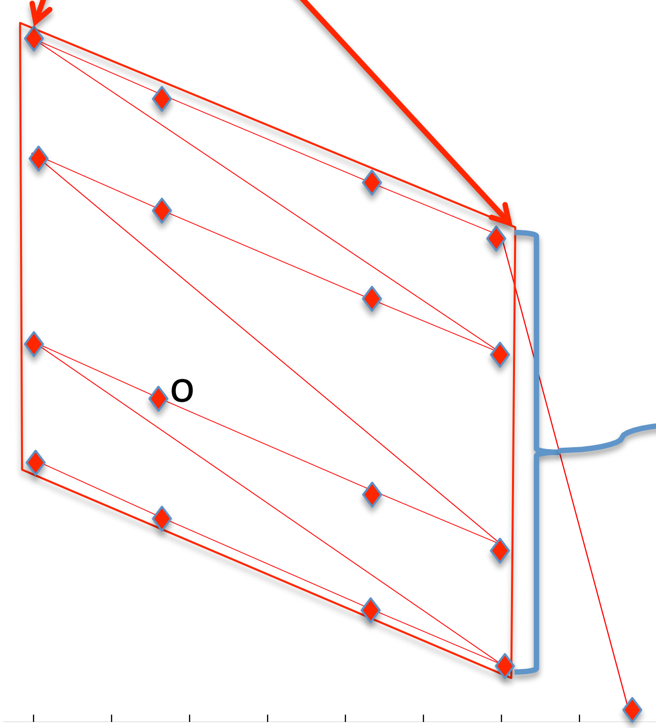
[1] NCAR/CISL/IMAGe

Cubed-Sphere Example:



6 ‘faces’ cover the sphere

ne ‘elements’ per face edge (6 here)



Each element is composed of a 4x4 grid of quadrature nodes (“corners”), where the model fields are defined.

Equations of motion are solved on all the grid points (‘nodes’) in an element at the same time.
Edge nodes are shared with adjacent elements for this solution, but are not redundant in the initial file.

‘np’ nodes in each direction are not evenly spaced.

3 of the 4 closest nodes to an ob may be outside the cell containing the ob.

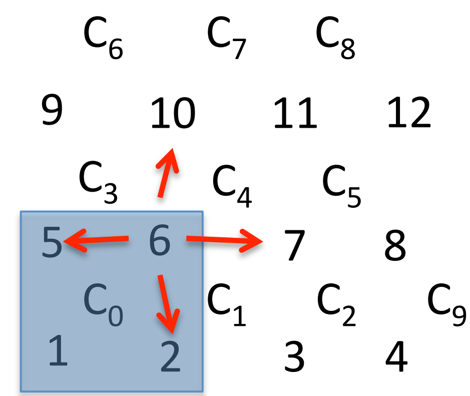
Global set of nodes is arranged as a 1D array.
Adjacent in array ≠ adjacent on sphere

HommeMapping.nc contains neighboring node information

But it needs to be turned 'inside out'.
It will give the names of 4 nodes around a 'center', aka 4 nodes defining a quad.

ncorners[0] ncenters[0] element_corners[0]=1
ncorners[1] ncenters[0] element_corners[48600]=5
ncorners[2] ncenters[0] element_corners[97200]=6
ncorners[3] ncenters[0] element_corners[145800]=2

ncorners[0] ncenters[1] element_corners[1]=2
ncorners[1] ncenters[1] element_corners[48601]=6
ncorners[2] ncenters[1] element_corners[97201]=7
ncorners[3] ncenters[1] element_corners[145801]=3



But we want the names of the neighbors of a given node

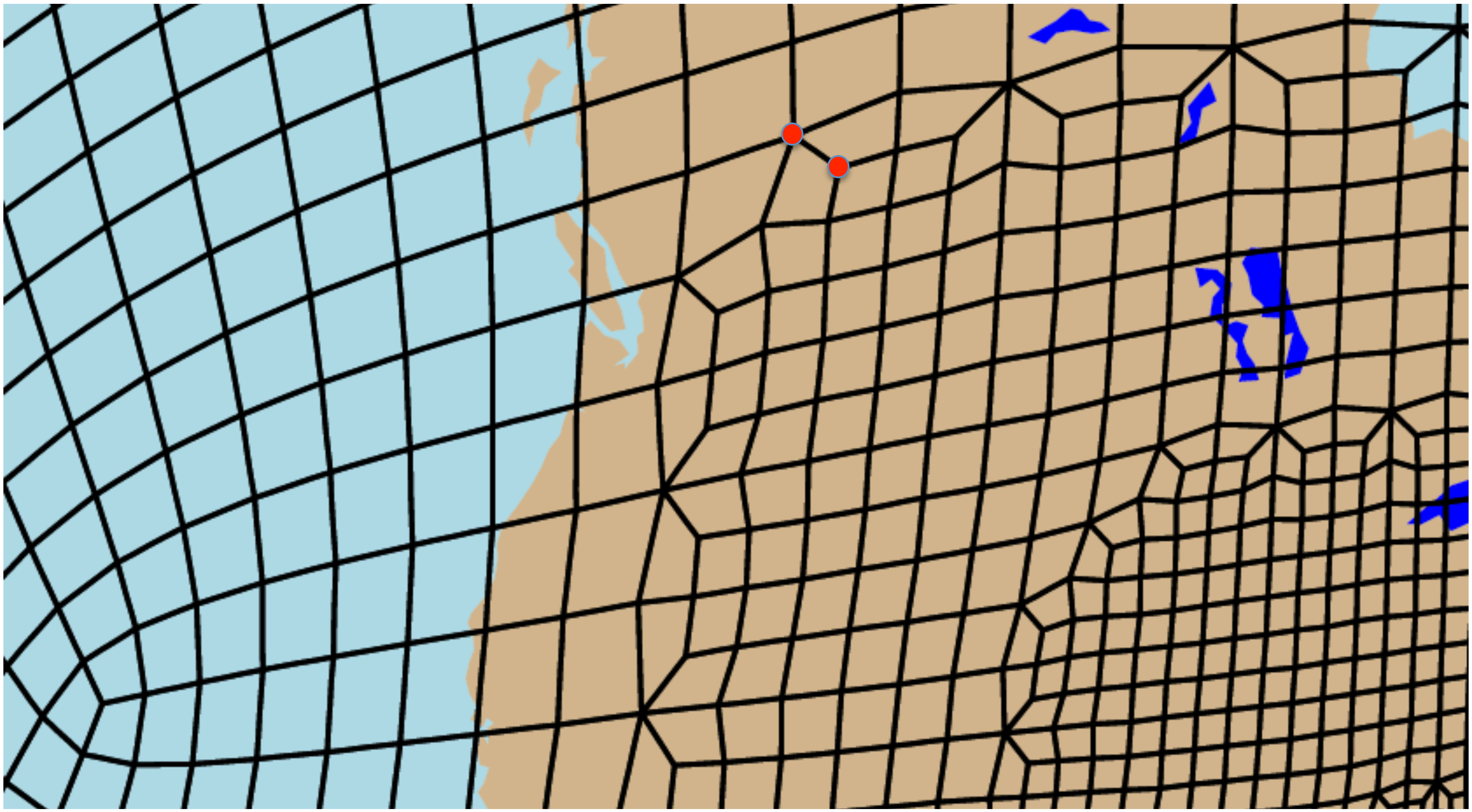
Node '6' has 4 neighbors: 7,2,5,10

Store cells/centers associated with each node in a file before any assimilation.

Lats and lons of nodes are given in the CAM initial file.
Same node labeling as in HommeMapping.nc.

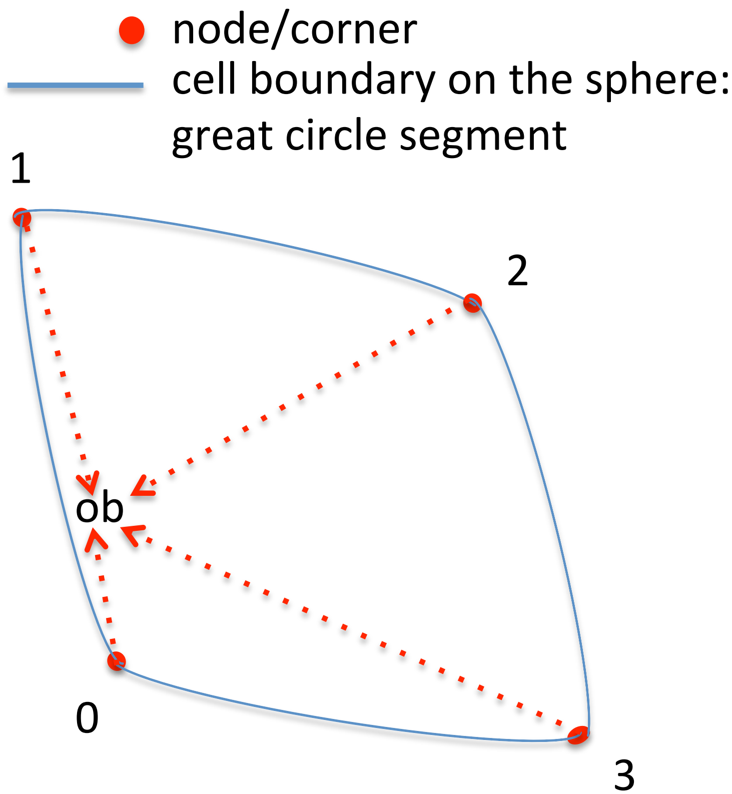
This works for any grid made of quadrilaterals.

Refined mesh grid: all elements are 4-sided, but 3-6 elements (and cells) may share a corner node.



The Goal; interpolate field values at 4 nodes to an observation location (horizontal part)

Bazillions of times



We can find the node closest to the ob more efficiently than a “naïve exhaustive” search.

But finding the 4 nodes that enclose an ob is more complicated; some of the 4 nodes closest to an ob may not be corners of the cell containing the ob.

The Goal; interpolate field values at 4 nodes to an observation location.

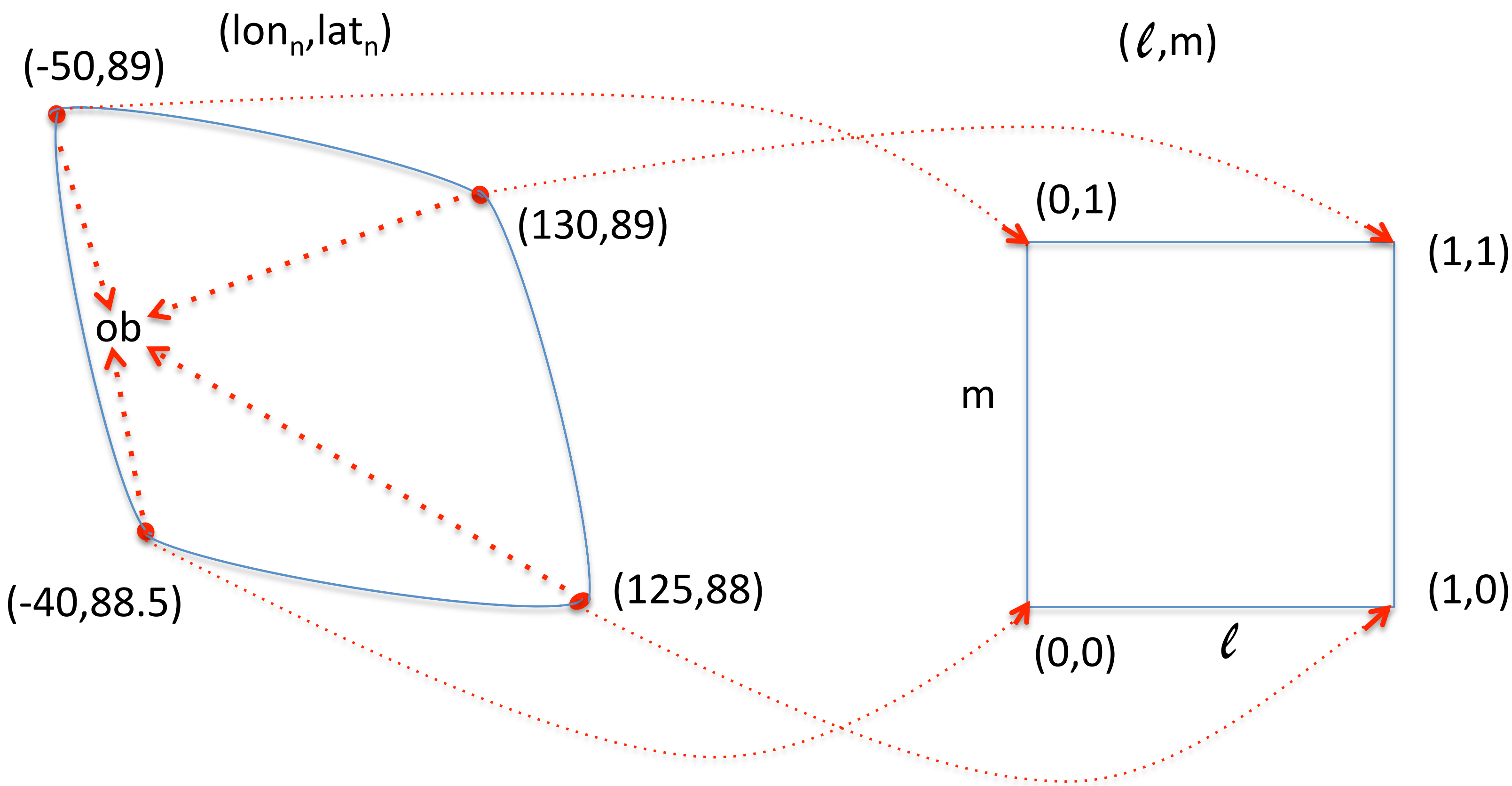
Interpolation can be easier on the unit square

● node/corner

cell boundary on the sphere: great circle segments

$$\text{lon} = a_0 + a_1 * \ell * m + a_2 * m + a_3 * \ell$$

$$\text{lat} = b_0 + b_1 * \ell * m + b_2 * m + b_3 * \ell$$



We can use the 4 corner mappings to generate 4 equations in 4 unknowns (a_n or b_n), solve for a_n in terms of the 4 lon_n (known for this quad), and solve the resulting 2 equations for ℓ and m for any given (ob) lon and lat.

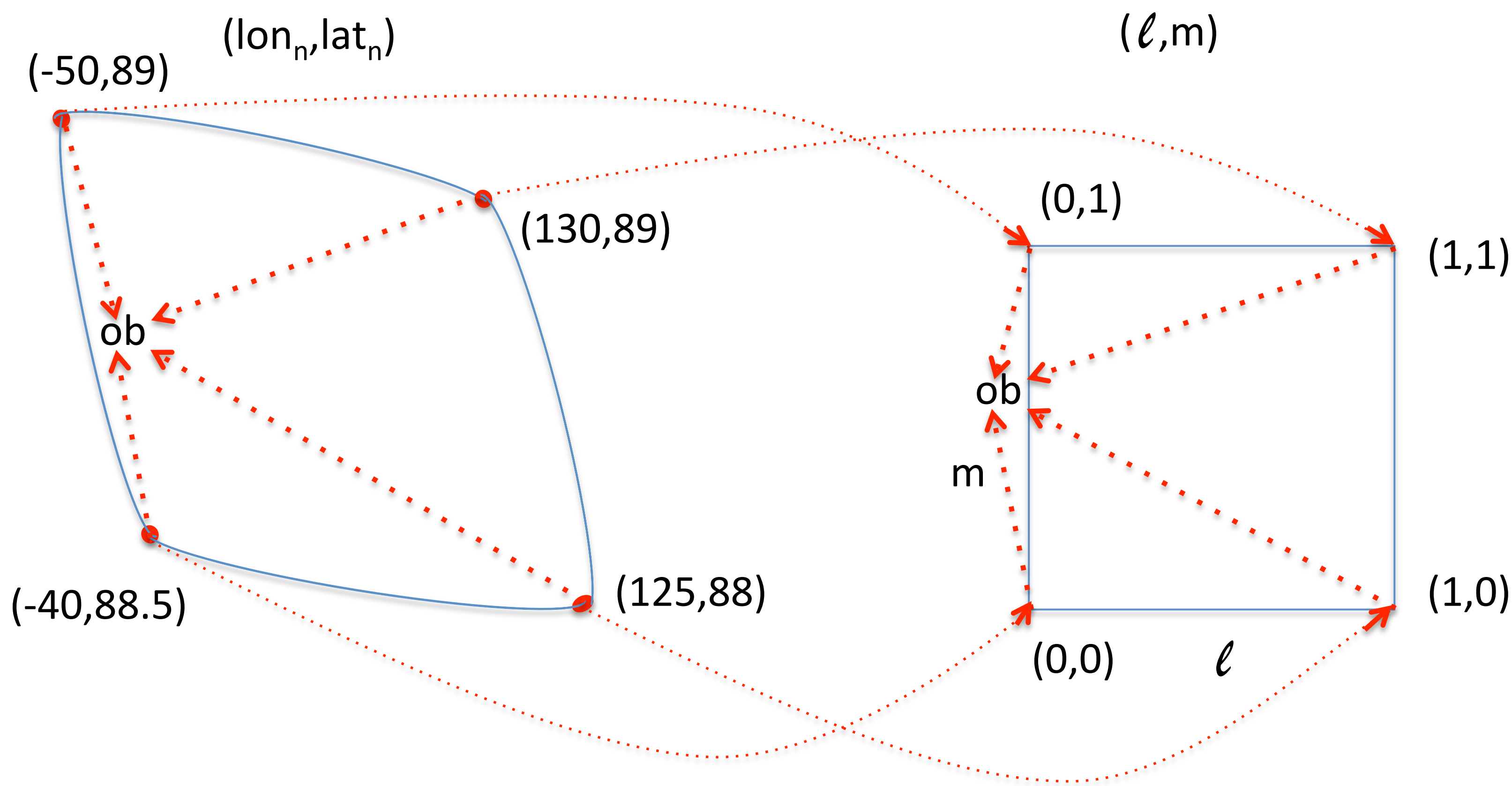
The Goal; interpolate field values at 4 nodes to an observation location.

But a linear mapping of (lon,lat) to the unit square isn't robust.

- node/corner
- cell boundary on the sphere: great circle segment

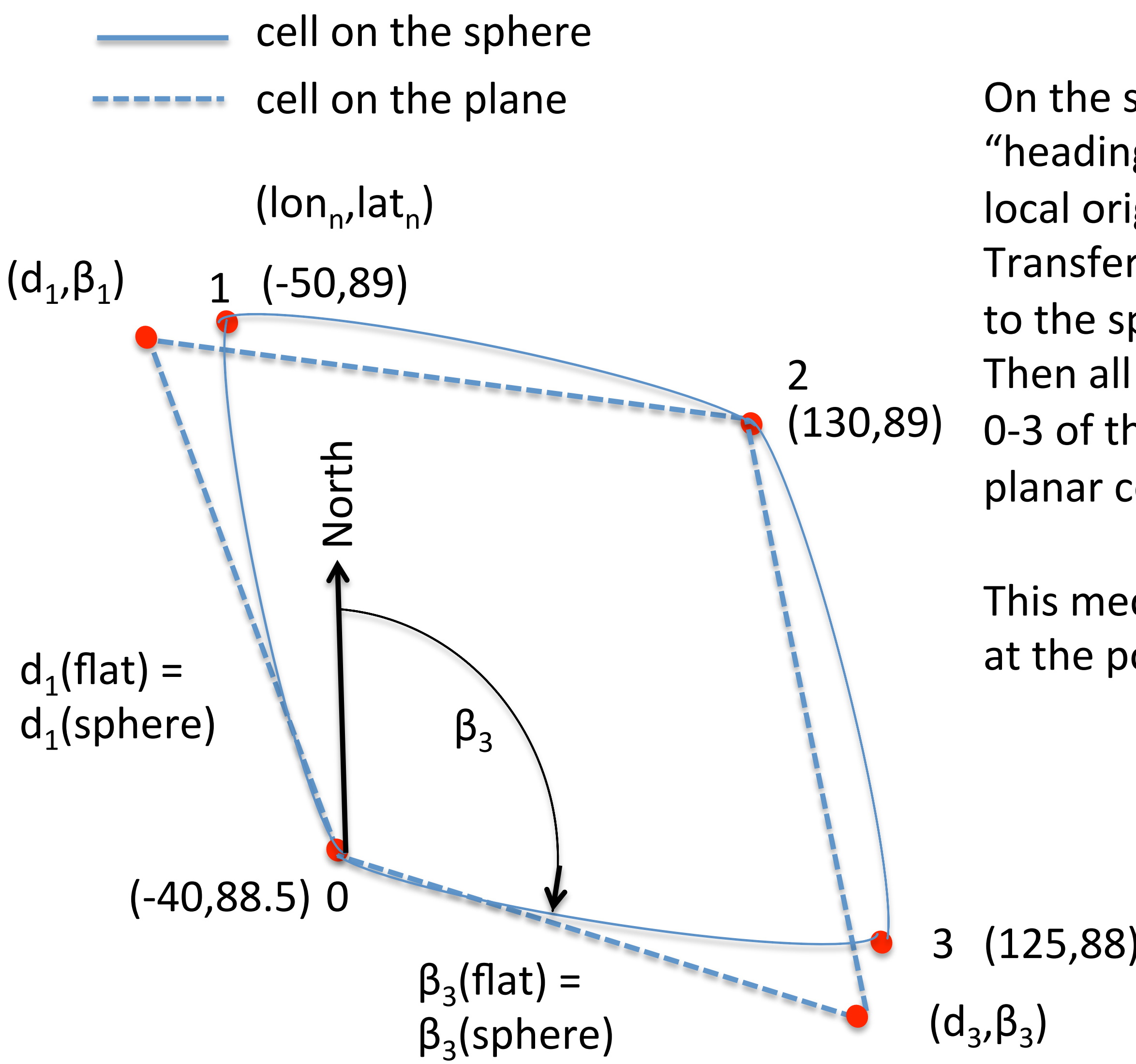
$$lon = a_0 + a_1 * \ell * m + a_2 * m + a_3 * \ell$$

$$lat = b_0 + b_1 * \ell * m + b_2 * m + b_3 * \ell$$



Intermediate maps prevent this

Map 1; flatten the cell using a local radial coordinate system *.



On the sphere, find bearings (β) (aka "headings") and distances (d), from the local origin to corners 1, 2, and 3. Transfer the (d_n, β_n) to the plane tangent to the sphere at the local origin. Then all points "inside" of sides 0-1 and 0-3 of the sphere cell will be inside the planar cell.

This mechanism can handle nodes at the poles.

* I'm looking for a reference for this method.

d = great circle distance, as calculated by DART's get_dist.

β = the bearing; the direction from one point on a sphere to another, along a great circle. North is 0. (Details in slides at the end)

$\beta = \mathrm{atan2}(\ \sin(\Delta\lambda)*\cos(\phi_2),\ \ \cos(\phi_1)*\sin(\phi_2) - \sin(\phi_1)*\cos(\phi_2)*\cos(\Delta\lambda)\ )$

λ = longitude Φ = latitude 1=starting point 2 = destination

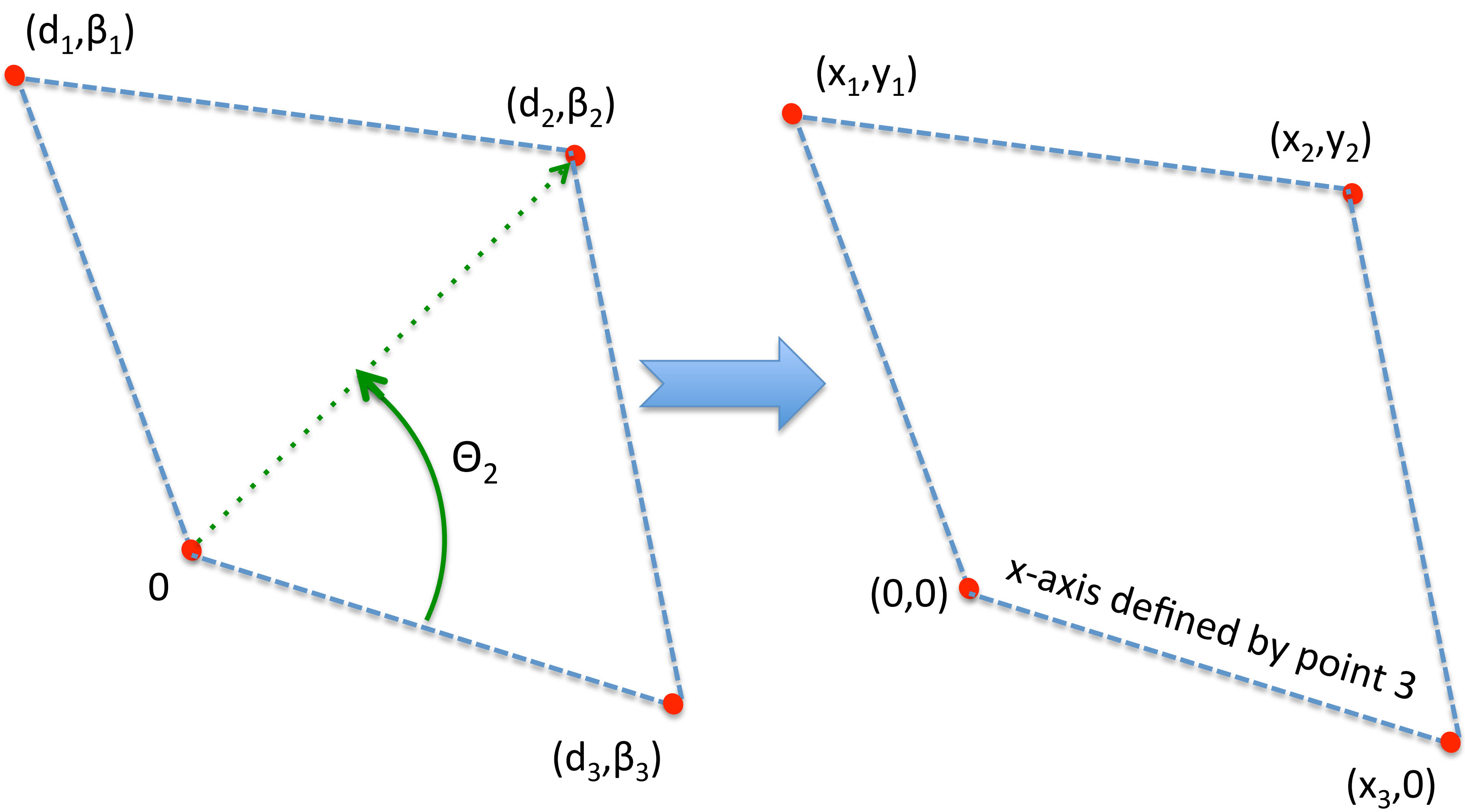
formula from http://www.movable-type.co.uk/scripts/latlong.html

Intermediate maps prevent this

Map 2; Change variables from radial coordinates to cartesian coordinates.

$\Theta_n = \beta_3 - \beta_n$ $\qquad x_n = d_n * \cos(\Theta_n)$ $\qquad y_n = d_n * \sin(\Theta_n)$

This defines 0-3 as the x-axis in (x,y) space.



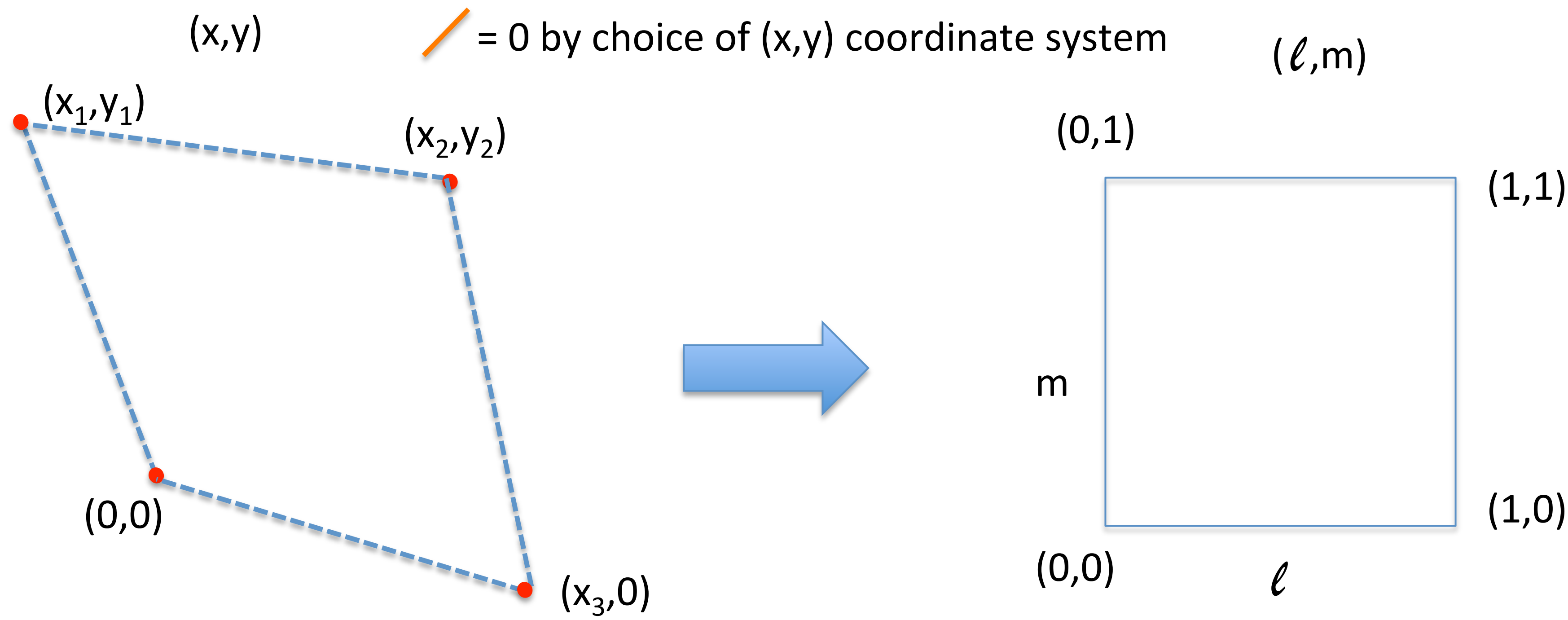
Intermediate maps prevent this

Map 3; Convert the cartesian cell into a unit square.

$$x = \cancel{a_0} + a_1 * \ell * m + a_2 * m + a_3 * \ell$$

$$y = \cancel{b_0} + b_1 * \ell * m + b_2 * m + \cancel{b_3} * \ell$$

$\cancel{}$ = 0 by choice of (x,y) coordinate system



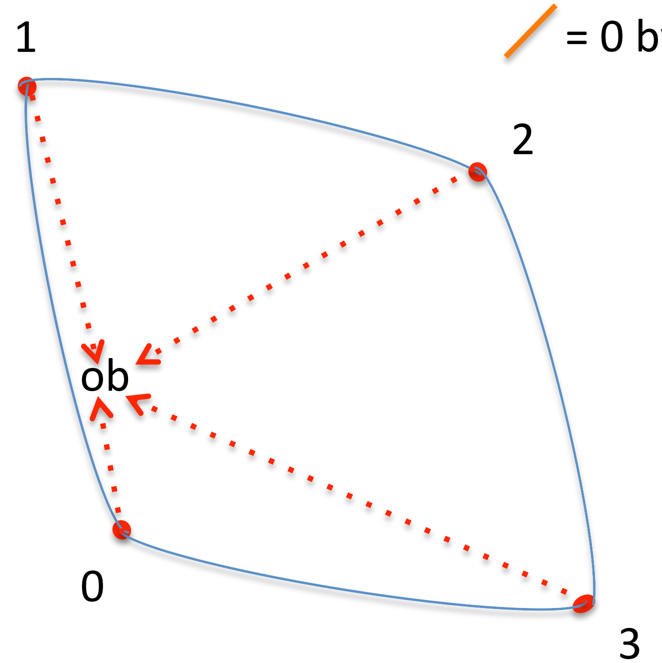
Intermediate maps prevent this

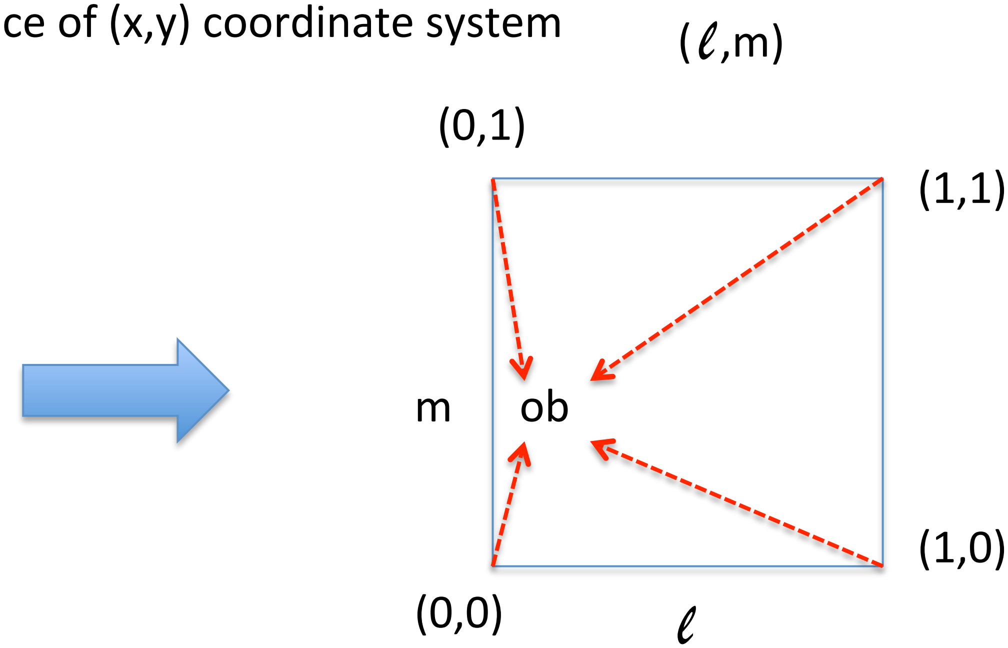
Now observations in the original cell map into the unit square.

$$x = \cancel{a_0} + a_1 * \ell * m + a_2 * m + a_3 * \ell$$

$$y = \cancel{b_0} + b_1 * \ell * m + b_2 * m + \cancel{b_3} * \ell$$

$\cancel{}$ = 0 by choice of (x,y) coordinate system





Use the 4 corner mappings to generate 3 equations in 3 unknowns (e.g. a_n). Solve for a_n in terms of the 3 x_n. Repeat for the 2 b_n in terms of the y_n. Solve the resulting 2 equations in ℓ and m for any given (mapped) observation location (x_o,y_o). See digression about the quadratic equation for m, below.

Summary of the Generation of HommeMapping_cs_grid.nc from HommeMapping.nc

Once for each grid , map each cell from (lon,lat) to the unit square (ℓ,m), using each corner as an origin (see distance distortion slides, below).
This multi-mapping is stored as only 6 numbers at each corner:
$a_{1,2,3}$, $b_{1,2}$, and β_3.

Also store the lists of corners of each cell, and which cells use each corner.

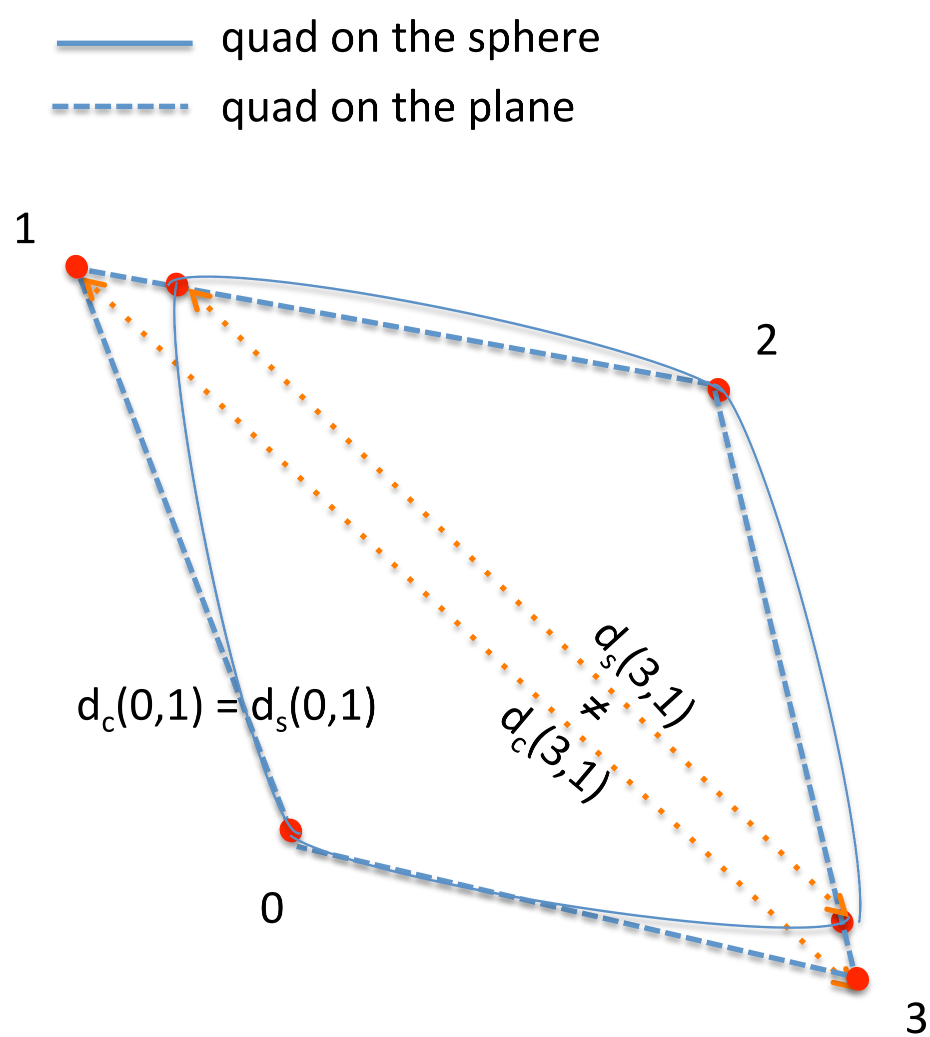
HommeMapping_cs_grid.nc

```
netcdf HommeMapping_cs_grid_ne30 {
dimensions:
    ncenters = 48600 ;
    ncorners = 4 ;
    max_neighbors = 6 ;
    ncol = 48602 ;
    ncoef_a = 3 ;
    ncoef_b = 2 ;

global attributes:
        :elements_per_cube_edge = 30 ;
        :nodes_per_element_edge = 4 ;
```

```
variables:
    int num_nghbrs(ncol) ;
        num_nghbrs:long_name = "number of neighbors of each node" ;
        num_nghbrs:units = "nondimensional" ;
        num_nghbrs:valid_range = 1, 6 ;
    int centers(ncol, max_neighbors) ;
        centers:long_name = ”cells which use node as a corner" ;
        centers:units = "nondimensional" ;
        centers:valid_range = 1, 48600 ;
    int corners(ncorners, ncenters) ;
        corners:long_name = "corners/nodes of each cell" ;
        corners:units = "nondimensional" ;
        corners:valid_range = 1, 48602 ;
    double a(ncenters, ncorners, ncoef_a) ;
        a:long_name = "Coefficients of mapping from planar x coord to unit square" ;
        a:units = "nondimensional" ;
    double b(ncenters, ncorners, ncoef_b) ;
        b:long_name = "Coefficients of mapping from planar y coord to unit square" ;
        b:units = "nondimensional" ;
    double x_ax_bearings(ncenters, ncorners) ;
        x_ax_bearings:long_name = "bearing (clockwise from North) from origin
                      node(corner 4)  of each mapping to corner 3" ;
        x_ax_bearings:units = "radians" ;
        x_ax_bearings:valid_range = -3.14159265358979, 3.14159265358979 ;
```

Why use each corner as an origin?



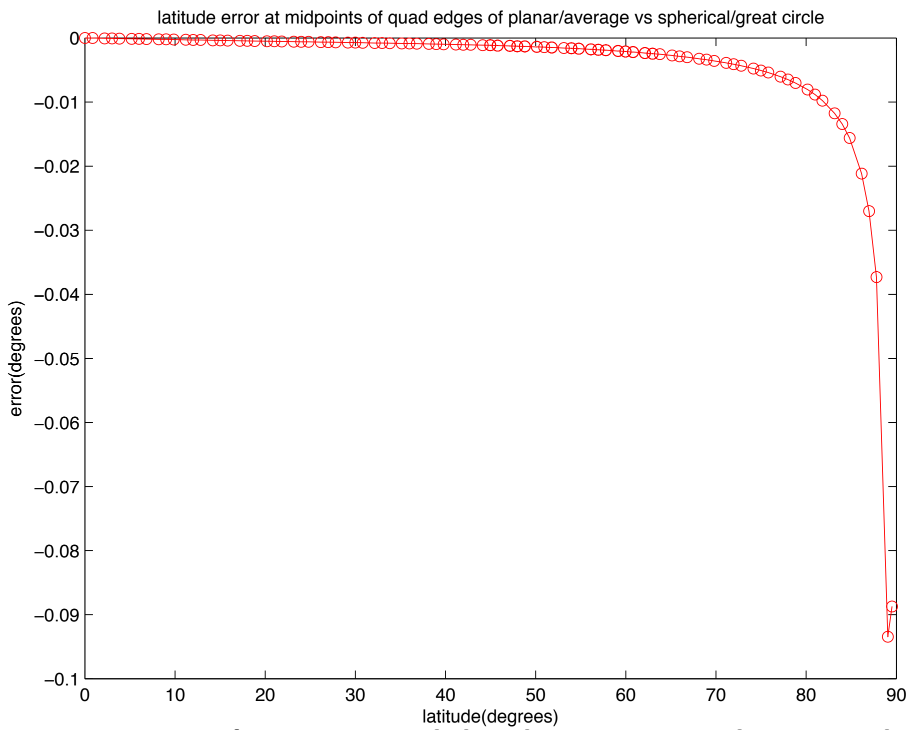
- Bearings and distances from origin to corners 1, 2, and 3 are preserved by definition.
- But other distances are distorted and obs near sides 1-2 and 2-3 may not be inside the plane quad.

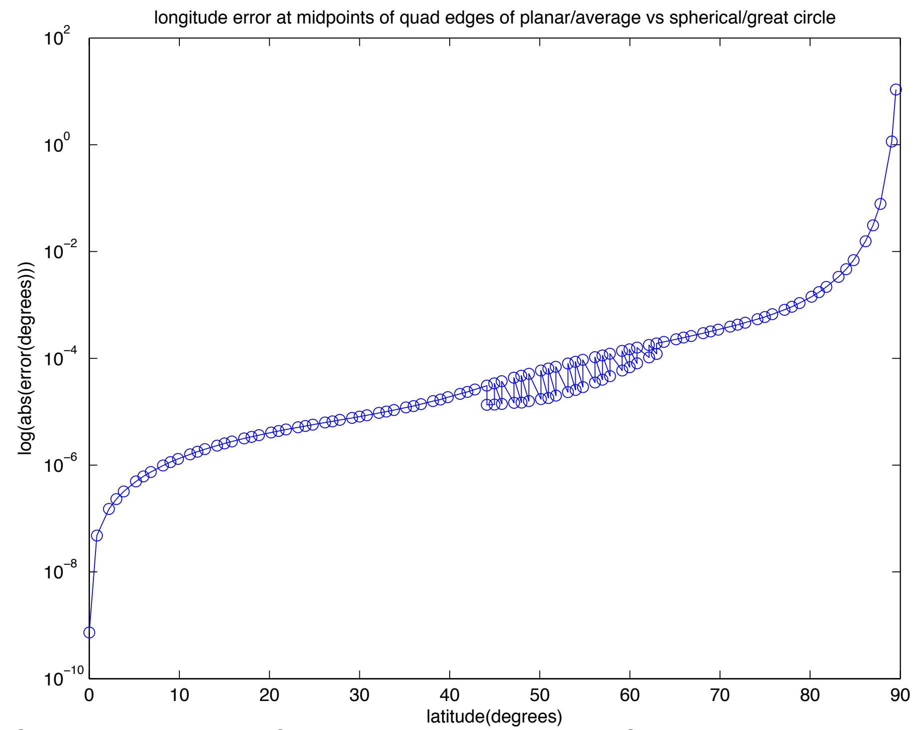
But

This implies a linear interpolation of longitude and latitude between the corners, which is inconsistent with the cell boundaries, which are great circle segments.

Obs just inside a quad boundary can appear outside the unit square boundary, and vice versa.

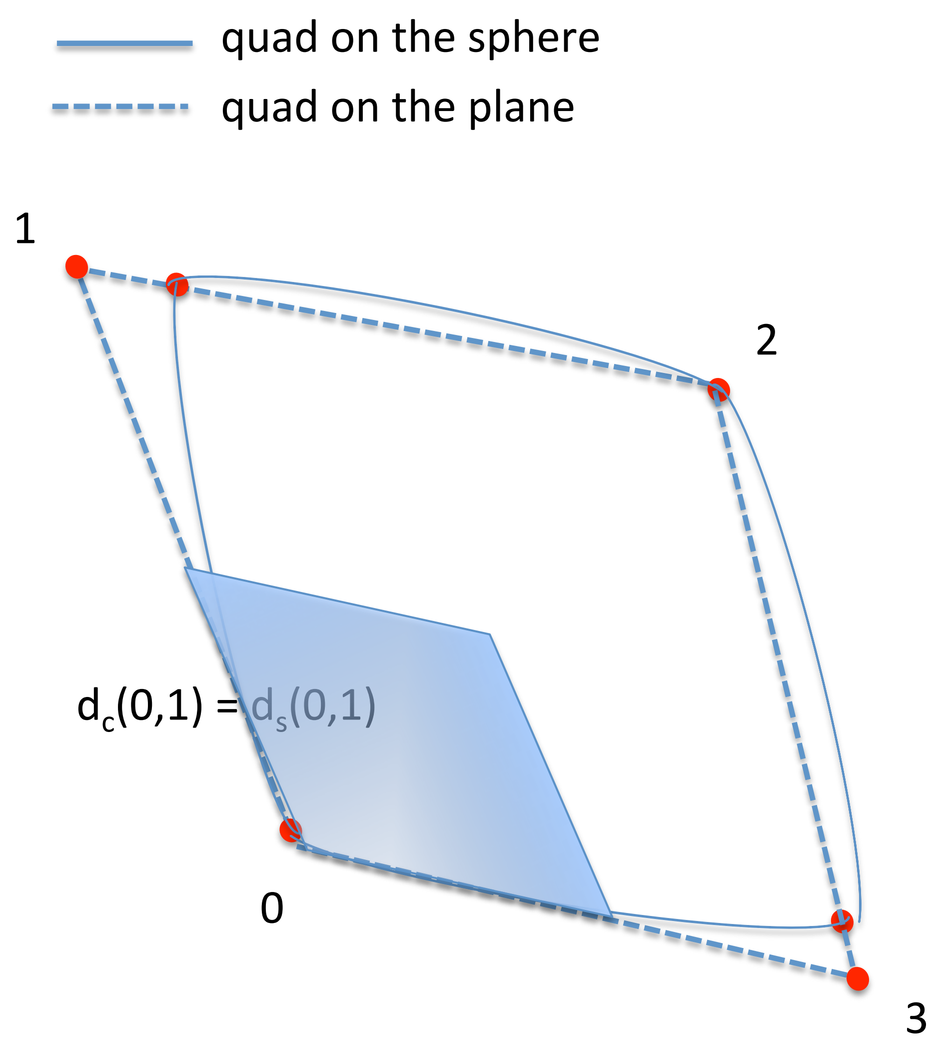
Here are the errors of the positions of the (~1-degree) cell edge midpoints, as calculated by linear interpolation, relative to a spherical coordinate mid-point formula.





It's most likely near the poles, but it can happen anywhere.

Robust solution to planar mapping distance distortion



- This is OK if there are no obs near sides 1-2 and 2-3. Do this by defining a separate planar coordinate system for each corner. Then obs will always be in the quadrant farthest from sides 1-2 and 2-3 and closest to the origin.
- This sounds complicated and time-consuming, but it doesn't take long, since it's a 2D problem, and it can all be done once for each grid, before any assimilations.

Summary of 3-Maps Interpolation Method

① Before any assimilation, map each cell from (lon,lat) to the unit square (ℓ,m), using each corner as the origin. Each mapping is stored as only 6 numbers: $a_{1,2,3}$, $b_{1,2}$, and β_3.

Also store the lists of corners of each quad, and which quads use each corner.

② During an assimilation use location_mod:get_close_obs to identify the nodes which are closest to the ob.

③ Search the 3-6 cells that use the closest (or 2nd closest) node as a corner to see which contains the ob:

Θ_o = angle from (stored) x-axis of the closest node to observation

$(x_o,y_o) = d_o * [\cos\Theta_o, \sin\Theta_o]$

Solve $0 = m^2(a_1b_2 - a_2b_1) + m(a_3b_2 - a_1y_o + b_1x_o) - a_3y_o$ for m

and $\ell = (x_o - a_2m)/(a_3 + a_1m)$

(from plugging (x_o,y_o) into the mapping equations and solving for m and ℓ)

If $0 \le m \le 1$ and $0 \le \ell \le 1$ then we've found the containing cell AND numbers that can be used in interpolation weights.

Evaluation of which root of the m quadratic equation to use.

$aa*m^2 + bb*m + cc = 0$

$aa = a_1b_2 - a_2b_1$
$bb = a_3b_2 - a_1y_o + b_1x_o$
$cc = - a_3y_o$

The cell coordinate systems were defined so that x3 > 0, which means a3 > 0.
All y ≥ 0 (for points in the cell). So cc can be written as -|cc|.

Then the solution to the quadratic equation can be written $m = \frac{bb}{2aa}\left(-1 \pm \sqrt{1 + \frac{4aa|cc|}{bb^2}}\right)$

For aa > 0 the sqrt term > 1. Looking at the case of the largest bb, for:
- bb > 0 only the +root can yield m > 0.
- bb < 0 only the –root can yield m > 0.

Smaller bb make the sqrt term larger, and it dominates the -1 term even more.

For aa < 0 the sqrt term < 1. Looking at the case of the largest bb, for:
- bb > 0 either root can yield m > 0. But which, if either, yields m < 1 depends on exact sizes of aa,bb,cc.
- bb < 0 neither root can yield m > 0.

Smaller bb make the sqrt term smaller, and the -1 term dominates it even more.

bb<0 (that is, the same sign as cc) for cells that are distorted towards triangular, either by having a very short side, or by having a corner pushed toward the center, so that 2 sides are nearly co-linear.
This is explored in a matlab script and its output in PIC_check_bb (was bb_ccneg) and roots_of_m_equation.pptx.

$$0 = m^2(a_1b_2 - a_2b_1) + m(a_3b_2 - a_1y_o + b_1x_o) - a_3y_o$$

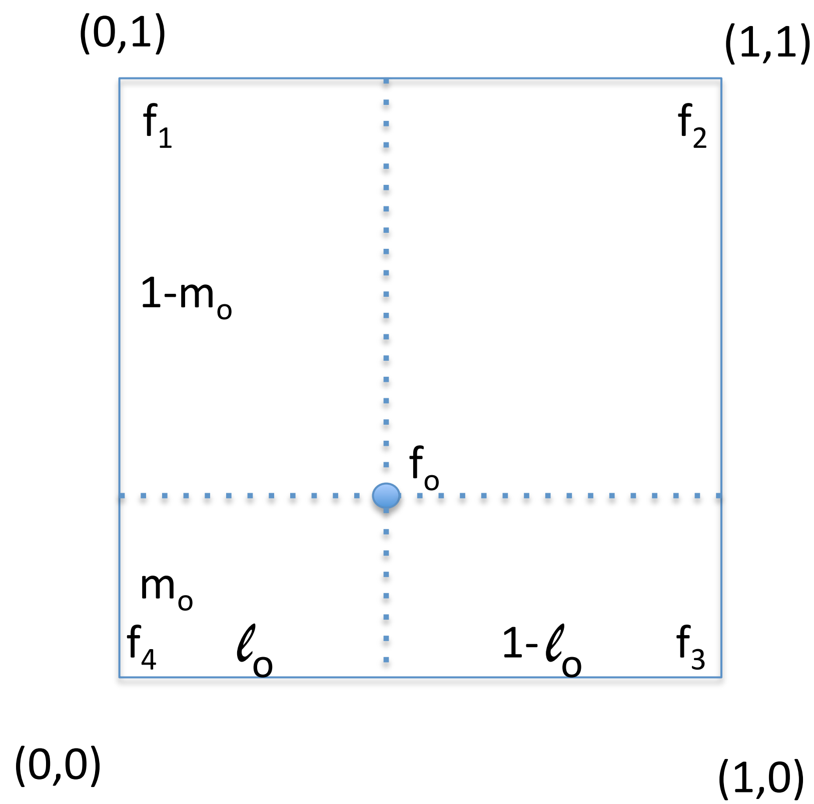
This can be restricted further, in the case of grids actually used for CAM-SE.
In particular, if bb > 0 always, then the +root will yield a good mapping.
We can use the fact that the cells are not highly distorted in the sense that all 4 sides are roughly the same size, and they are not squished into skinny diamond shapes or nearly triangular.

From the mapping of the 4 corners of the (x,y) cell to the (l,m) square we have
$a_3 = x_3 > 0$ and $b_2 = y_1 > 0$ by definition of the (x,y) cell. x_3 and y_1 are not small.
So the first bb term is > 0 and not small.
$a_1 = x_2 - x_1 - x_3 = (x_2 - x_1) - (x_3\text{-}0)$ = the difference of the baseline side and the opposite side. This is small.
$b_1 = y_2 - y_1$. $y_n > 0$ and roughly equal, so b_1 is small.
In the "worst case" for making bb > 0, we would have the signs alligning to make the 2nd and 3rd terms < 0, and x_o and y_o not small, but they're multiplied by small numbers, so the first (large positive) term of bb dominates.

The code tests for bb < 0, uses the –root if needed, and prints a warning that it appears that the gird has highly distorted cells. It does not keep both roots in the case where both yield usable mappings.

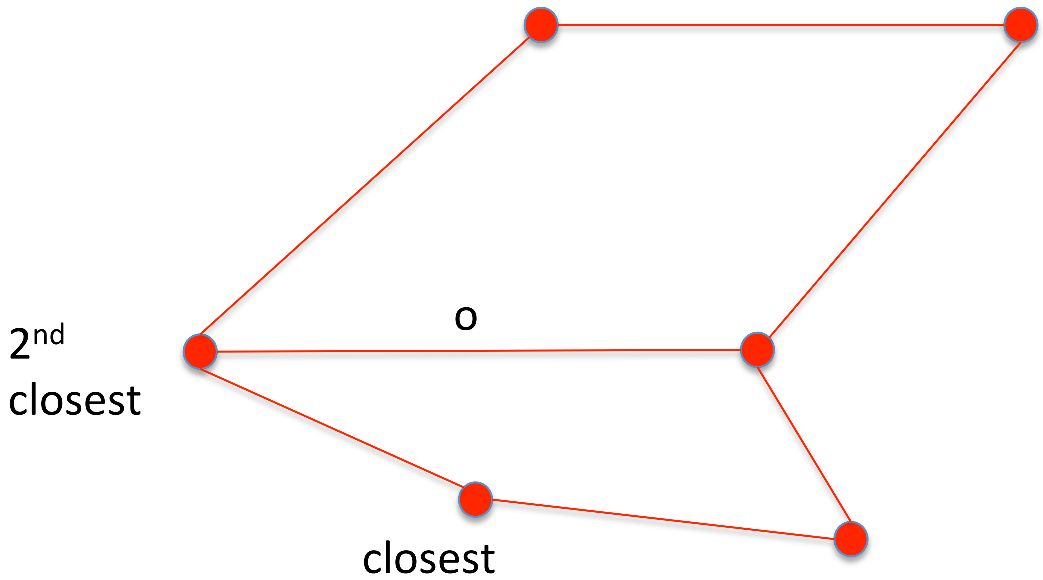
④ Use ℓ and m as weights to interpolate the field values at the corners, f_n, to the ob location.

$$
\begin{aligned}
f_o = \; & f_2 * \ell_o * m_o \\
& + f_1 * (1-\ell_o) * m_o \\
& + f_4 * (1-\ell_o) * (1-m_o) \\
& + f_3 * \ell_o * (1-m_o)
\end{aligned}
$$



Refined Grid; 'wrong quad' problem

Near the boundary between coarser and finer grids the nodes/quads can look like

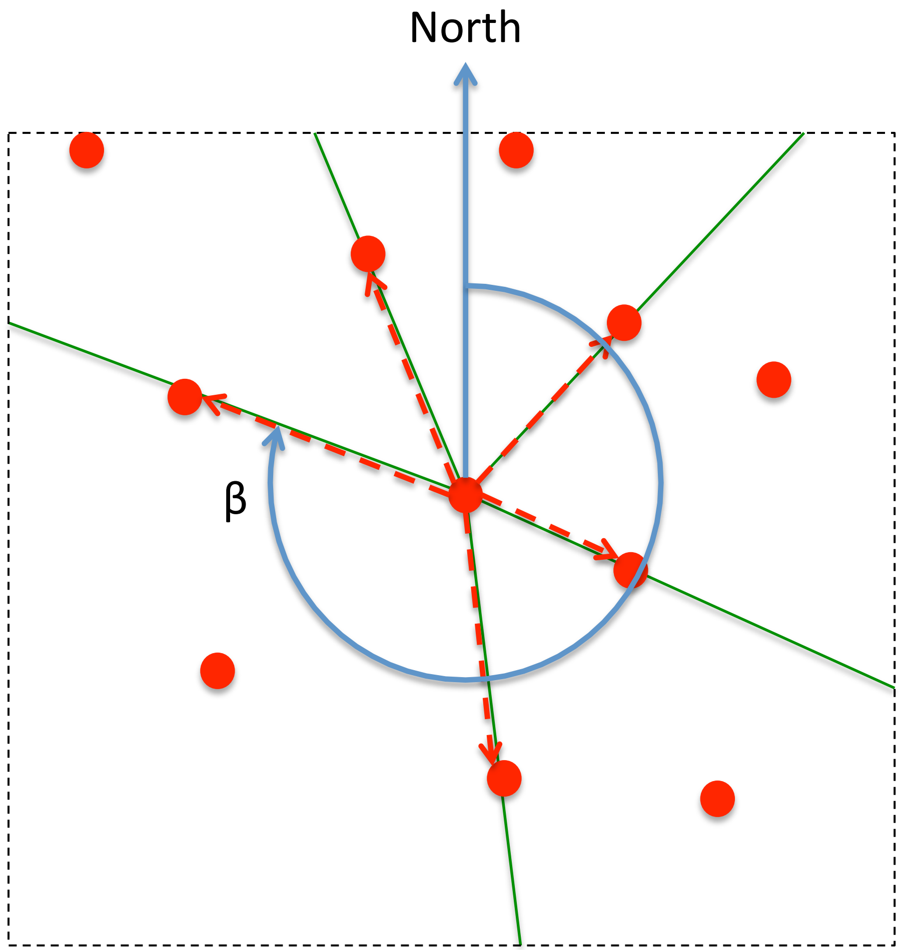


The closest node is not one that defines the cell that the ob is in.
But the 2nd closest must be (at least for the cubed sphere grid).

Check if this is the case by mapping the ob location into each of the cells that use the closest node as a corner. If they all fail, do the same for the 2nd closest node.

Bearings Details

'Bearing': direction from one point on a sphere to another along a great circle.



Not a cheap calculation, so store the bearing of the x-axis for each before any assimilation.

For the 1° refined grid,
num_nodes = ~147,000,
num_corners = ~4,
so ~600,000 bearings.

1° standard grid: 1/3 as much

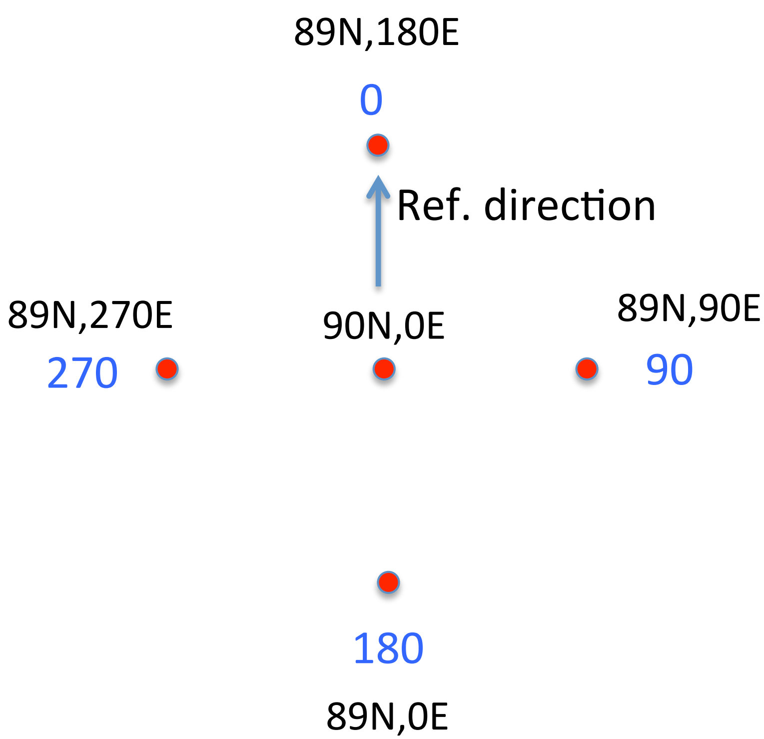
$$\beta = \mathrm{atan2}(\ \sin(\Delta\lambda)*\cos(\phi_2),\ \ \cos(\phi_1)*\sin(\phi_2) - \sin(\phi_1)*\cos(\phi_2)*\cos(\Delta\lambda)\ )$$

λ = longitude Φ = latitude 1=starting point 2 = destination

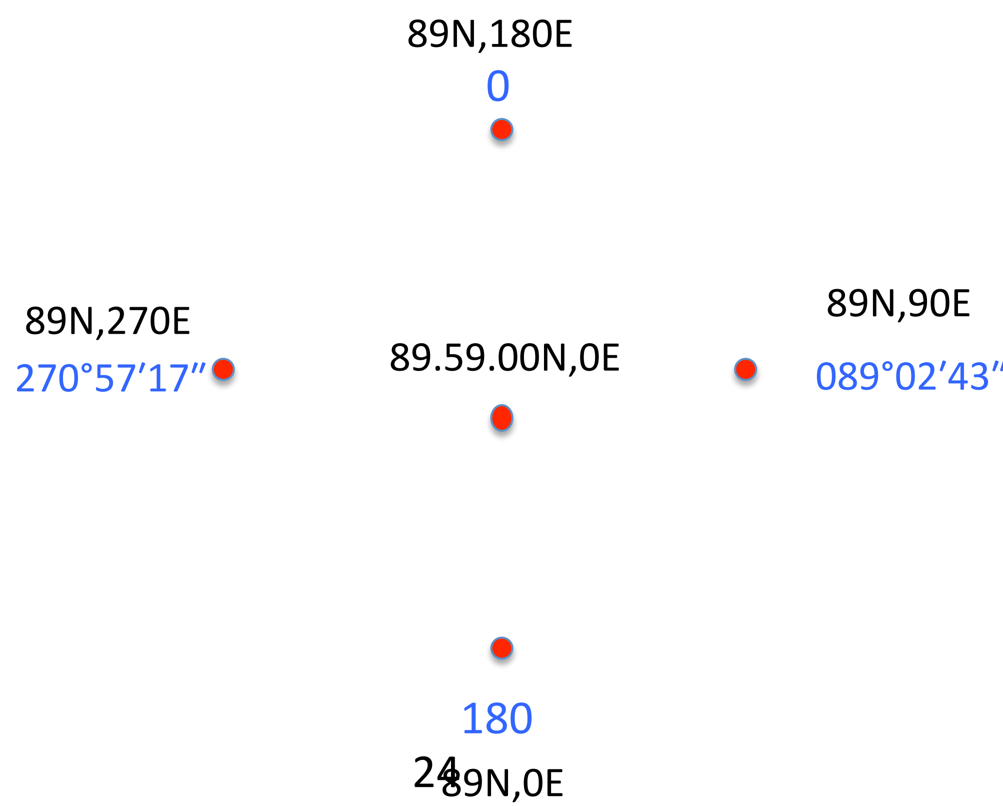
What's the bearing at and near the poles?

Setting λ_1 = 0E (for φ_1=90N) can be understood as arbitrarily setting the reference direction "to the north pole" to the be the vector from any point on the 0E meridian *towards* the north pole. Then bearings from the north pole to other points are measured from that reference. Such a choice would be necessary, at most, once for a cell, so there won't be a confusion of reference directions.

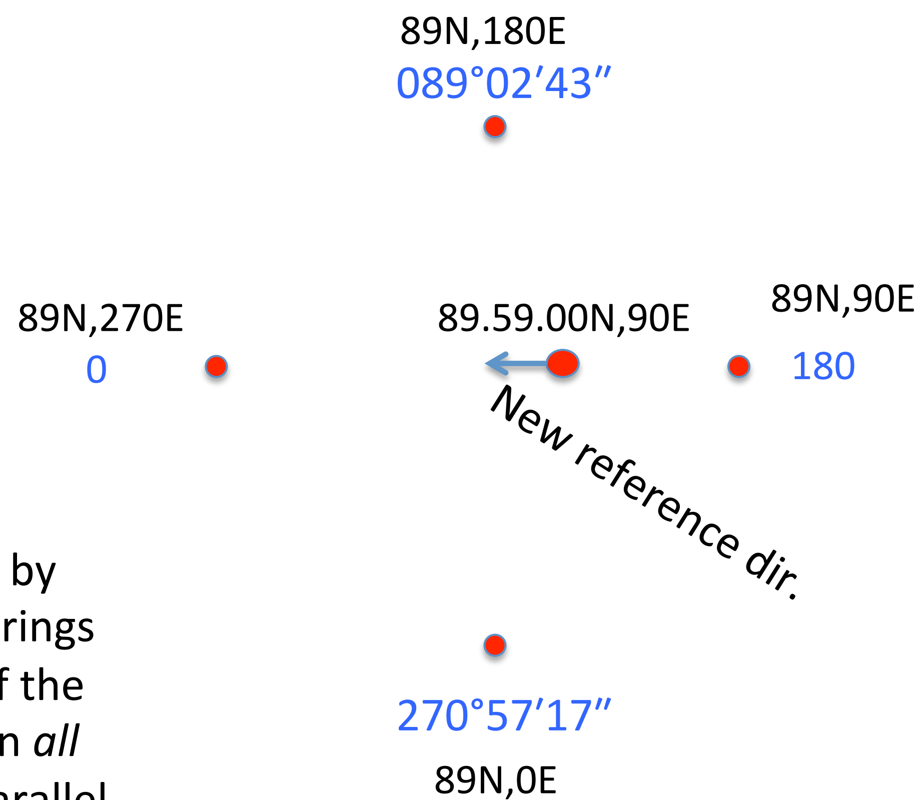
Initial point at the N pole (and '0E')



Initial point displaced towards 0E



Initial point displaced toward 90E



Note rotation of bearing values caused by rotation of reference direction. So bearings are not continuous for perturbations of the initial point away from the north pole in *all* directions. Only in directions mostly parallel to the north pole reference direction.

Discussion

Efficiencies:

- Use the new (x,y,z) get_close_obs, which returns a list of closest obs.
- Small angle approximations to avoid sines and cosines (away from the poles)?
- Trig function look up tables? (as in threeed_sphere/location_mod.f90)?
- Order the quads around each node, in order to calculate the right one, instead of searching all (average of 2 failures (x 4 corners)/quad).
- Cache interpolation weights for obs at the same location.
- Timing of recalculating HommeMapping_cs_grid.nc? vs using a pre-existing file, which complicates the scripts.
- …?