Data Assimilation with SE-CAM and DART

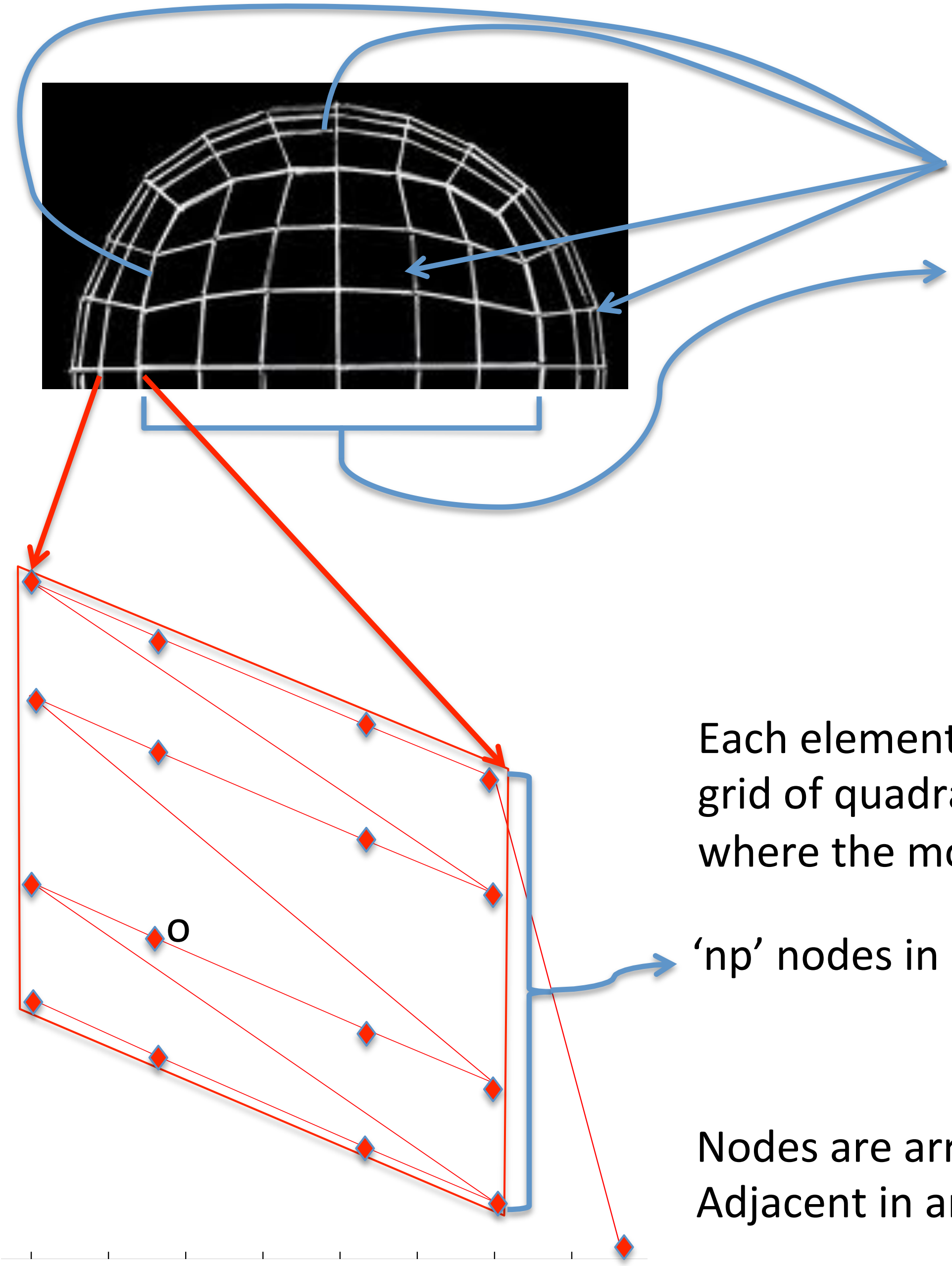
Kevin Raeder[1], Jeffrey L. Anderson[1], Colin Zarzycki[2]

[1] NCAR/CISL/IMAGe
[2] University of Michigan

Motivation

- SE-CAM is/will be the default for "1-degree" and finer CAMs.
- Refined mesh studies will add to the desire to use CAM in forecast mode for synoptic studies.
- DA can help with evaluation of SE-CAM.
- We need the good scaling to make use of current and future supercomputers.

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.

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

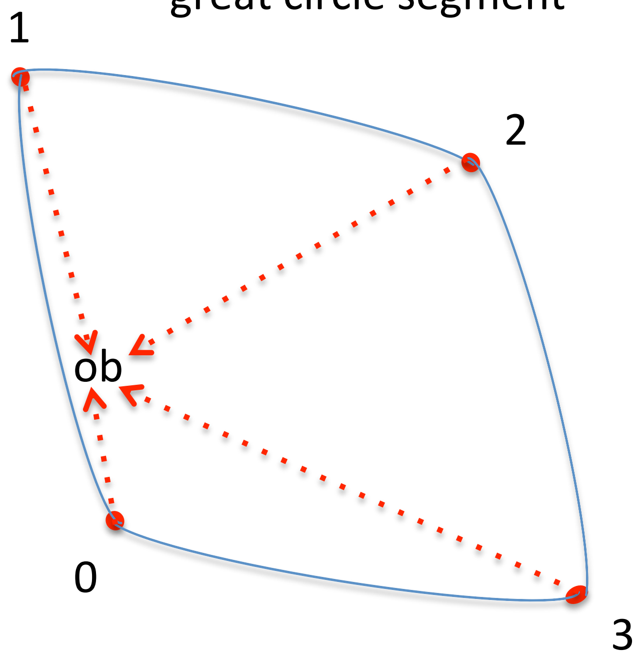
Nodes are arranged as a 1D array.
Adjacent in array ≠ adjacent on sphere

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

Bazillions of times

● node/corner

— cell boundary on the sphere: great circle segment



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

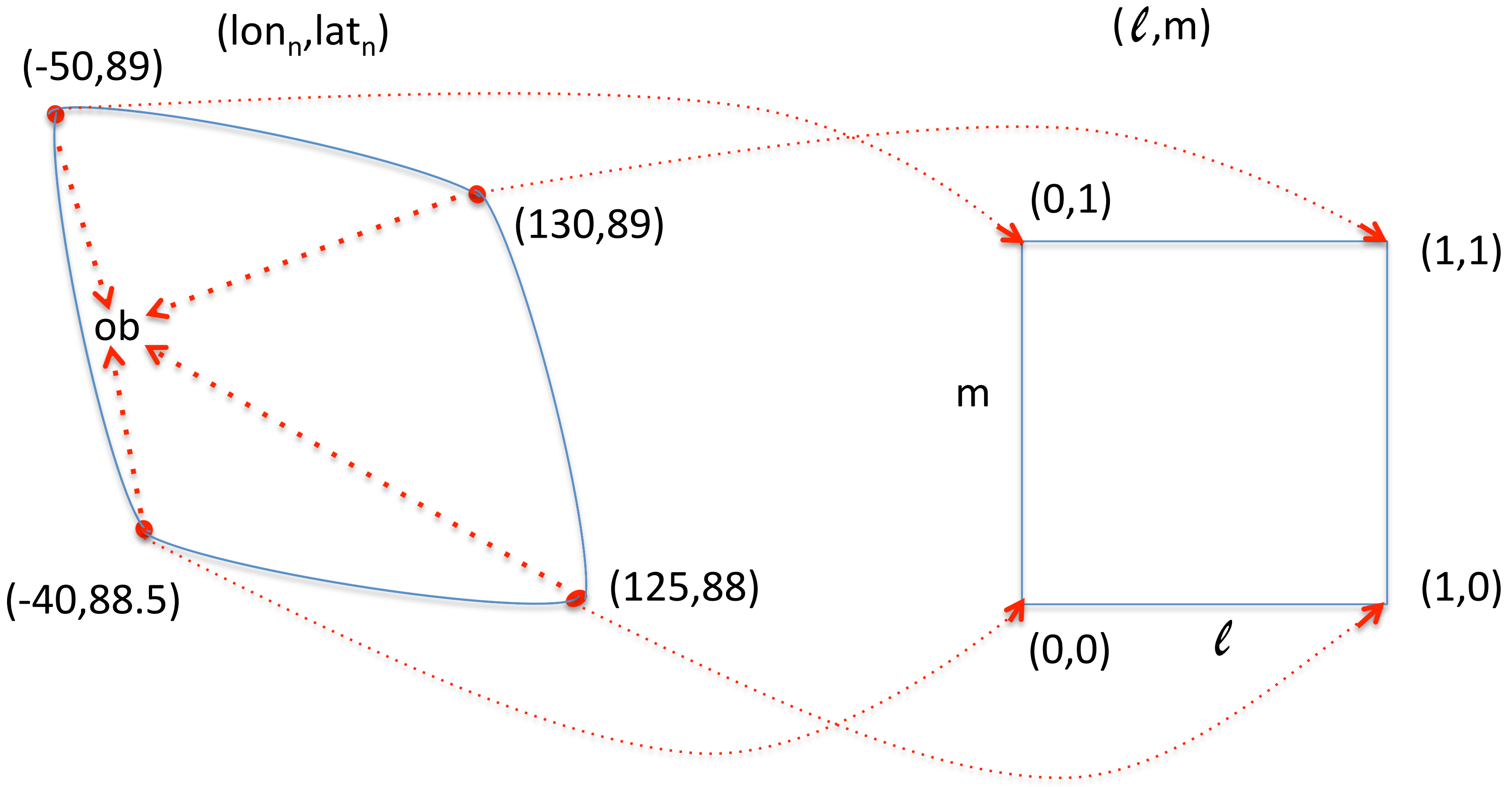
But finding the 4 nodes that enclose an ob is more complicated; some of the 4 nodes closest to an ob may not be corners of the cell containing the ob.

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 segment

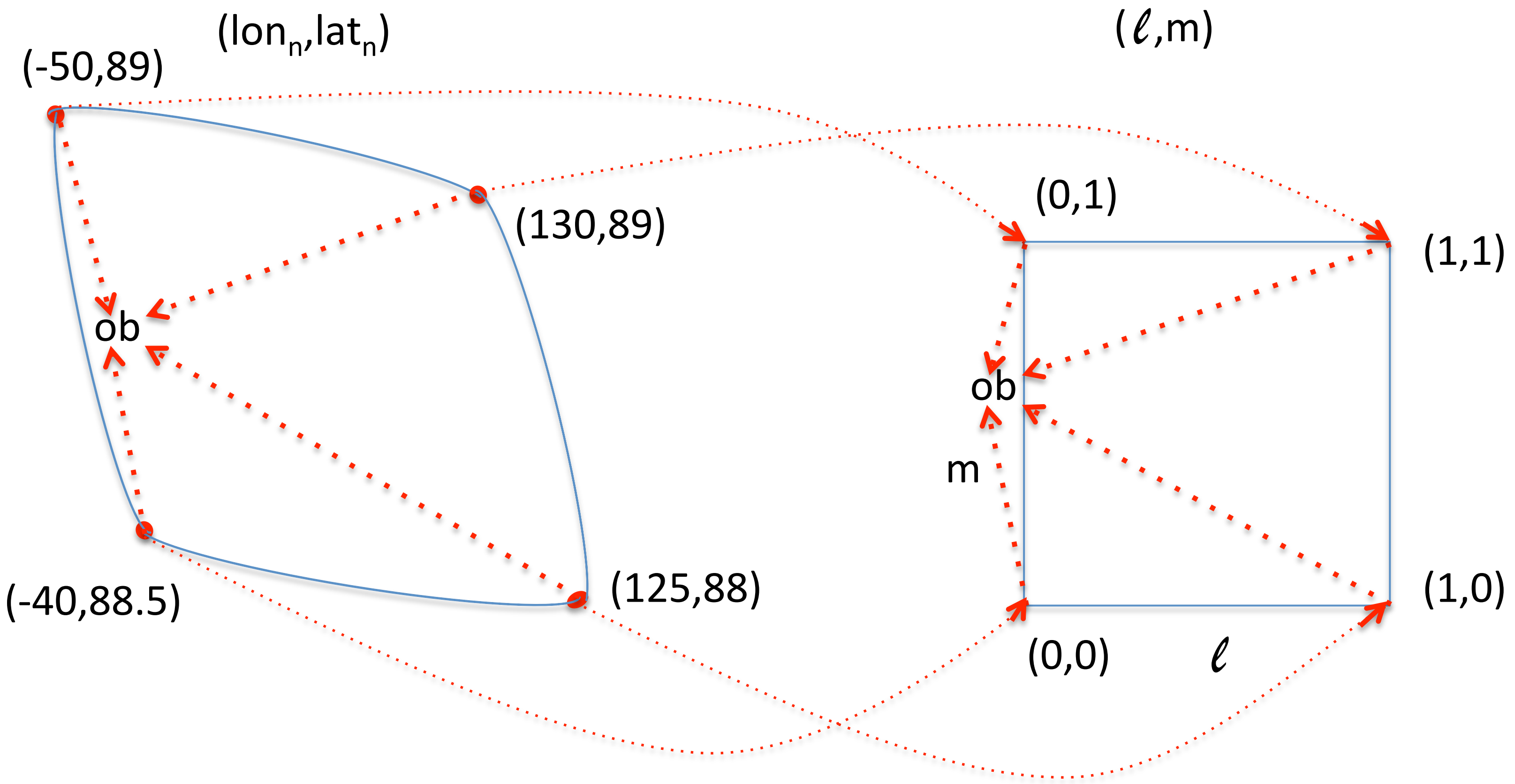


The Goal; interpolate field values at 4 nodes to an observation location.

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

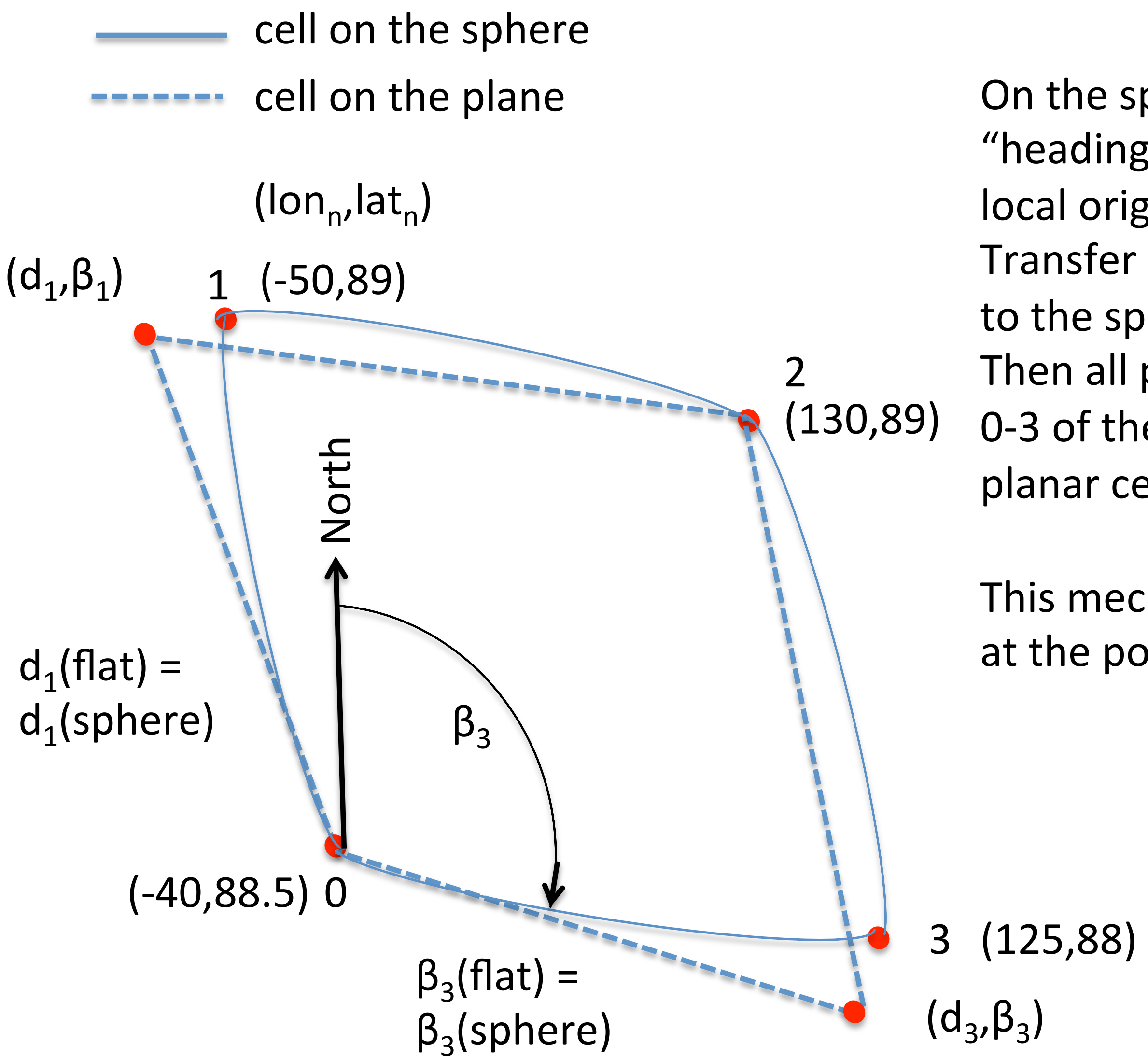
● node/corner

— cell boundary on the sphere:
great circle segment



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 $(d_{n,}\beta_n)$ to the plane tangent to the sphere at the origin. Then all points inside of sides 0-1 and 0-3 of the sphere cell will be inside the planar cell.

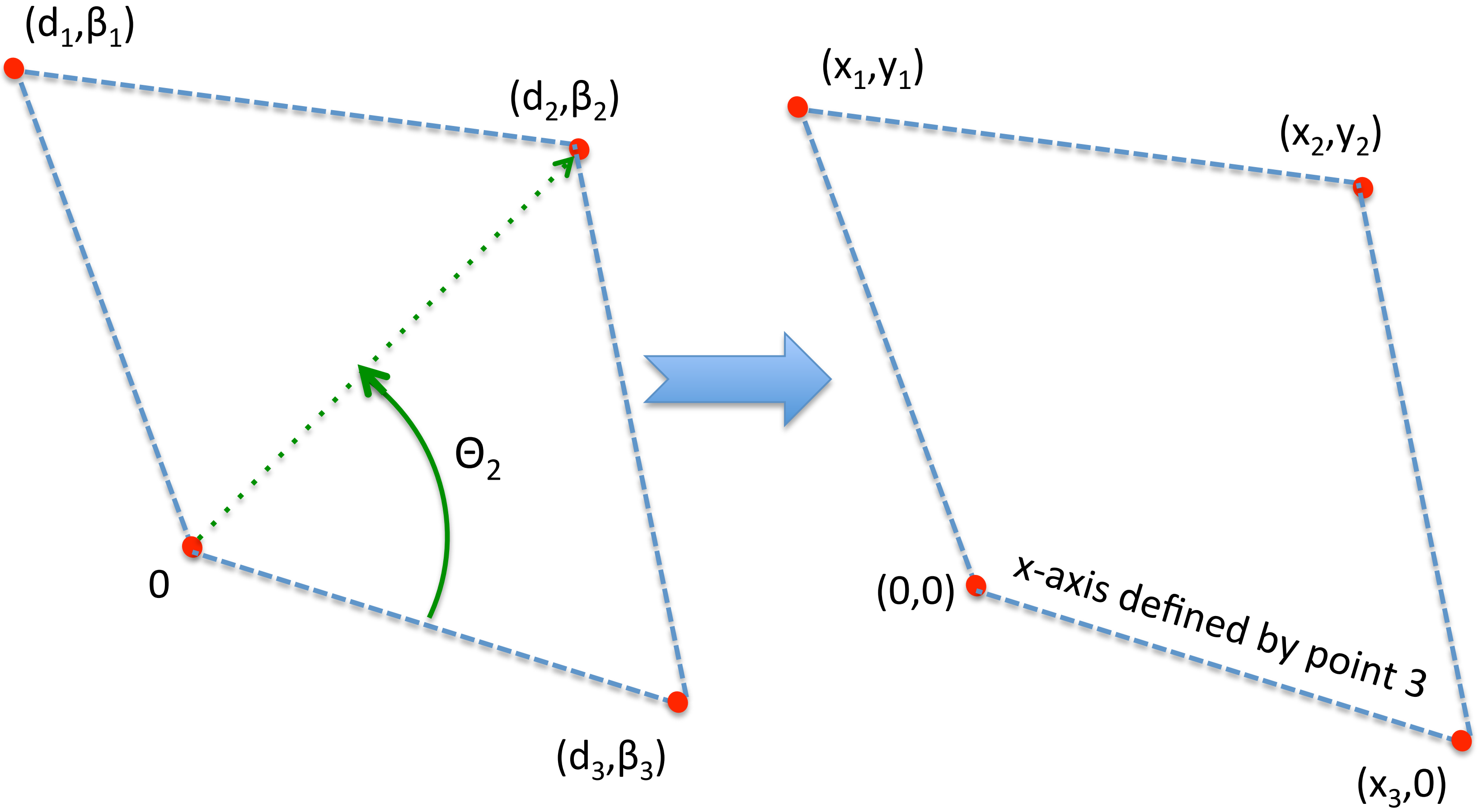
This mechanism can handle nodes at the poles.

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

Intermediate maps prevent this

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

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



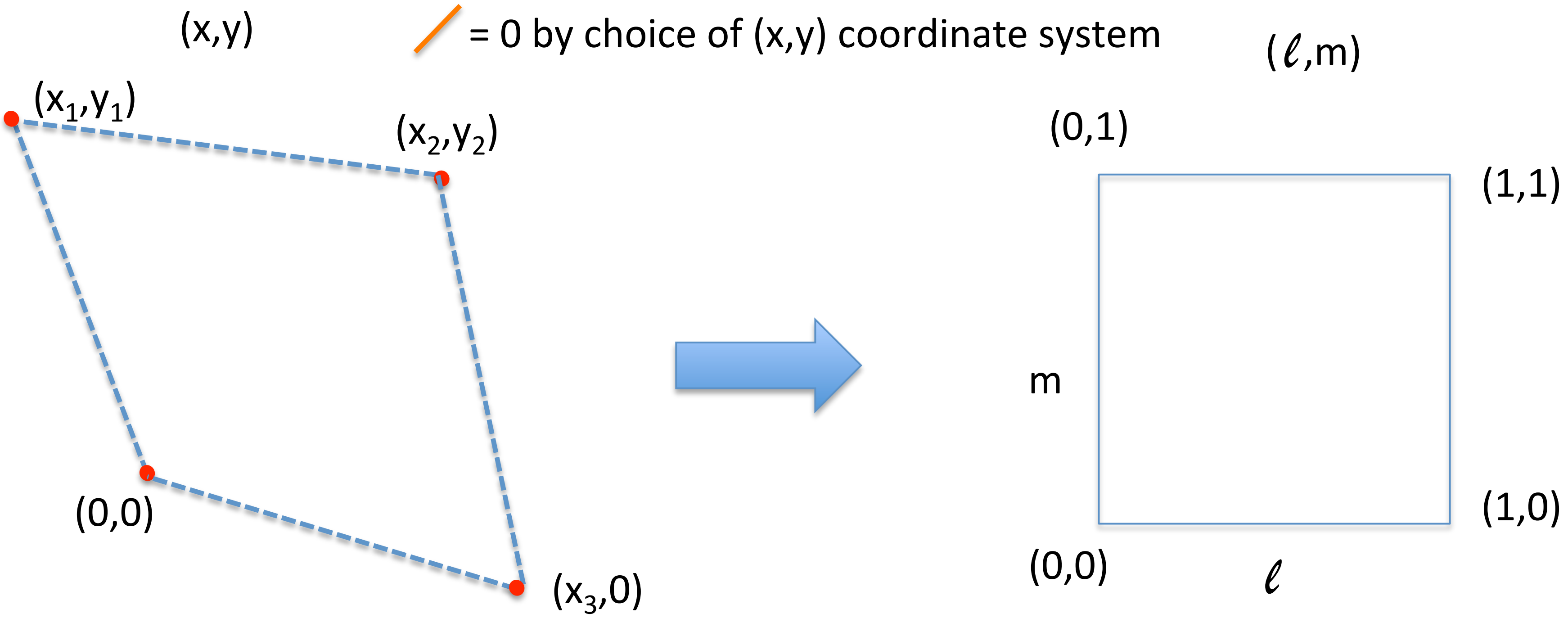
Intermediate maps prevent this

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



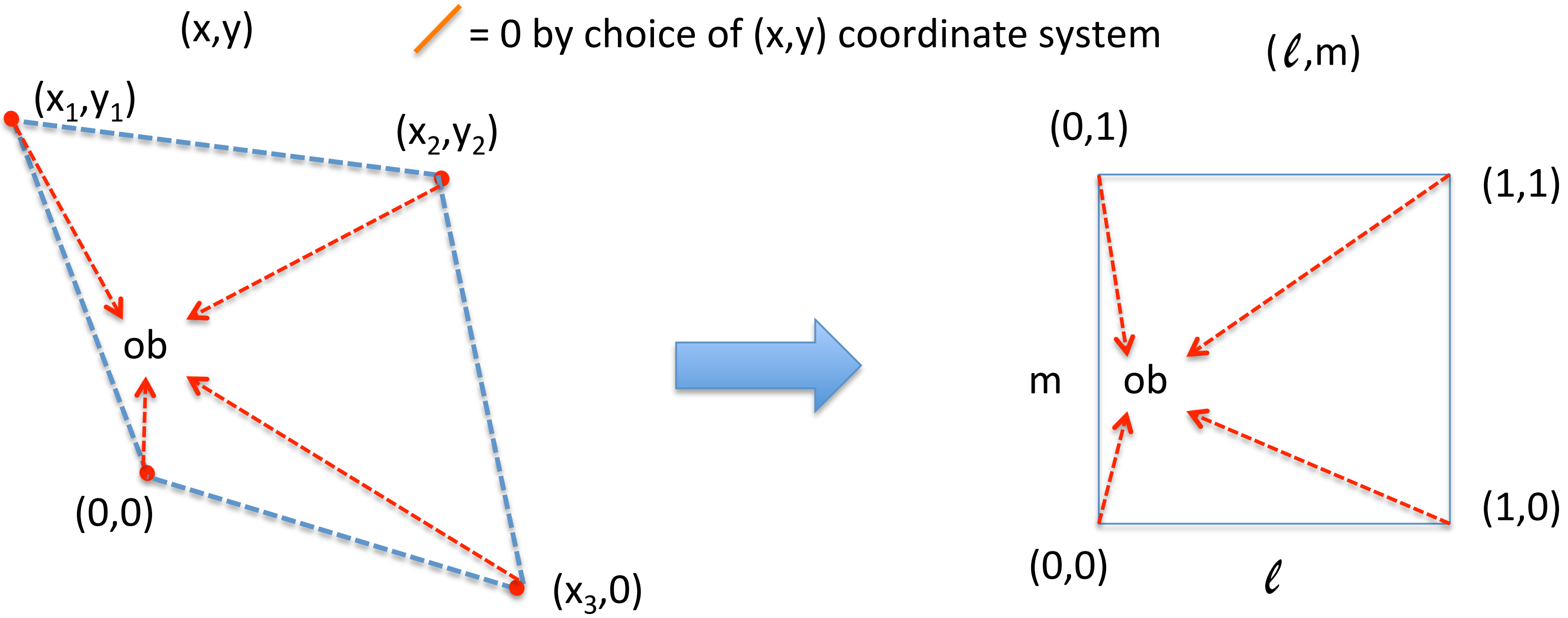
Intermediate maps prevent this

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 4 x_n. Repeat for b_n.
Solve the resulting 2 equations in ℓ and m for any given (mapped) observation location (x_o,y_o).

Summary of 3-Maps Interpolation Method

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

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

Summary of 3-Maps Interpolation Method

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

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

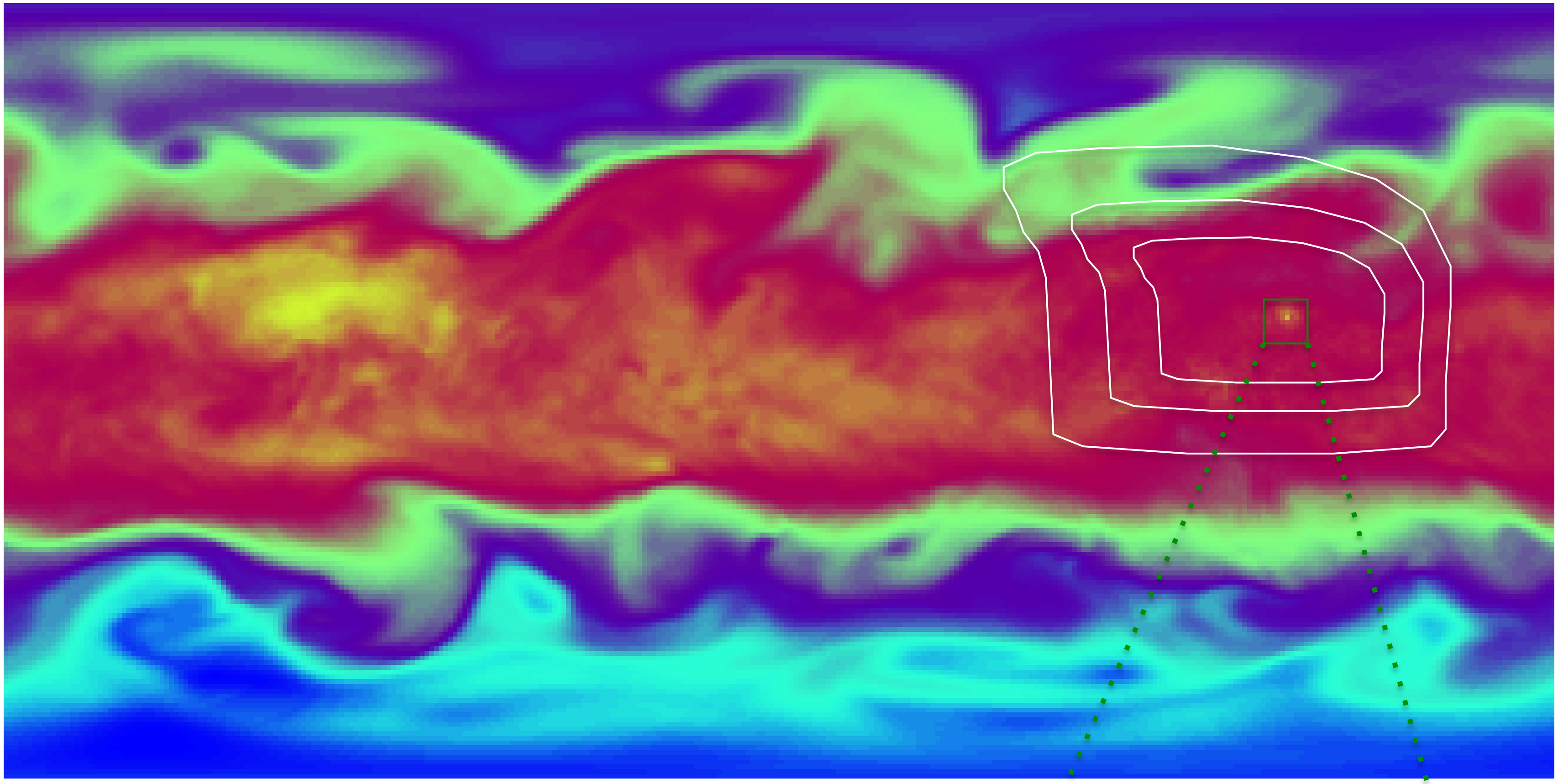
It's certainly complicated.

Does it even work?

Refined grid interpolation for Katrina

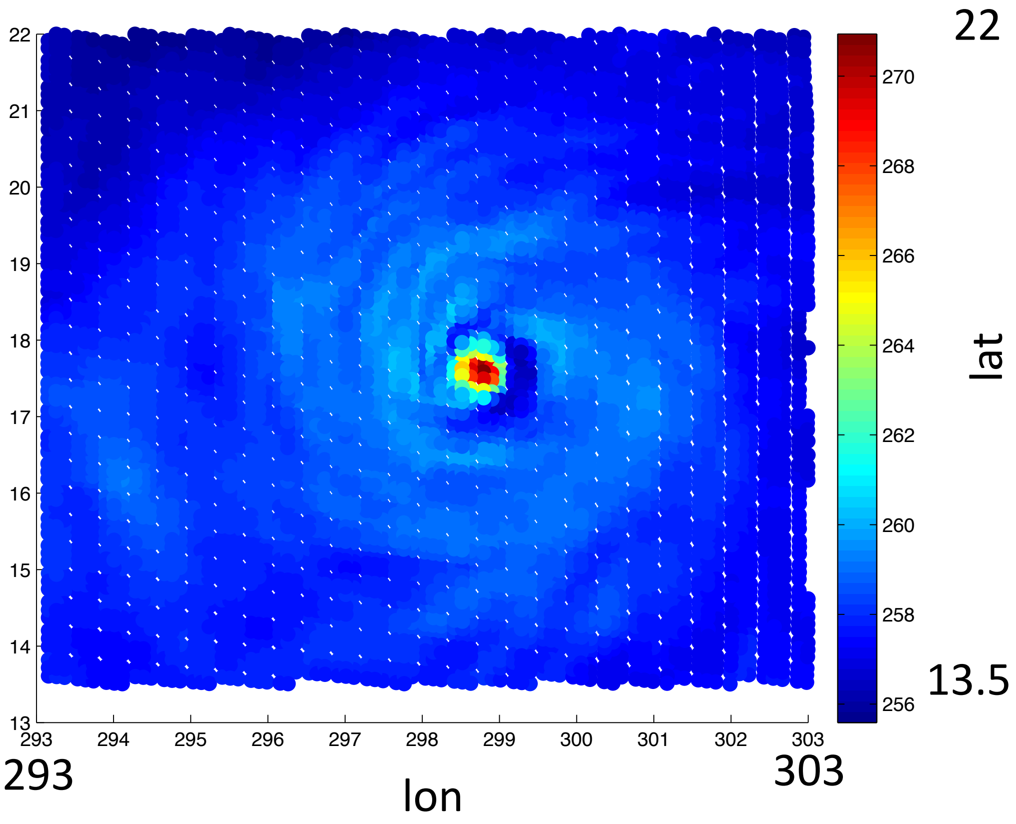
Colin Zarzycki (U Mich) provided the grid and case details for a test.

- ~1-degree global + 3 nested grids down to ~1/8-degree.
- Test method by interpolating SE-CAM forecast to a global, 1/10-degree grid.

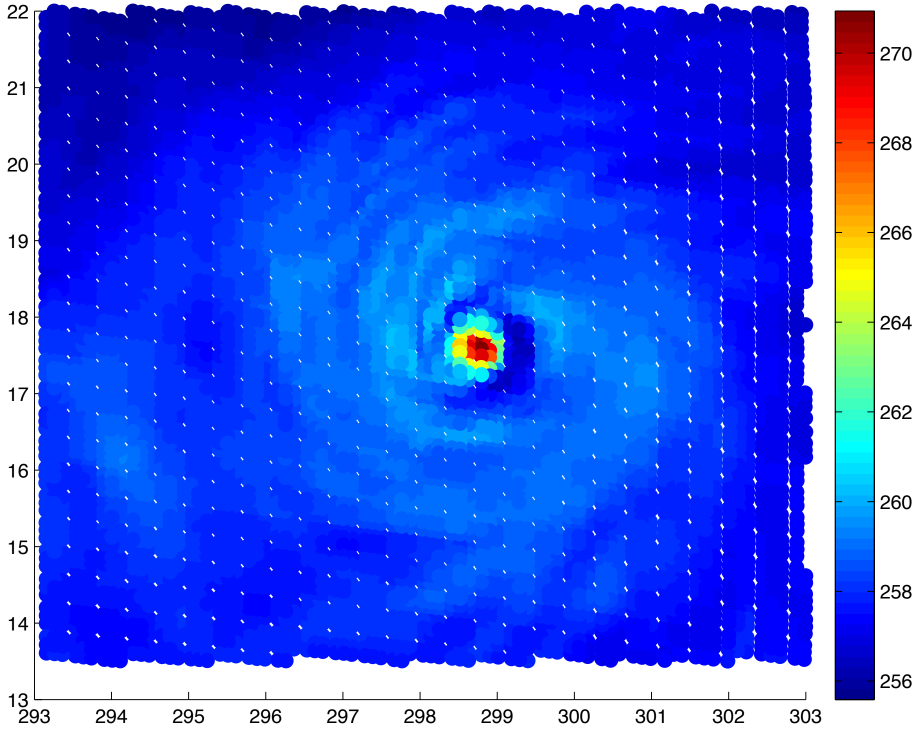


T 400 hPa

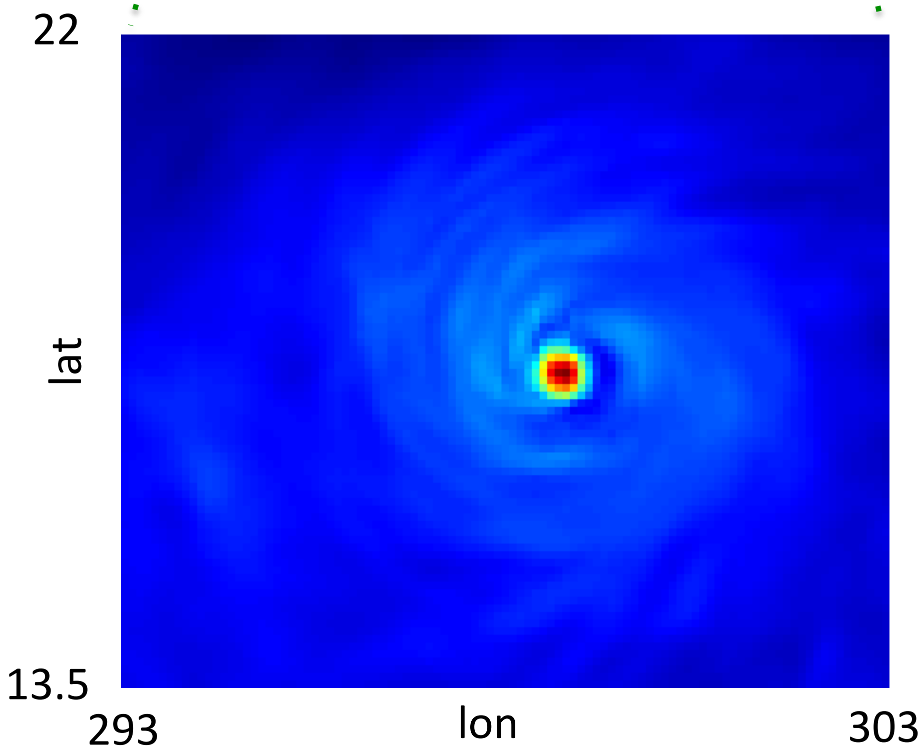
1/8-degree SE-CAM output



1/8-degree SE-CAM output



1/10-degree interpolation



Preliminary assimilations with this grid and case indicate that refined grids can be used in data assimilation using SE-CAM.

“Perfect Model” Assimilation

- A free run of the ~1-degree (ne30, np4) SE-CAM is used as the truth.
- T, U, and V, observations are taken on 15 levels at 600 approximately uniformly distributed locations at 0Z and 12Z.
- Random, reasonable, observational error is added to each.
- An assimilation is performed using 80 members in CESM1_1_1’s multi-instance capability.
- Get the initial ensemble from random, round-off, T perturbations of a single model state.
- I allotted 5 nodes/instance, 400 nodes total, and it finished each forecast in ~11 min and each assimilation in ~8 min. 1 node/instance would have worked.

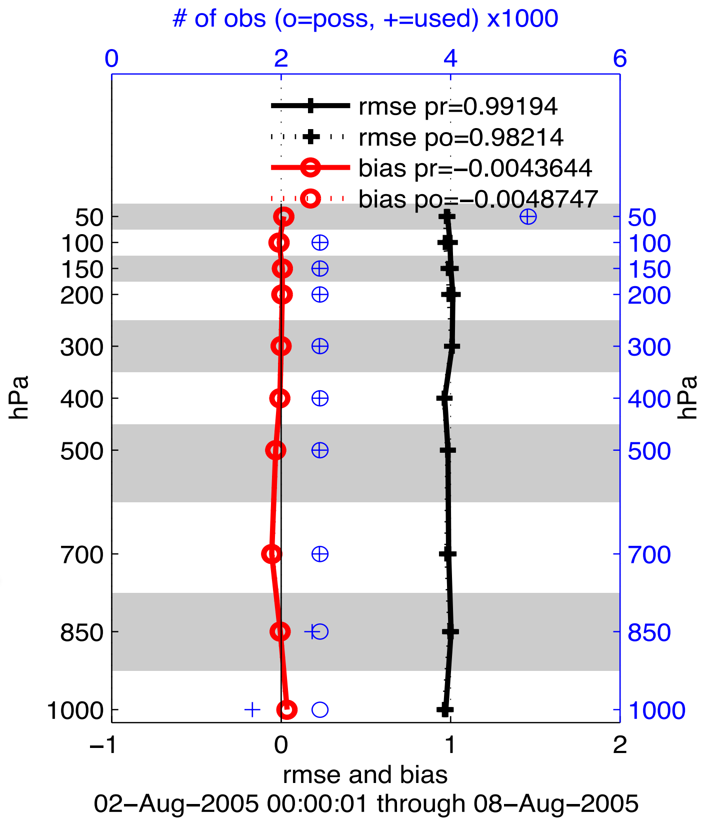
"Perfect Model" Assimilation Example of Results

It's a fast and easy test;

few observations,

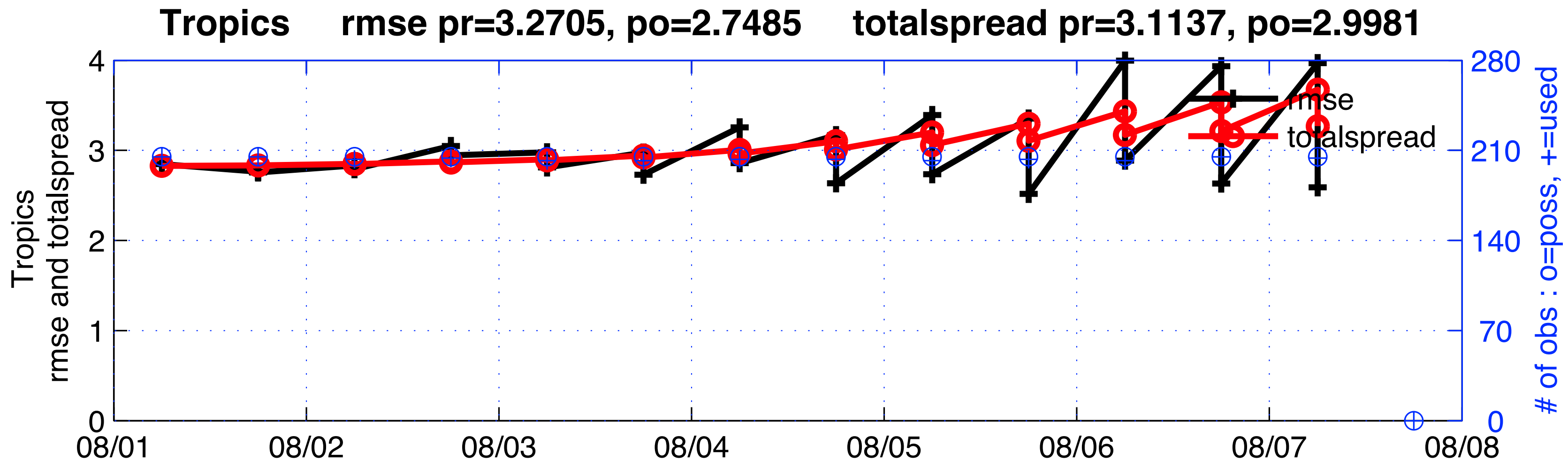
truth = a natural model state.

Not enough days yet to evaluate and claim success.



Temperature bias at obs locations

200 hPa horizontal wind error.



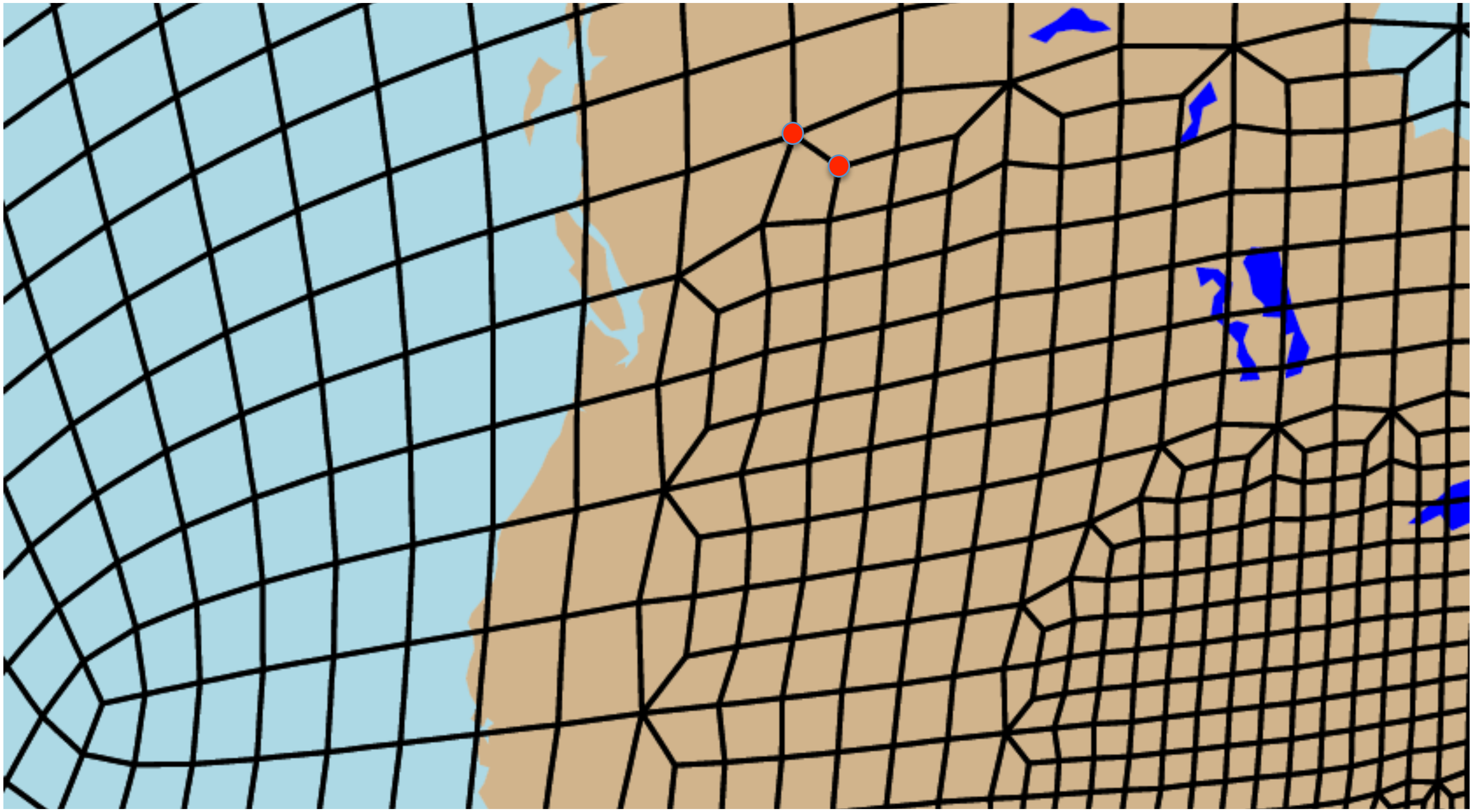
Summary

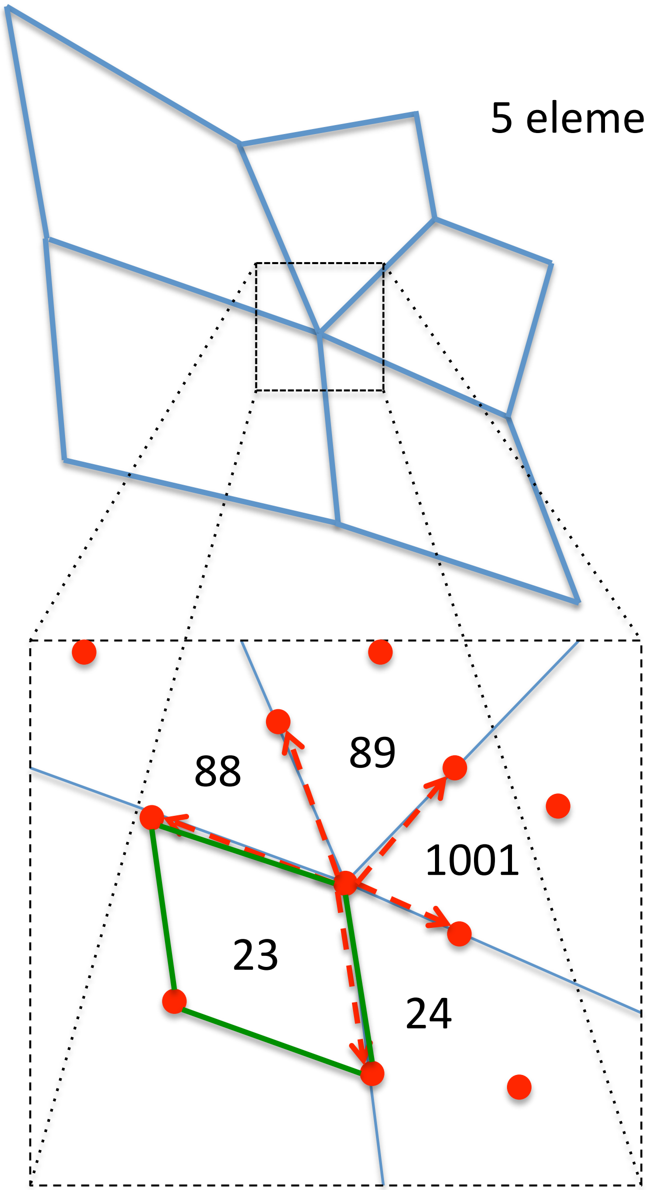
- Tools exist to use SE-CAM in data assimilation.
- Refined grids are not a mechanical problem.
- Current interpolation scheme works, but efficiency needs to be compared with other schemes.

Future Work

- Timing tests vs. other interpolation schemes.
- Assimilations with comprehensive real observation sets.
- Comparison against FV assimilations.
- The usual data assimilation pursuits:
  - ❑ generate initial conditions for forecast studies
  - ❑ identify model biases
  - ❑ sensitivity studies of atmospheric phenomena
  - ❑ observation system simulation experiments
  - ❑ . . .

Refined mesh grid: all elements are 4-sided, but 3-6 elements may share a node.



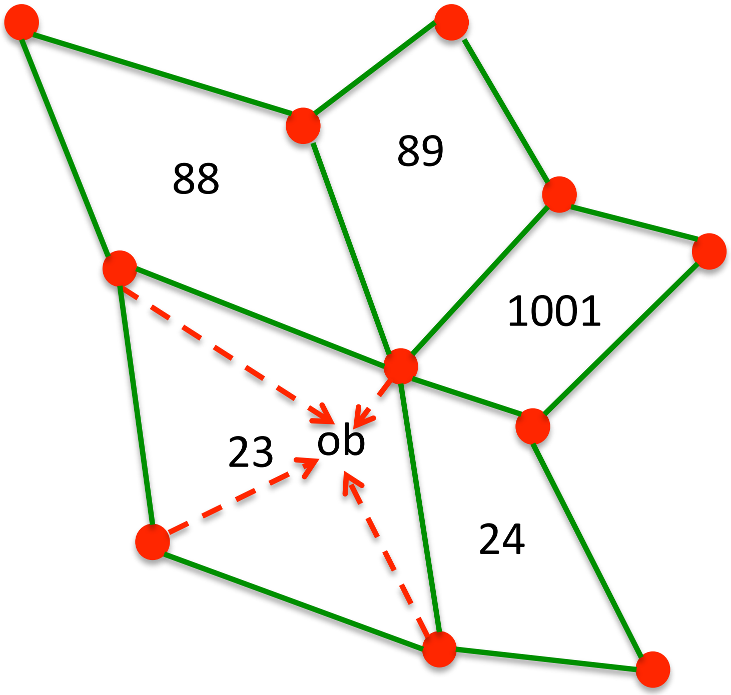


5 elements at the boundary between coarse grid and refined grid.

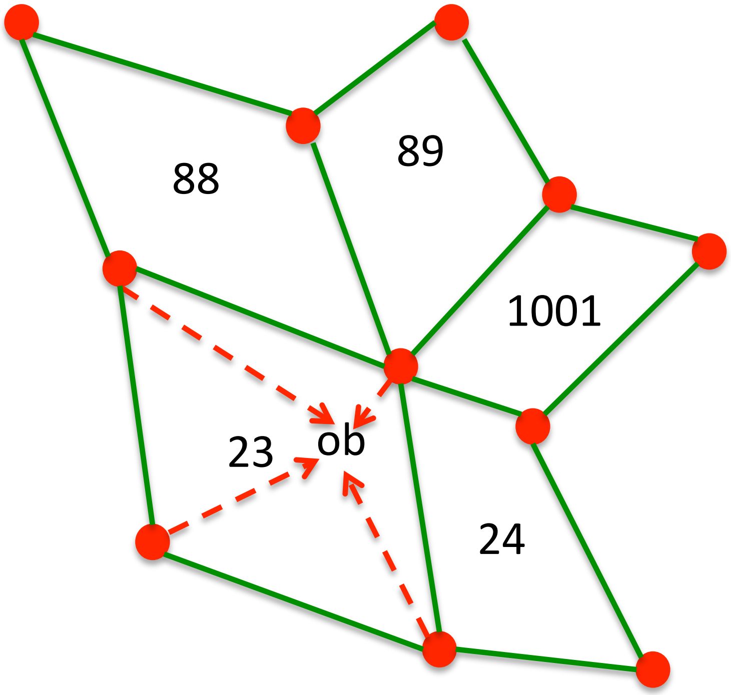
Each element has nodes on its corners, edges, and interior.
The nodes form a 3x3 quilt of quadrilaterals within each element (shown earlier).

For each node, store the the number of quads and the list of quads which use it as a corner, once before any assimilation.

The Goal; interpolate fields from nodes to an observation location



A Method

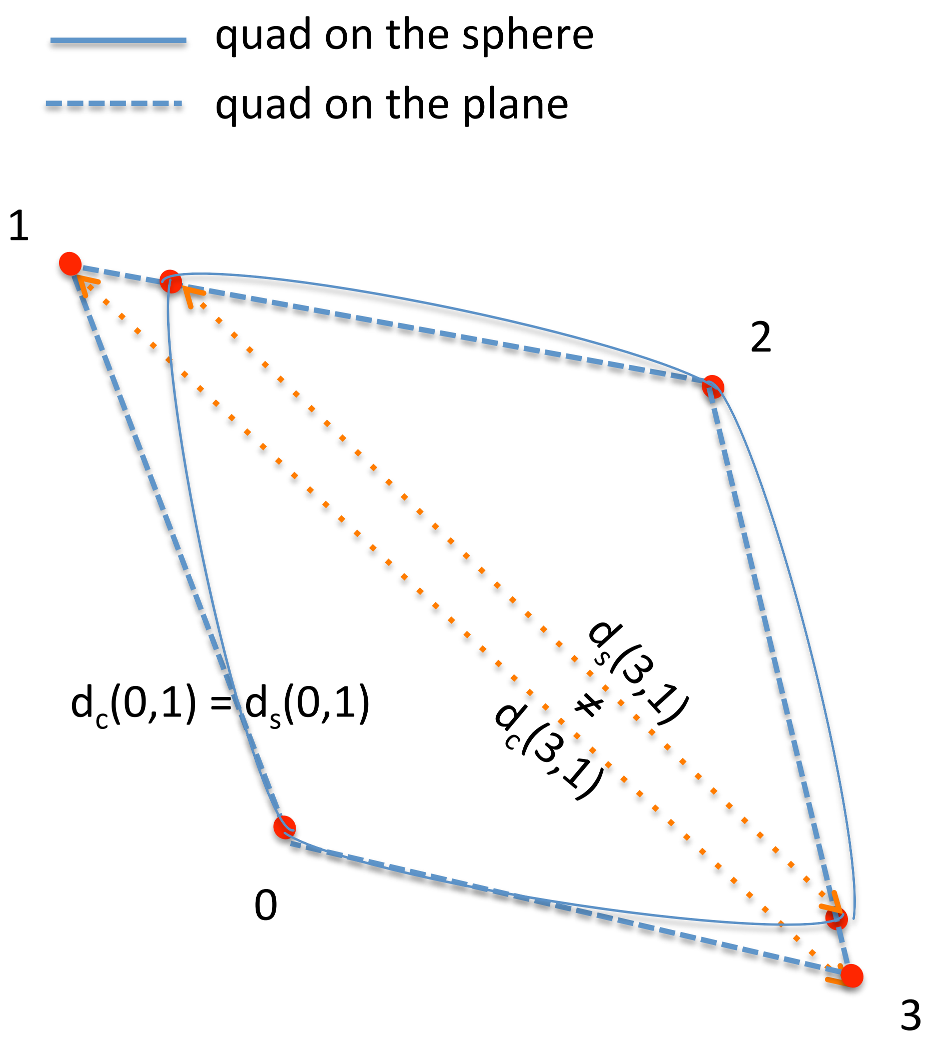


1) Identify the node closest to the ob.
   a) Use DART's get_close mechanism to identify candidates.
   b) Sort them by distance (along great circles).

2) Identify the quad which has the 'closest' node as one corner and contains the ob. (It may be necessary to use the 2nd closest node in some cases.)

3) Map the quad corners' locations and ob location (all in (lon,lat) coordinates) to the unit square, where interpolation is easy.

Details of mapping to (x,y)

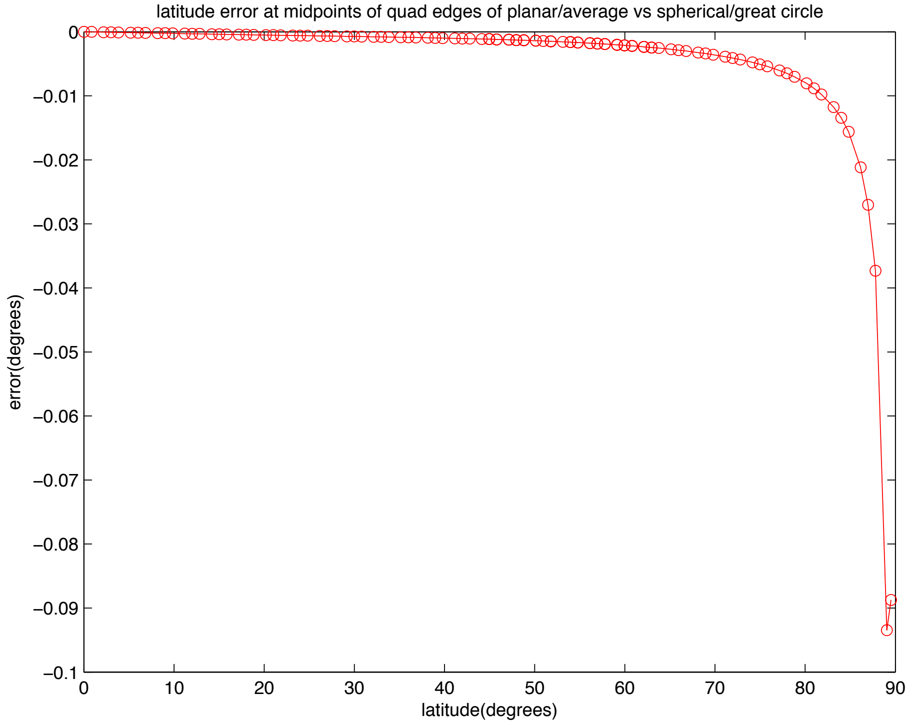


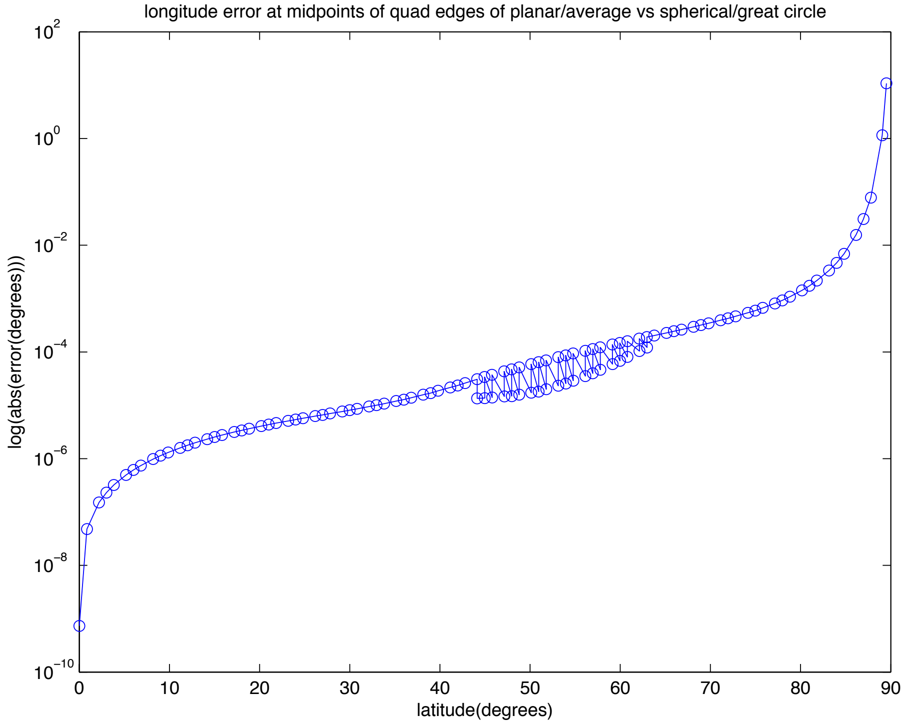
- Bearings and distances from origin to corners 1, 2, and 3 are preserved. So all obs inside of sides 0-1 and 0-3 of the sphere quad will be inside the plane quad.
- But other distances are distorted and obs inside sides 1-2 and 2-3 may not be inside the plane quad.

But

This implies a linear interpolation of longitude and latitude between the corners, which is inconsistent with the quad 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) quad edge midpoints, as calculated by linear interpolation, relative to a spherical coordinate mid-point formula.

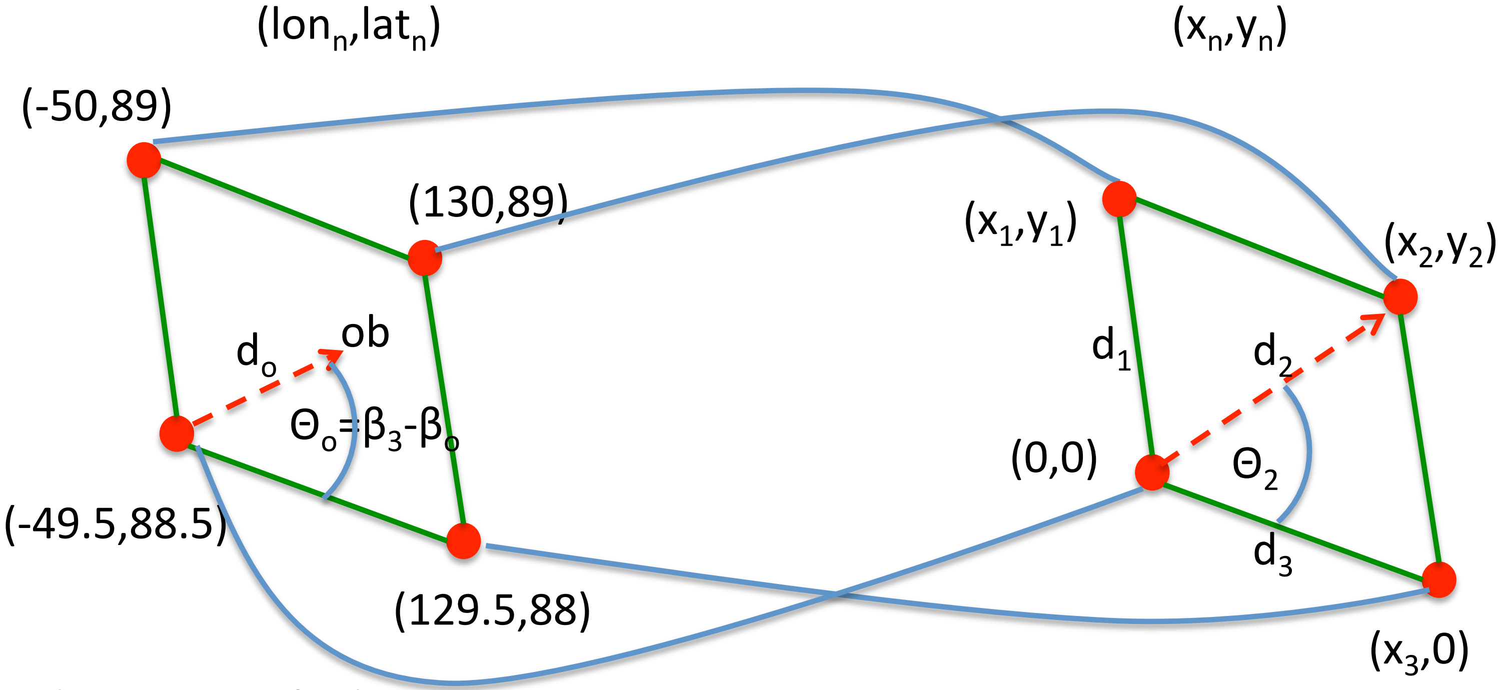




It’s most likely near the poles, but it can happen anywhere.

An intermediate map solves the problem.

Map the curved (lon,lat) space of each quad into its own flat (x,y) space of approximately the same shape.



d = great circle distance.

β = the bearing; the direction from one point on a sphere to another, along a great circle. North is 0.

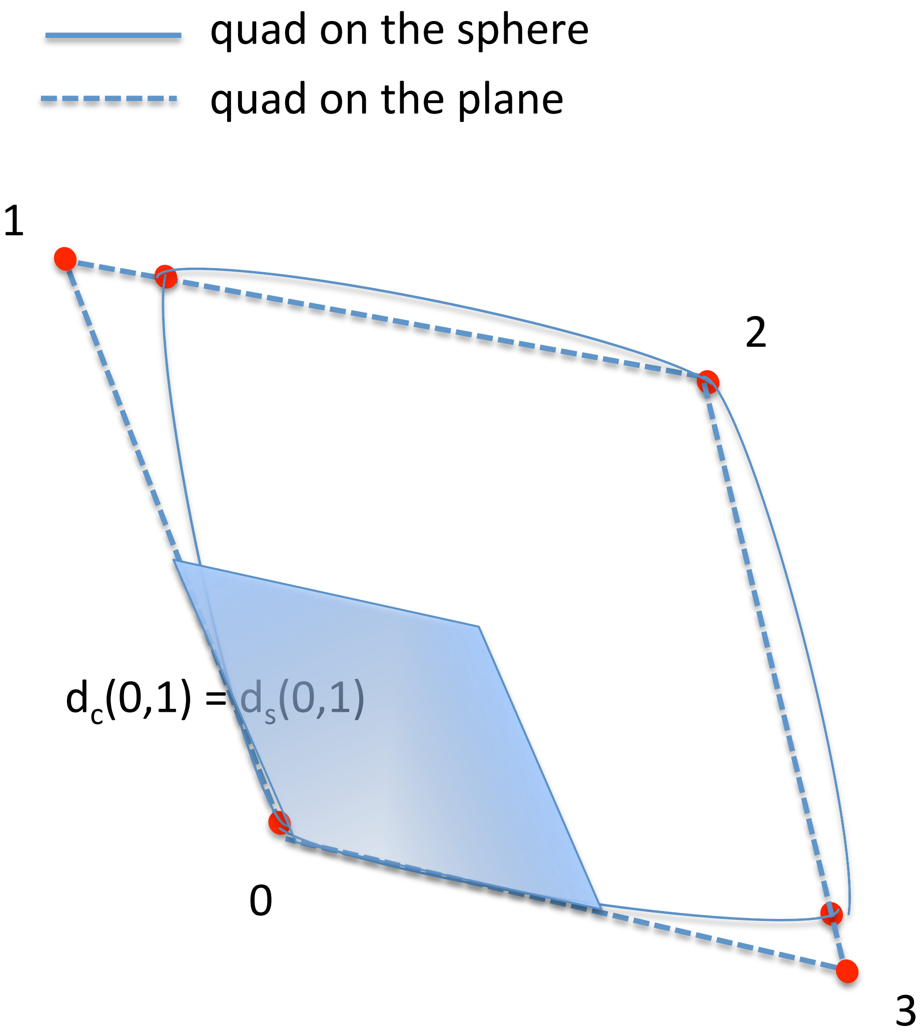
$$\beta = \mathrm{atan2}(\ \sin(\Delta\lambda)*\cos(\phi_2),\quad \cos(\phi_1)*\sin(\phi_2) - \sin(\phi_1)*\cos(\phi_2)*\cos(\Delta\lambda)\ )$$

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

The (d,Θ) formulation guarantees that the ob will be in the (flattened) quad.

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

Robust solution to planar mapping distance distortion



- Distances, other than to the nodes, are distorted and obs inside sides 1-2 and 2-3 may not be inside the plane quad.
- 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.
- Cheaper than distance correction during assimilation.
- And the differences are . . .

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. This mapping is stored as only 6 numbers: $a_{1,2,3}$, $b_{1,2}$, and Θ_3.

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

② During an assimilation identify the node which is closest to the ob, and is also a corner of the quad that contains the ob.

③ Search the 3-6 quads that use the node as a corner to see which contains the ob:

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

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

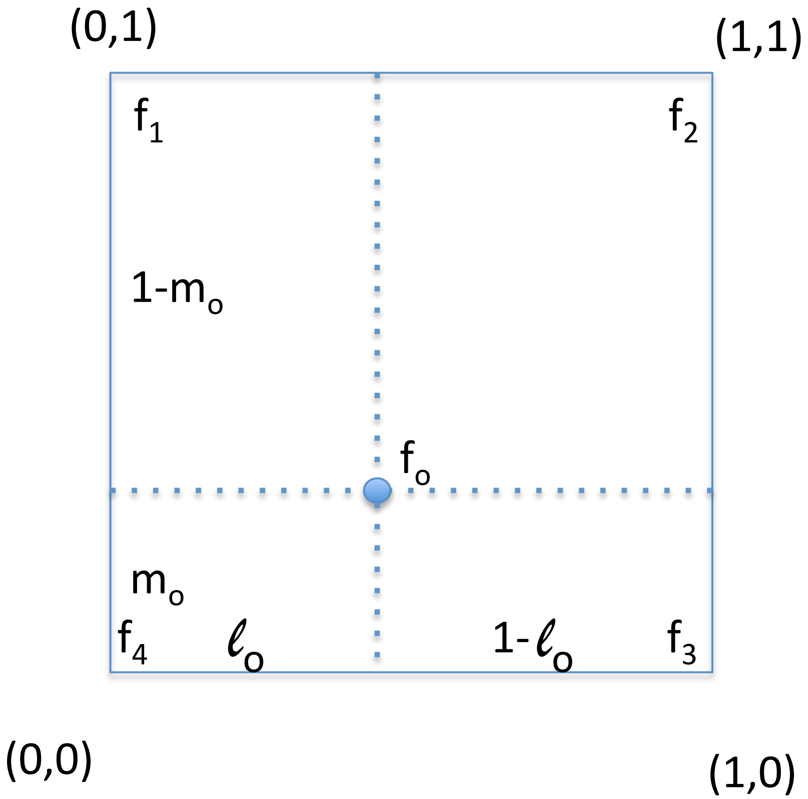
If $0 \leq m \leq 1$ and $0 \leq \ell \leq 1$ then we've found the containing quad.

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

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

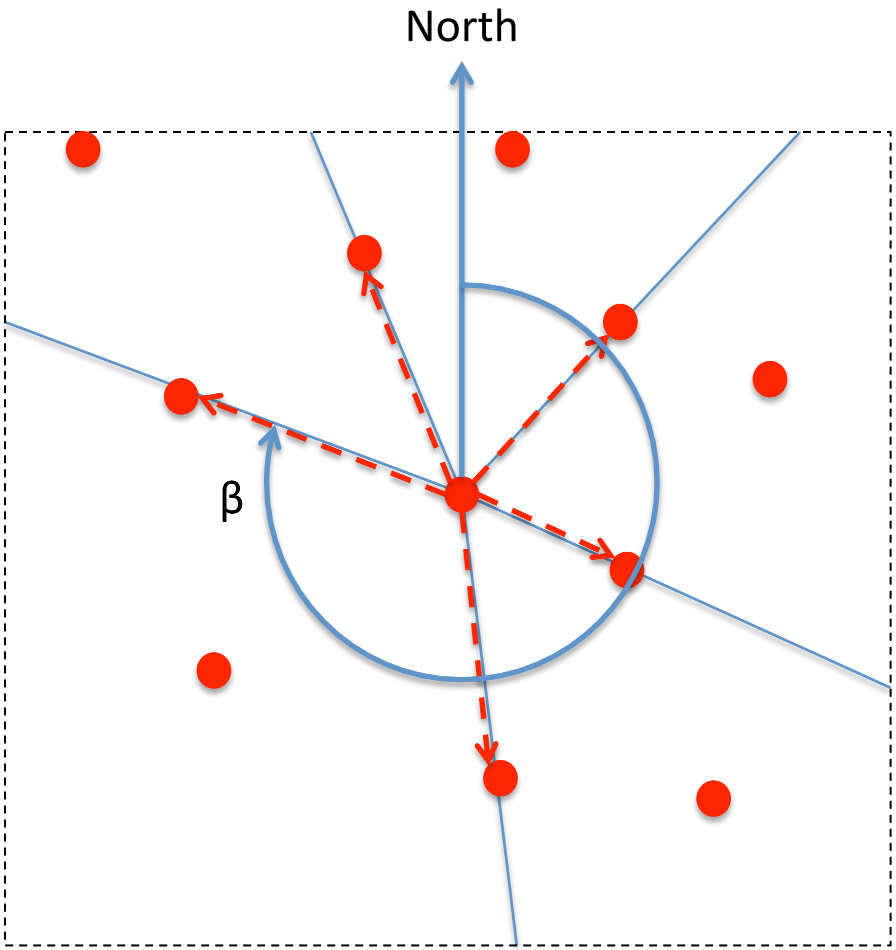
④ Use l and m as weights to interpolate the field values at the corners, f_n, to the ob location.

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



Digression 1: 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

β = atan2($\sin(\Delta\lambda)*\cos(\phi_2)$, $\cos(\phi_1)*\sin(\phi_2) - \sin(\phi_1)*\cos(\phi_2)*\cos(\Delta\lambda)$)

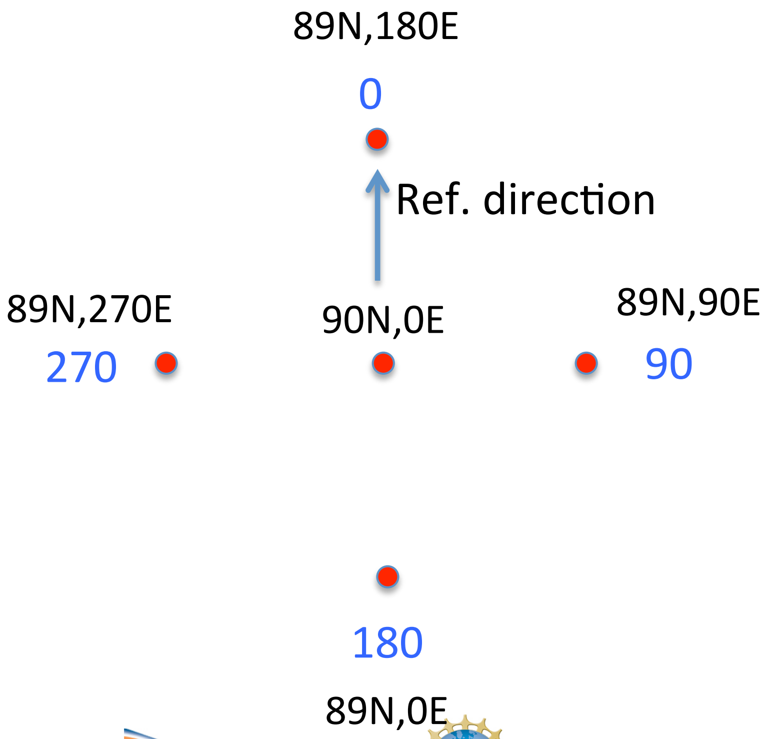
λ = longitude Φ = latitude 1=starting point 2 = destination

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

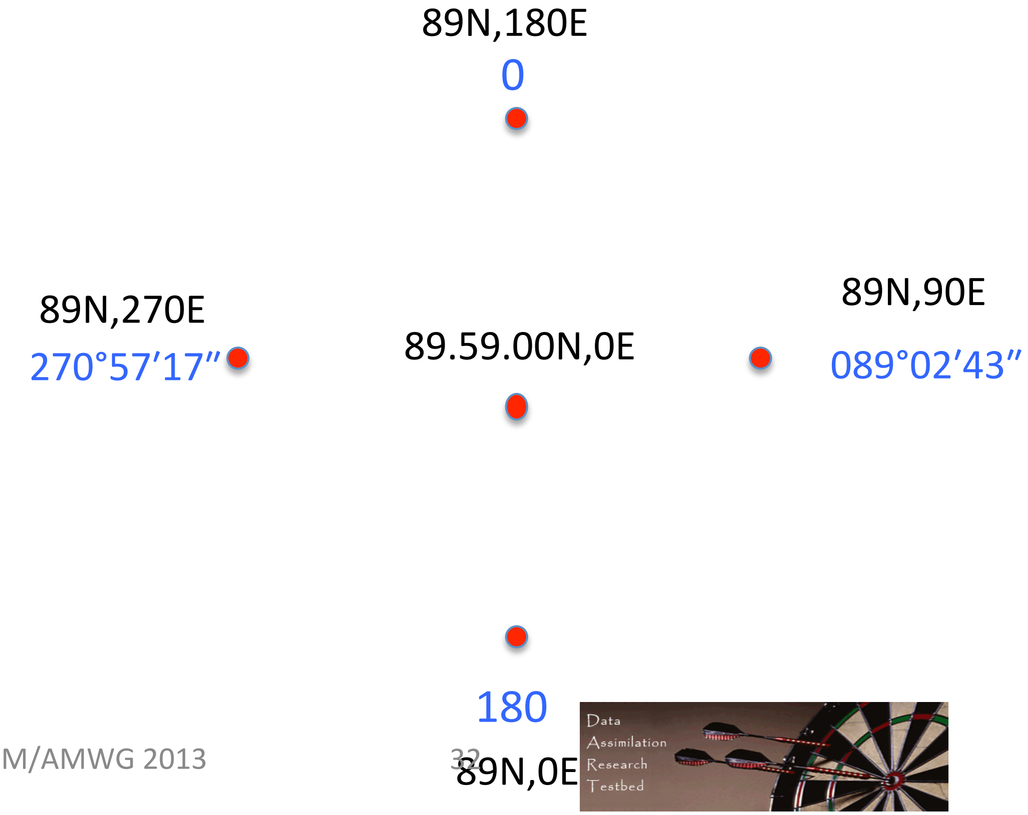
What’s the bearing at and near the poles?

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 quad, so there won’t be a confusion of reference directions.

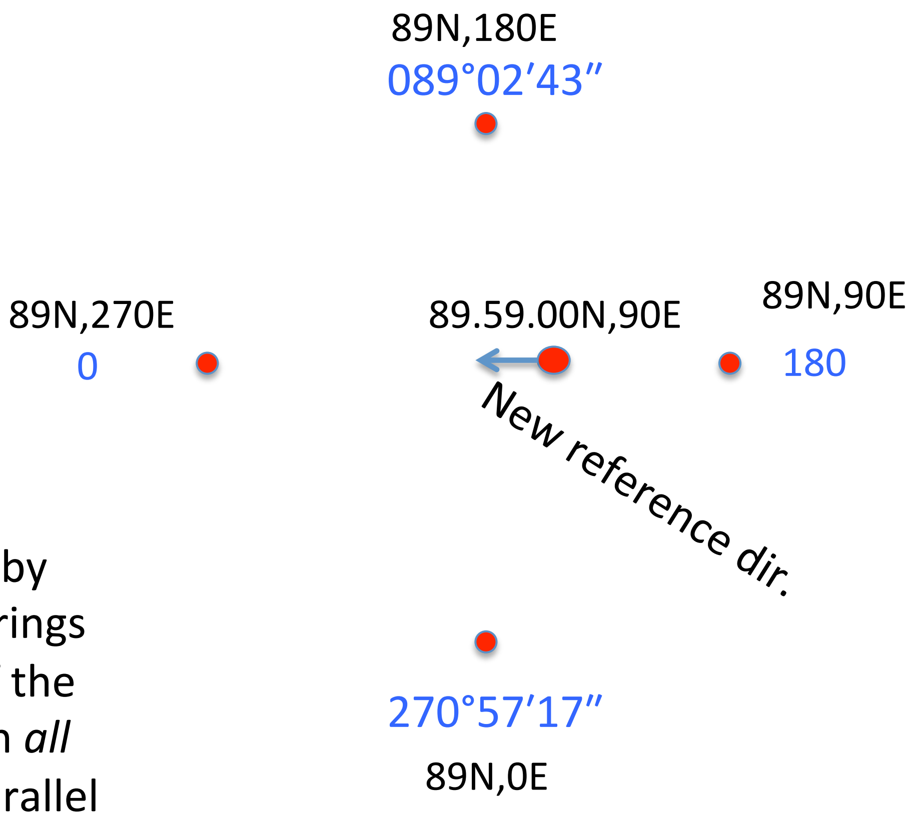
Initial point at the N pole (and ‘0E’)



Initial point displaced towards 0E



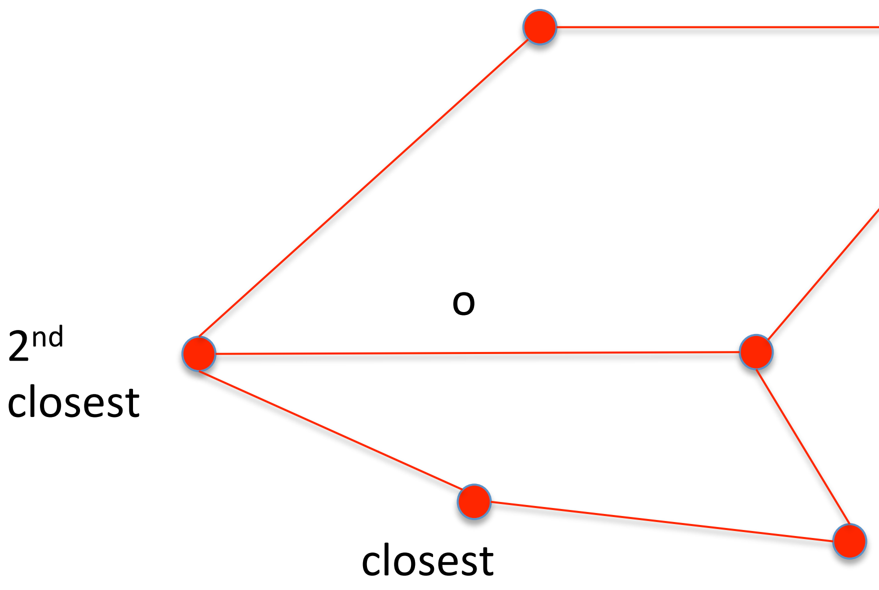
Initial point displaced toward 90E



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.

Digression 2: 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 quad 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 (double) mapping the ob location into each of the quads that use the closest node as a corner. If they all fail, do the same for the 2nd closest node.