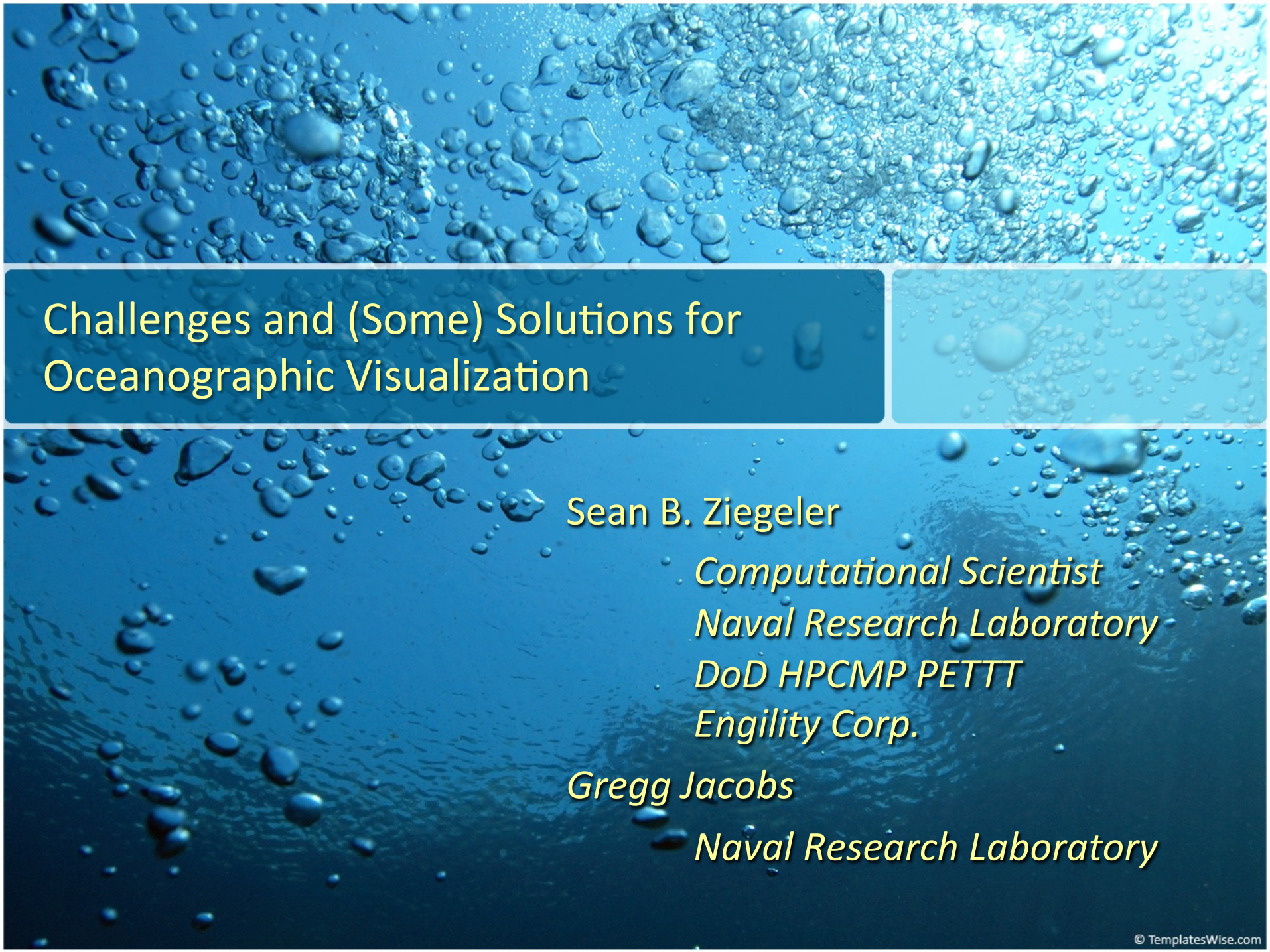


Challenges and (Some) Solutions for Oceanographic Visualization

Sean B. Ziegeler

Computational Scientist
Naval Research Laboratory
DoD HPCMP PETTT
Engility Corp.

Gregg Jacobs

Naval Research Laboratory

Overview

- Viz Utilization in "METOC" Behind the Curve
- Why it is Catching Up
- Case Studies
  - Regional Arctic Simulation Model (RASM)
  - MIT Global Circulation Model (MITgcm)
  - Navy Coastal Ocean Model (NCOM)
- What's Still Needed
- The Future

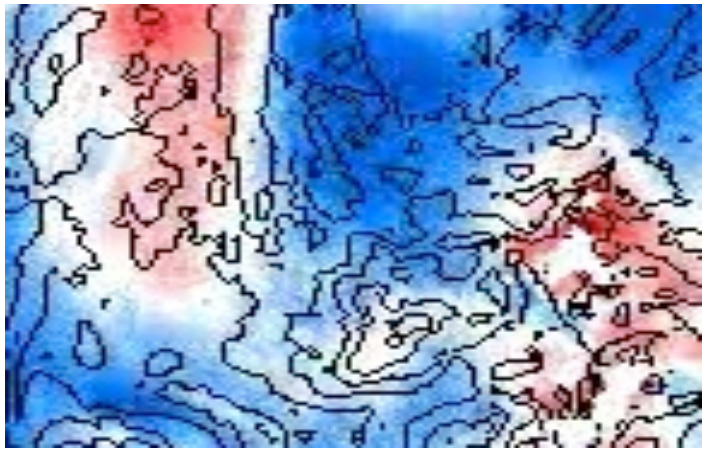
Viz Behind the Curve…Why?

- The Hydrostatic Assumption

$$\frac{h}{L} << 1$$

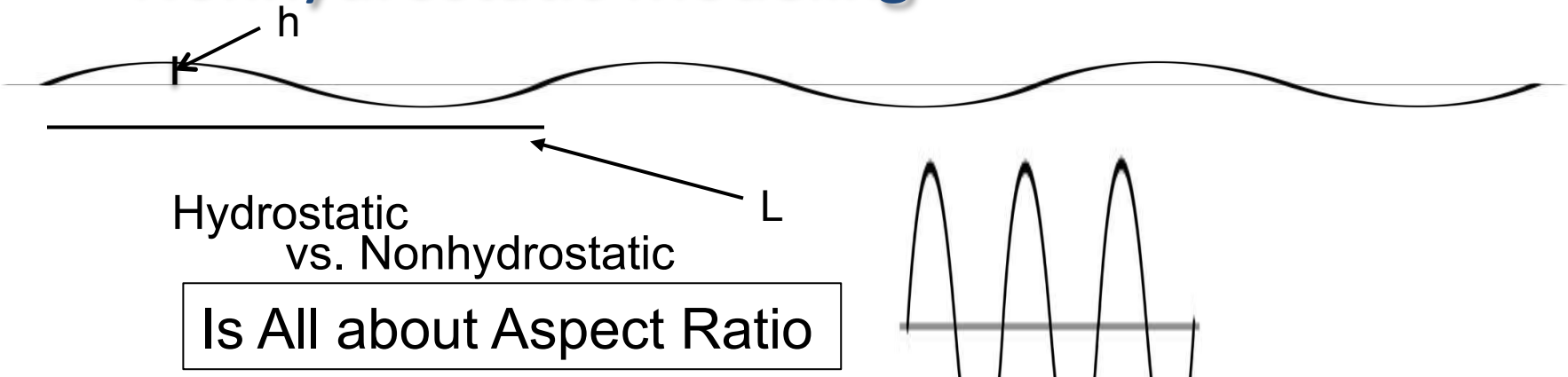
  - removes "w"

- 2D Plotting Culture
  - "good enough"
- File Formats



Viz Advancing for the Ocean

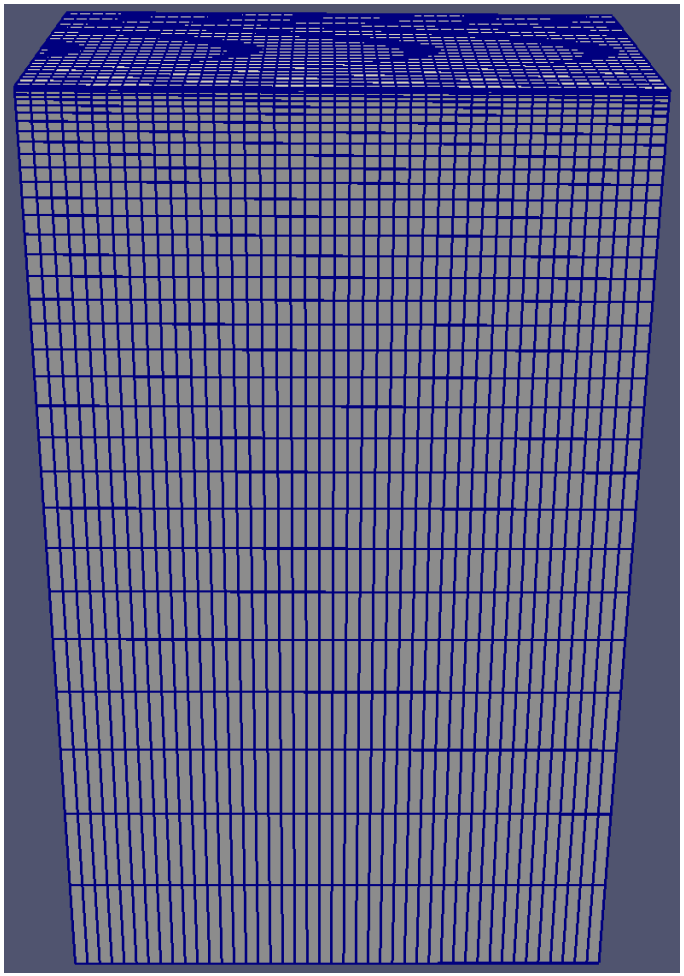
- Nonhydrostatic Modeling



- Vertical Structures
  - e.g., internal waves, cells

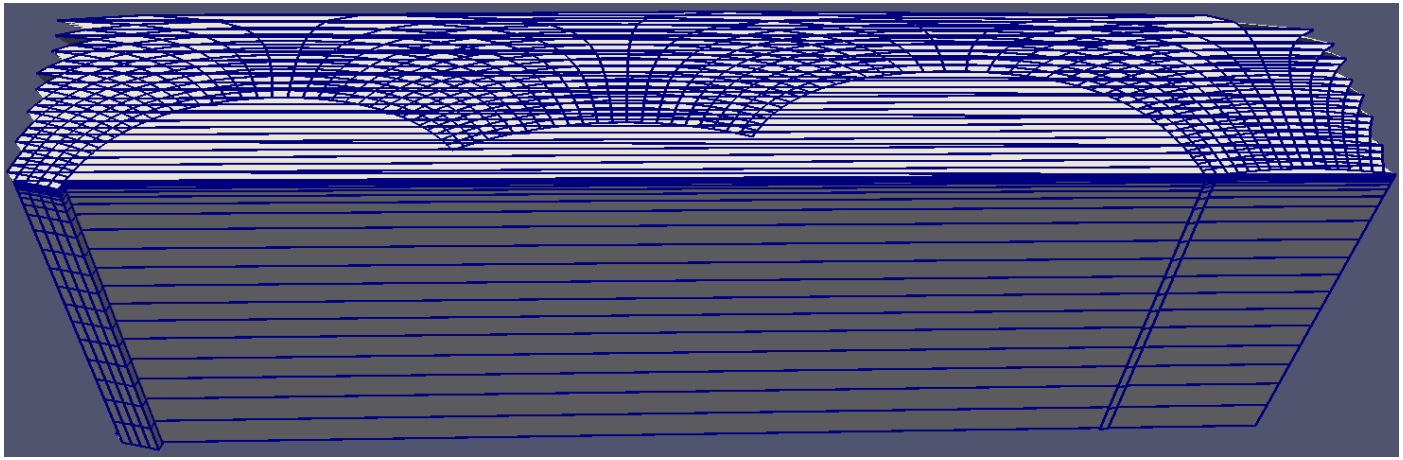
Viz Advancing for the Ocean (cont)

- Better File and Grid Support
  - NetCDF (and ALL its flavors)
  - Coordinate oddities, layers
  - Rectilinear Grids (faster)
  - Curvilinear Grids
- Python Scripting!
  - To fill in the support gaps



Case Study: RASM

- Uses POP as ocean component
- Arctic region → polar grid, in lon/lat coords



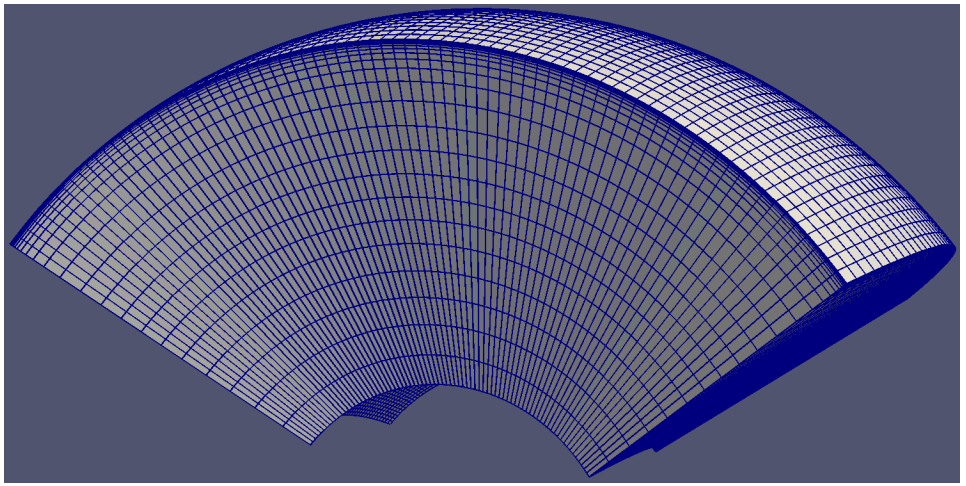
Case Study: RASM (cont)

ParaView:

- Spherical

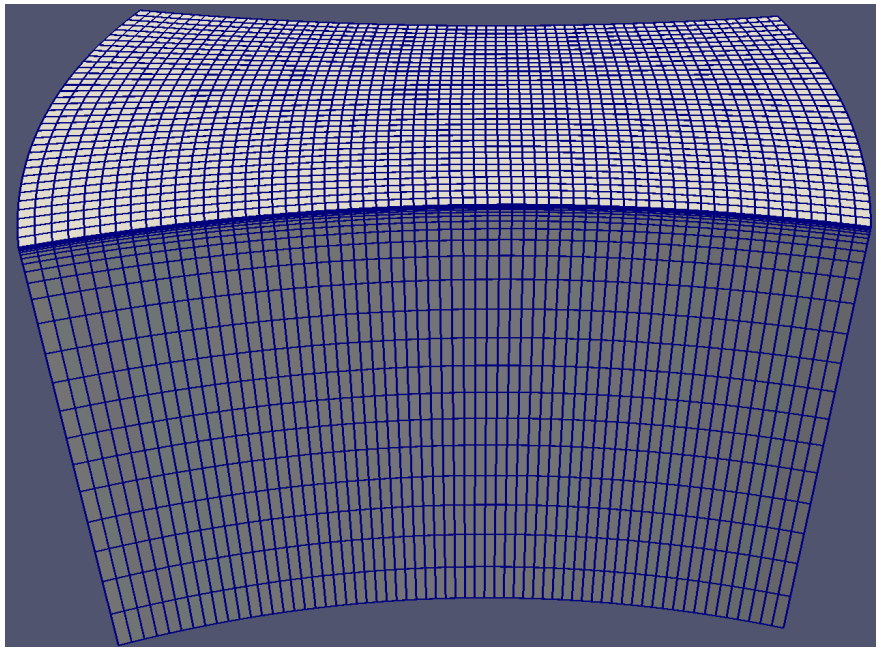
(in the reader)

  - horizontal flow particle issues



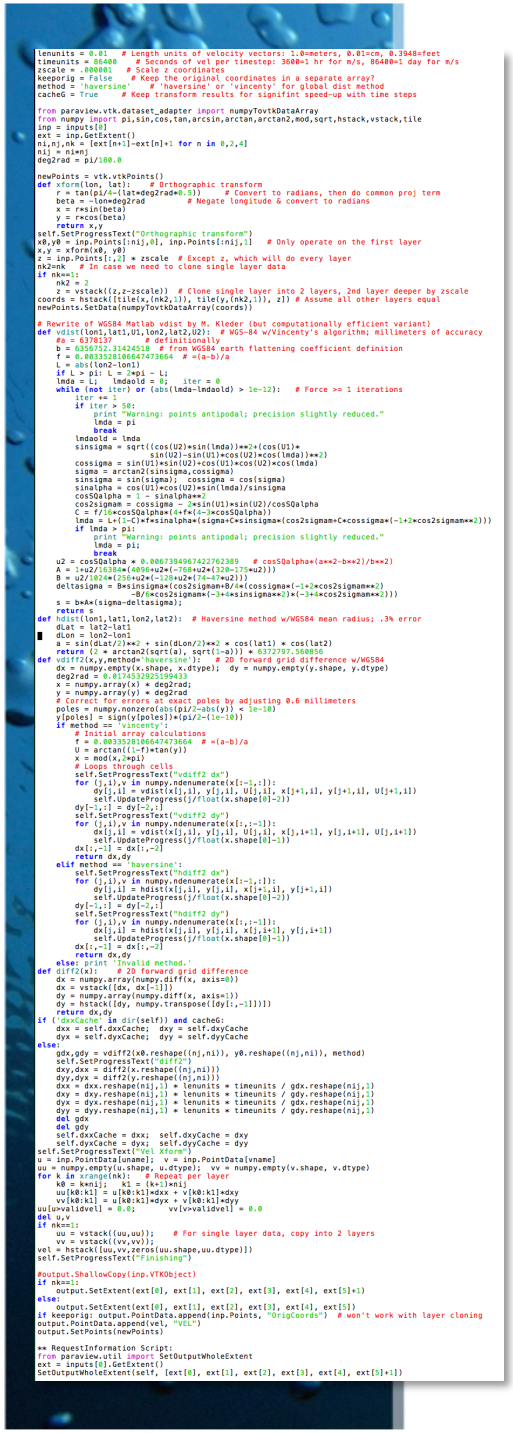
- Orthographic

(WGS-84 with Vincenty (mm accurate) and Haversine (fast) algorithms



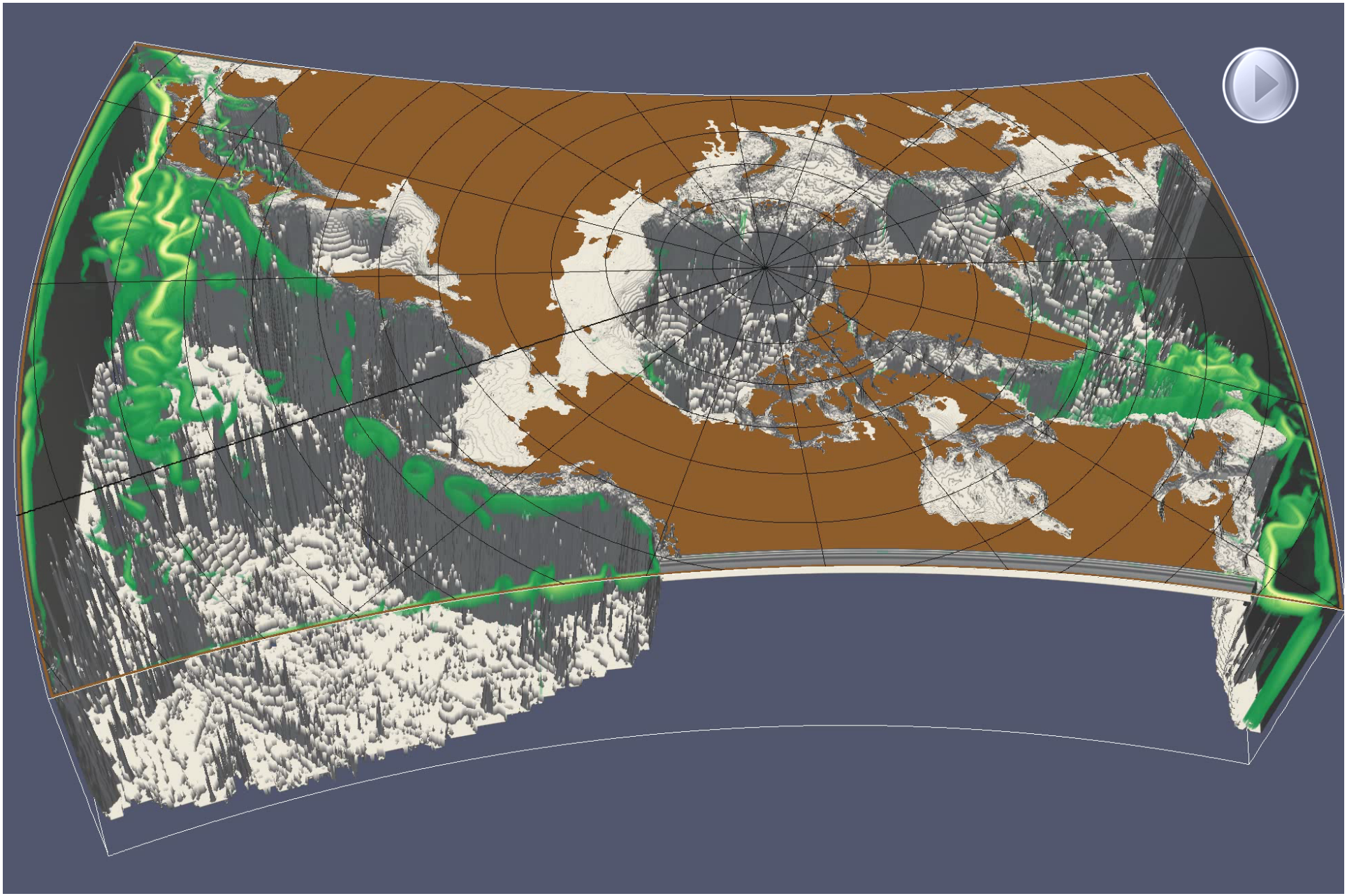
VisIt:

- Projections: flat only?



```
lenunits = 0.01   # Length units of velocity vectors: 1.0=meters, 0.01=cm, 0.3048=feet
timeunits = 86400    # Seconds of vel per timestep: 3600=1 hr for m/s, 86400=1 day for m/s
zscale = .000001   # Scale z coordinates
keeporig = False     # Keep the original coordinates in a separate array?
method = 'haversine'    # 'haversine' or 'vincenty' for global dist method
cacheG = True      # Keep transform results for signifint speed-up with time steps

from paraview.vtk.dataset_adapter import numpyTovtkDataArray
from numpy import pi,sin,cos,tan,arcsin,arctan,arctan2,mod,sqrt,hstack,vstack,tile
inp = inputs[0]
ext = inp.GetExtent()
ni,nj,nk = [ext[n+1]-ext[n]+1 for n in 0,2,4]
nij = ni*nj
deg2rad = pi/180.0

newPoints = vtk.vtkPoints()
def xform(lon, lat):    # Orthographic transform
    r = tan(pi/4-(lat*deg2rad*0.5))    # Convert to radians, then do common proj term
    beta = -lon*deg2rad                # Negate longitude & convert to radians
    x = r*sin(beta)
    y = r*cos(beta)
    return x,y
self.SetProgressText("Orthographic transform")
x0,y0 = inp.Points[:nij,0], inp.Points[:nij,1]   # Only operate on the first layer
x,y = xform(x0, y0)
z = inp.Points[:,2] * zscale  # Except z, which will do every layer
nk2=nk   # In case we need to clone single layer data
if nk==1:
    nk2 = 2
    z = vstack((z,z-zscale))  # Clone single layer into 2 layers, 2nd layer deeper by zscale
coords = hstack((tile(x,(nk2,1)), tile(y,(nk2,1)), z)) # Assume all other layers equal
newPoints.SetData(numpyTovtkDataArray(coords))

# Rewrite of WGS84 Matlab vdist by M. Kleder (but computationally efficient variant)
def vdist(lon1,lat1,U1,lon2,lat2,U2):  # WGS-84 w/Vincenty's algorithm; millimeters of accuracy
    #a = 6378137    # definitionally
    b = 6356752.314245518  # from WGS84 earth flattening coefficient definition
    f = 0.003352810664747366  # =(a-b)/a
    L = abs(lon2-lon1)
    if L > pi: L = 2*pi - L;
    lmda = L;   lmdaold = 0;   iter = 0
    while (not iter) or (abs(lmda-lmdaold) > 1e-12):   # Force >= 1 iterations
        iter += 1
        if iter > 50:
            print "Warning: points antipodal; precision slightly reduced."
            lmda = pi
            break
        lmdaold = lmda
        sinsigma = sqrt((cos(U2)*sin(lmda))**2+(cos(U1)*
                        sin(U2)-sin(U1)*cos(U2)*cos(lmda))**2)
        cossigma = sin(U1)*sin(U2)+cos(U1)*cos(U2)*cos(lmda)
        sigma = arctan2(sinsigma,cossigma)
        sinsigma = sin(sigma);  cossigma = cos(sigma)
        sinalpha = cos(U1)*cos(U2)*sin(lmda)/sinsigma
        cosSQalpha = 1 - sinalpha**2
        cos2sigmam = cossigma - 2*sin(U1)*sin(U2)/cosSQalpha
        C = f/16*cosSQalpha*(4+f*(4-3*cosSQalpha))
        lmda = L+(1-C)*f*sinalpha*(sigma+C*sinsigma*(cos2sigmam+C*cossigma*(-1+2*cos2sigmam**2)))
        if lmda > pi:
            print "Warning: points antipodal; precision slightly reduced."
            lmda = pi;
            break
    u2 = cosSQalpha * 0.006739496742276389   # cosSQalpha*(a**2-b**2)/b**2)
    A = 1+u2/16384*(4096+u2*(-768+u2*(320-175*u2)))
    B = u2/1024*(256+u2*(-128+u2*(74-47*u2)))
    deltasigma = B*sinsigma*(cos2sigmam+B/4*(cossigma*(-1+2*cos2sigmam**2)
                 -B/6*cos2sigmam*(-3+4*sinsigma**2)*(-3+4*cos2sigmam**2)))
    s = b*A*(sigma-deltasigma);
    return s
def hdist(lon1,lat1,lon2,lat2):  # Haversine method w/WGS84 mean radius; .3% error
    dLat = lat2-lat1
    dLon = lon2-lon1
    a = sin(dLat/2)**2 + sin(dLon/2)**2 * cos(lat1) * cos(lat2)
    return (2 * arctan2(sqrt(a), sqrt(1-a))) * 6372797.560856
def vdiff2(x,y,method='haversine'):   # 2D forward grid difference w/WGS84
    dx = numpy.empty(x.shape, x.dtype);  dy = numpy.empty(y.shape, y.dtype)
    deg2rad = 0.01745329251994333
    x = numpy.array(x) * deg2rad;
    y = numpy.array(y) * deg2rad
    # Correct for errors at exact poles by adjusting 0.6 millimeters
    poles = numpy.nonzero(abs(pi/2-abs(y)) < 1e-10)
    y[poles] = sign(y[poles])*(pi/2-(1e-10))
    if method == 'vincenty':
        # Initial array calculations
        f = 0.003352810664747366  # =(a-b)/a
        U = arctan((1-f)*tan(y))
        x = mod(x,2*pi)
        # Loops through cells
        self.SetProgressText("vdiff2 dx")
        for (j,i),v in numpy.ndenumerate(x[:-1,:]):
            dx[j,i] = vdist(x[j,i], y[j,i], U[j,i], x[j+1,i], y[j+1,i], U[j+1,i])
            self.UpdateProgress(j/float(x.shape[0]-2))
        dy[-1,:] = dy[-2,:]
        self.SetProgressText("vdiff2 dy")
        for (j,i),v in numpy.ndenumerate(x[:,:-1]):
            dx[j,i] = vdist(x[j,i], y[j,i], U[j,i], x[j,i+1], y[j,i+1], U[j,i+1])
            self.UpdateProgress(j/float(x.shape[0]-1))
        dx[:,-1] = dx[:,-2]
        return dx,dy
    elif method == 'haversine':
        self.SetProgressText("hdiff2 dx")
        for (j,i),v in numpy.ndenumerate(x[:-1,:]):
            dx[j,i] = hdist(x[j,i], y[j,i], x[j+1,i], y[j+1,i])
            self.UpdateProgress(j/float(x.shape[0]-2))
        dy[-1,:] = dy[-2,:]
        self.SetProgressText("hdiff2 dy")
        for (j,i),v in numpy.ndenumerate(x[:,:-1]):
            dx[j,i] = hdist(x[j,i], y[j,i], x[j,i+1], y[j,i+1])
            self.UpdateProgress(j/float(x.shape[0]-1))
        dx[:,-1] = dx[:,-2]
        return dx,dy
    else: print 'Invalid method.'
def diff2(x):    # 2D forward grid difference
    dx = numpy.array(numpy.diff(x, axis=0))
    dx = vstack((dx, dx[-1]))
    dy = numpy.array(numpy.diff(x, axis=1))
    dy = hstack((dy, numpy.transpose([dy[:,-1]])))
    return dx,dy
if ('dxxCache' in dir(self)) and cacheG:
    dxx = self.dxxCache;  dxy = self.dxyCache
    dyx = self.dyxCache;  dyy = self.dyyCache
else:
    gdx,gdy = vdiff2(x0.reshape((nj,ni)), y0.reshape((nj,ni)), method)
    self.SetProgressText("diff2")
    dxy,dxx = diff2(x.reshape((nj,ni)))
    dyy,dyx = diff2(y.reshape((nj,ni)))
    dxx = dxx.reshape(nij,1) * lenunits * timeunits / gdx.reshape(nij,1)
    dxy = dxy.reshape(nij,1) * lenunits * timeunits / gdy.reshape(nij,1)
    dyx = dyx.reshape(nij,1) * lenunits * timeunits / gdx.reshape(nij,1)
    dyy = dyy.reshape(nij,1) * lenunits * timeunits / gdy.reshape(nij,1)
    del gdx
    del gdy
    self.dxxCache = dxx;  self.dxyCache = dxy
    self.dyxCache = dyx;  self.dyyCache = dyy
self.SetProgressText("Vel Xform")
u = inp.PointData[uname];  v = inp.PointData[vname]
uu = numpy.empty(u.shape, u.dtype);  vv = numpy.empty(v.shape, v.dtype)
for k in xrange(nk):   # Repeat per layer
    k0 = k*nij;    k1 = (k+1)*nij
    uu[k0:k1] = u[k0:k1]*dxx + v[k0:k1]*dxy
    vv[k0:k1] = u[k0:k1]*dyx + v[k0:k1]*dyy
uu[u>validvel] = 0.0;        vv[v>validvel] = 0.0
del u,v
if nk==1:
    uu = vstack((uu,uu));    # For single layer data, copy into 2 layers
    vv = vstack((vv,vv));
vel = hstack((uu,vv,zeros(uu.shape,uu.dtype)))
self.SetProgressText("Finishing")

#output.ShallowCopy(inp.VTKObject)
if nk==1:
    output.SetExtent(ext[0], ext[1], ext[2], ext[3], ext[4], ext[5]+1)
else:
    output.SetExtent(ext[0], ext[1], ext[2], ext[3], ext[4], ext[5])
if keeporig: output.PointData.append(inp.Points, "OrigCoords")  # won't work with layer cloning
output.PointData.append(vel, "VEL")
output.SetPoints(newPoints)

** RequestInformation Script:
from paraview.util import SetOutputWholeExtent
ext = inputs[0].GetExtent()
SetOutputWholeExtent(self, [ext[0], ext[1], ext[2], ext[3], ext[4], ext[5]+1])
```

- Configuration (var names, vel units, time units, method)
- Conv to radians
- Single layer data gets cloned (for particles)
- WGS-84 grid transform (if method …)
  - Vincenty's algorithm (mm's of error)
    - …
    - …
    - …
  - Haversine algorithm (<0.3% error)
- Compute flow vector transform given grid transform
  - Vincenty
    - …
  - Haversine
- 2d forward grid difference
- Cache flow transform for future time steps
- Apply flow transform to every layer
- Set VTK/PV pipeline info

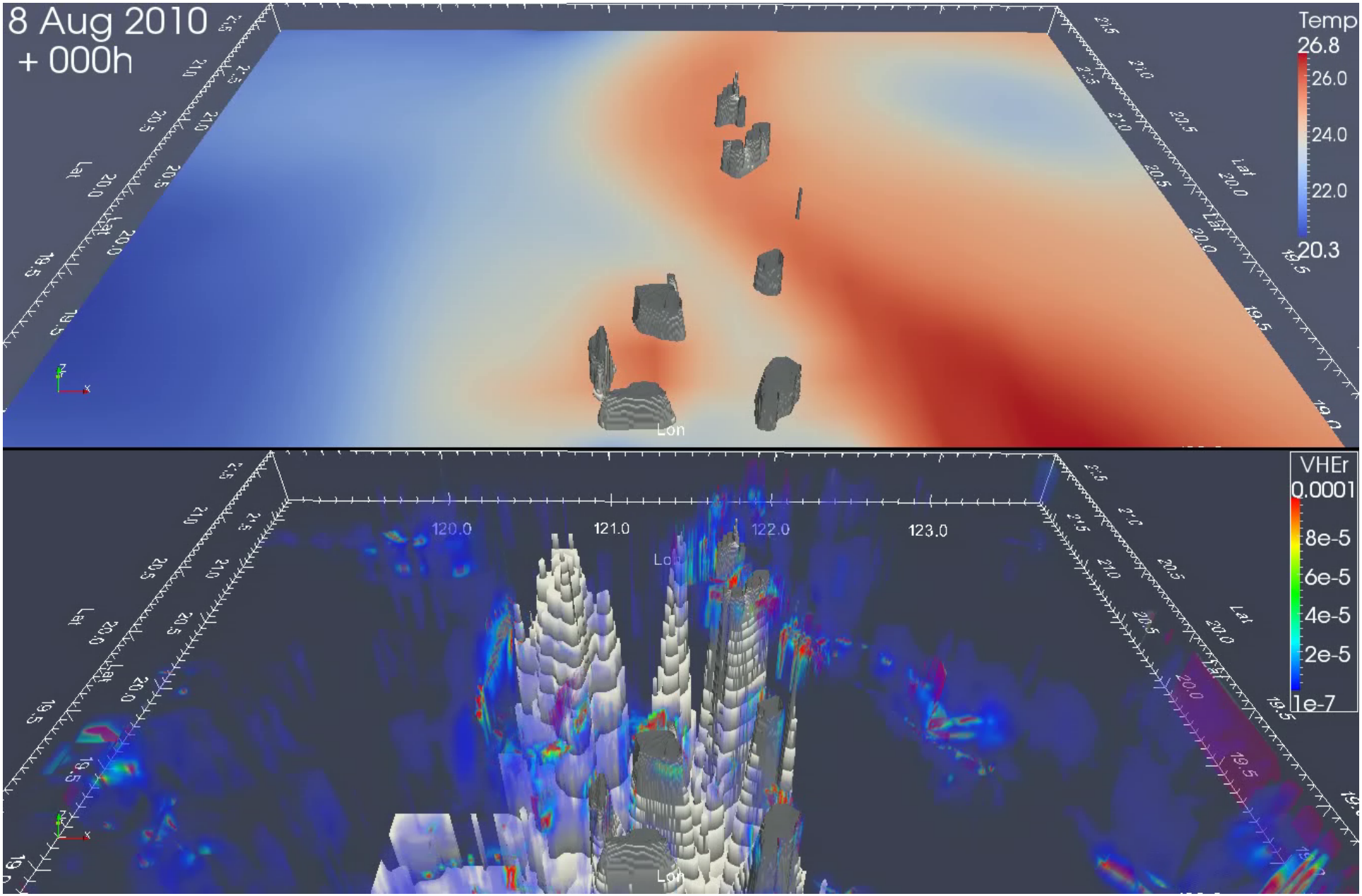


Case Study: MITgcm

- Nonhydrostatic model
- Internal waves can be highly nonhydrostatic in nature
  - Always have a 3D structure
  - Nonhydrostatic cases, 3D structure more pronounced
  - How to find them?

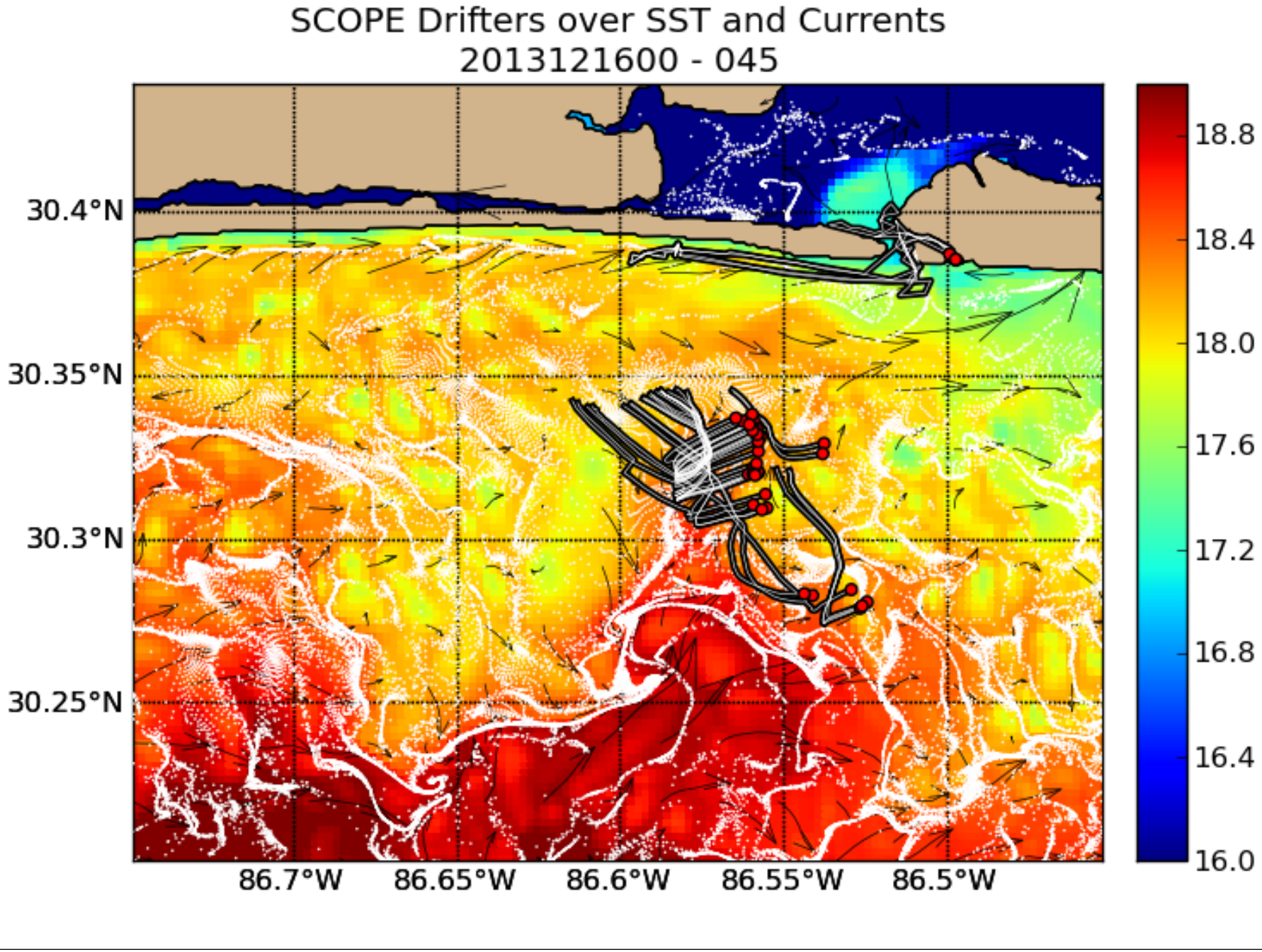
Case Study: MITgcm

- VHE calculations to isolate features
- Independent Python script
  - Load S,T,U,V,W
  - WKB Scaling (bvfreq)
  - Um = (U > eps) ? U : eps
  - vhe = W^2 / Um^2 * wkb^2
  - Volume rendering


8 Aug 2010
+ 000h
Temp
26.8
26.0
24.0
22.0
20.3
Lon
Lat
VHEr
0.0001
8e-5
6e-5
4e-5
2e-5
1e-7
120.0
121.0
122.0
123.0

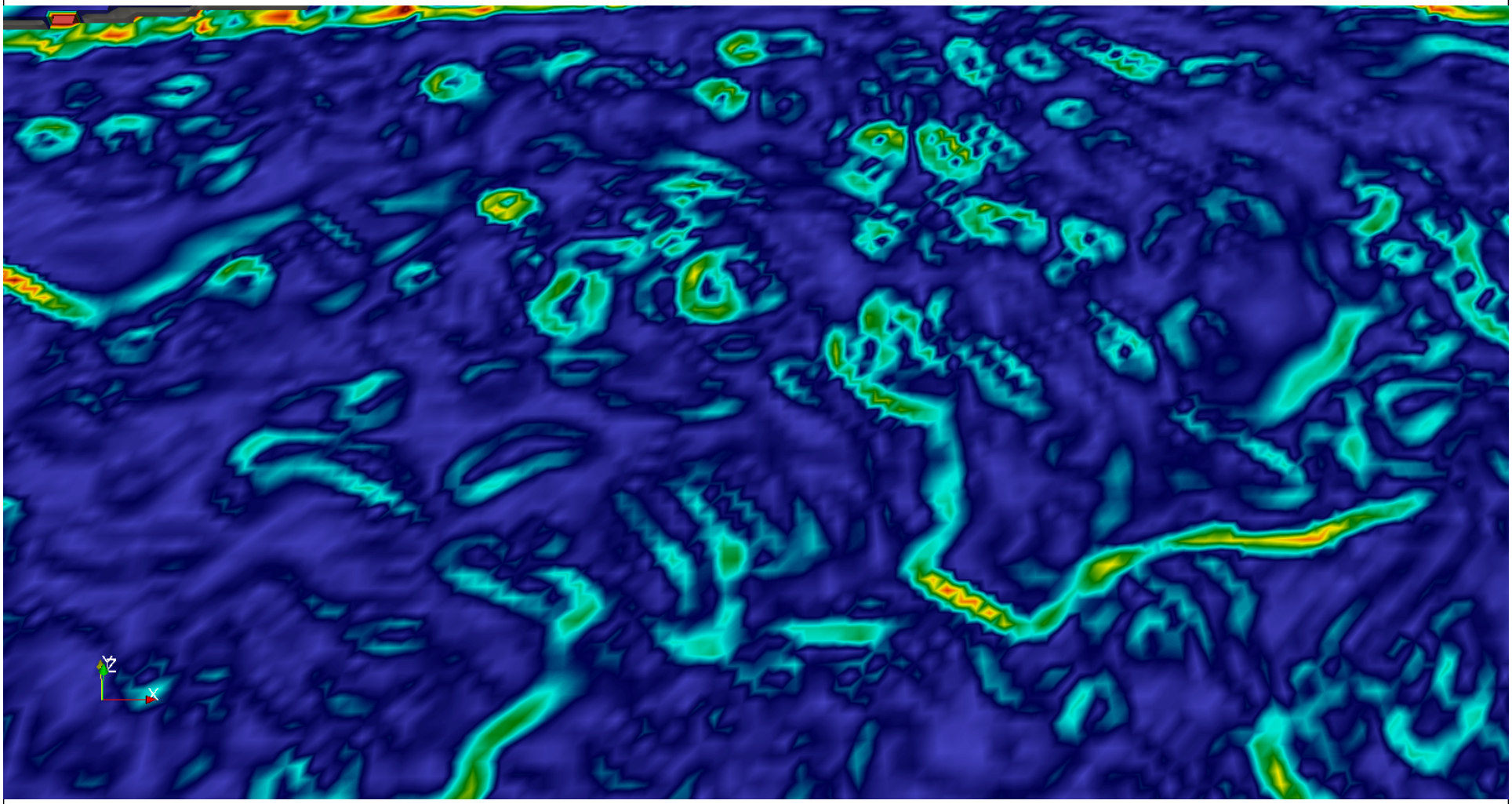
Case Study: NCOM

- 2D particle traces were finding clumps of particles at certain time steps
- Vertical current cells? 3D!

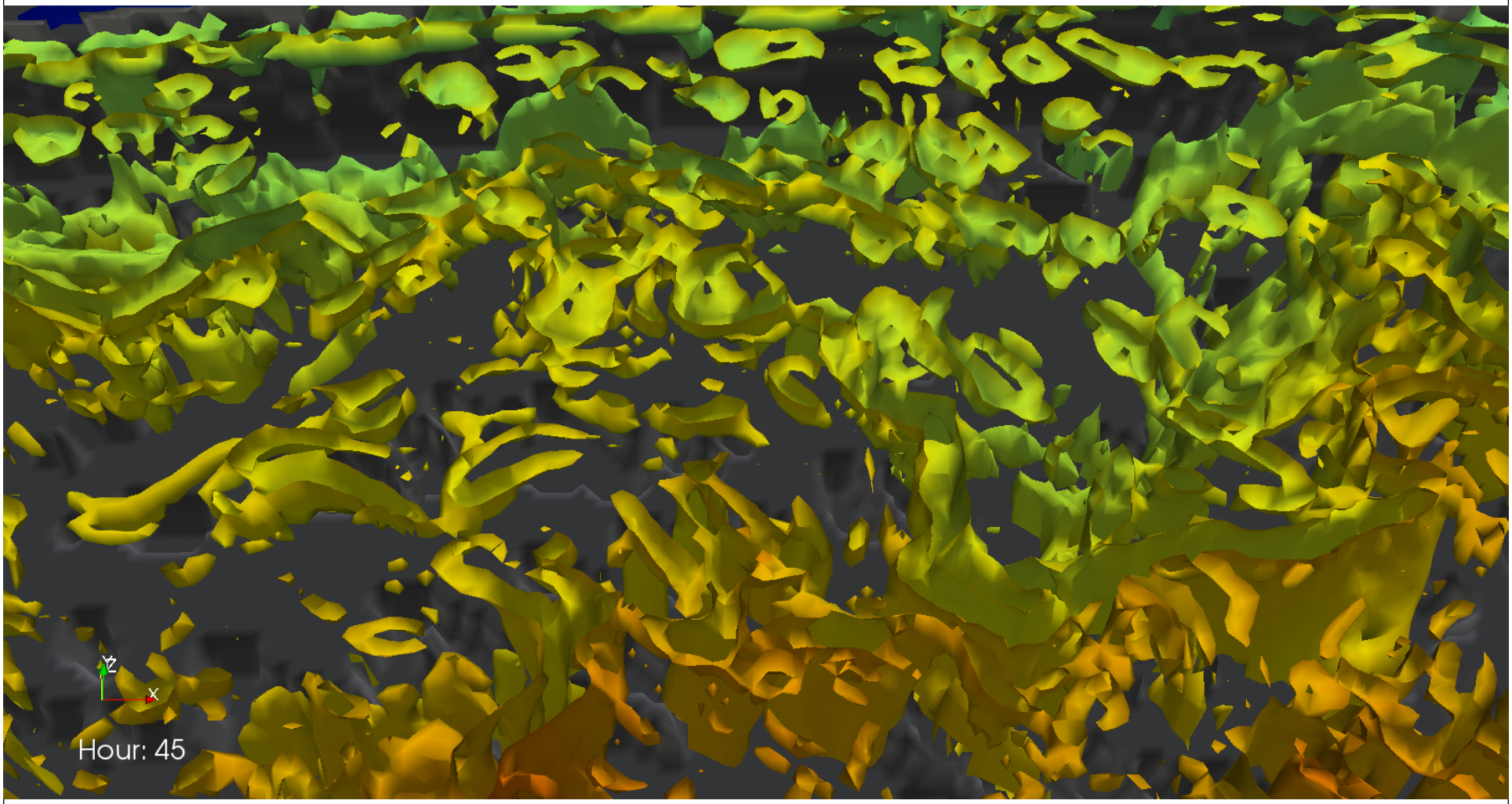

SCOPE Drifters over SST and Currents
2013121600 - 045
30.4°N
30.35°N
30.3°N
30.25°N
86.7°W
86.65°W
86.6°W
86.55°W
86.5°W
18.8
18.4
18.0
17.6
17.2
16.8
16.4
16.0

Case Study: NCOM (cont)

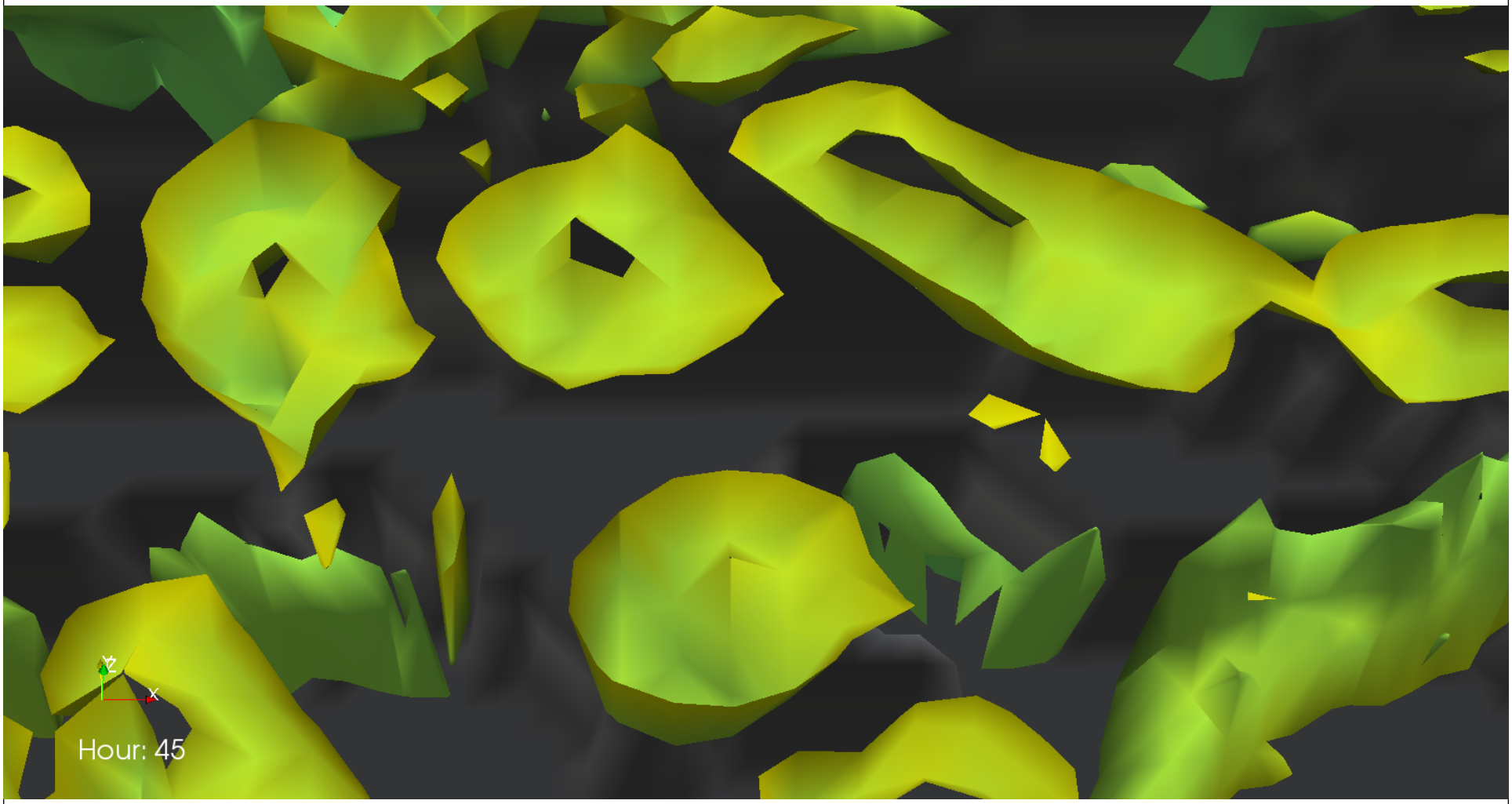
Features: temperature gradient important

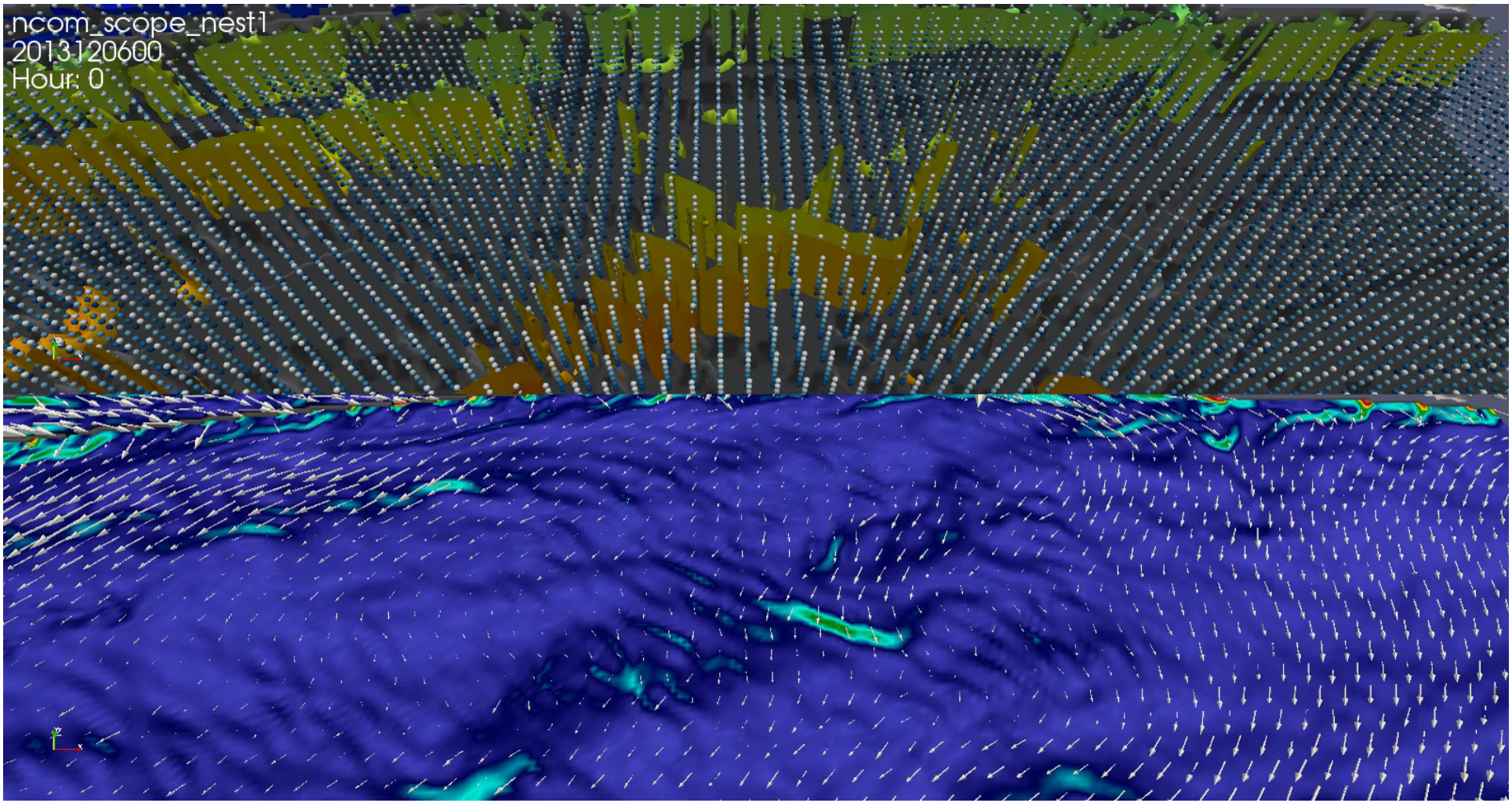


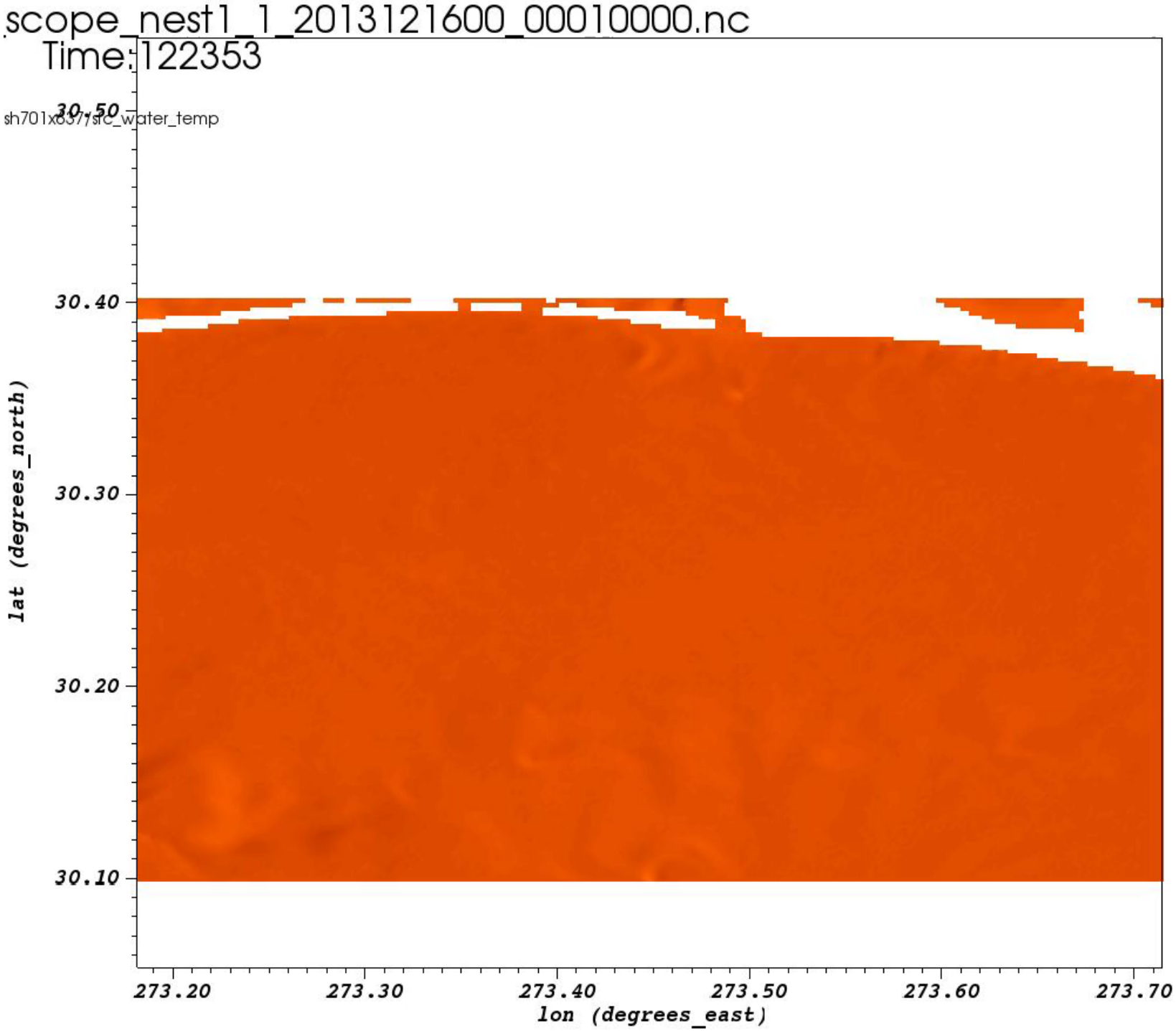
Case Study: NCOM (cont)

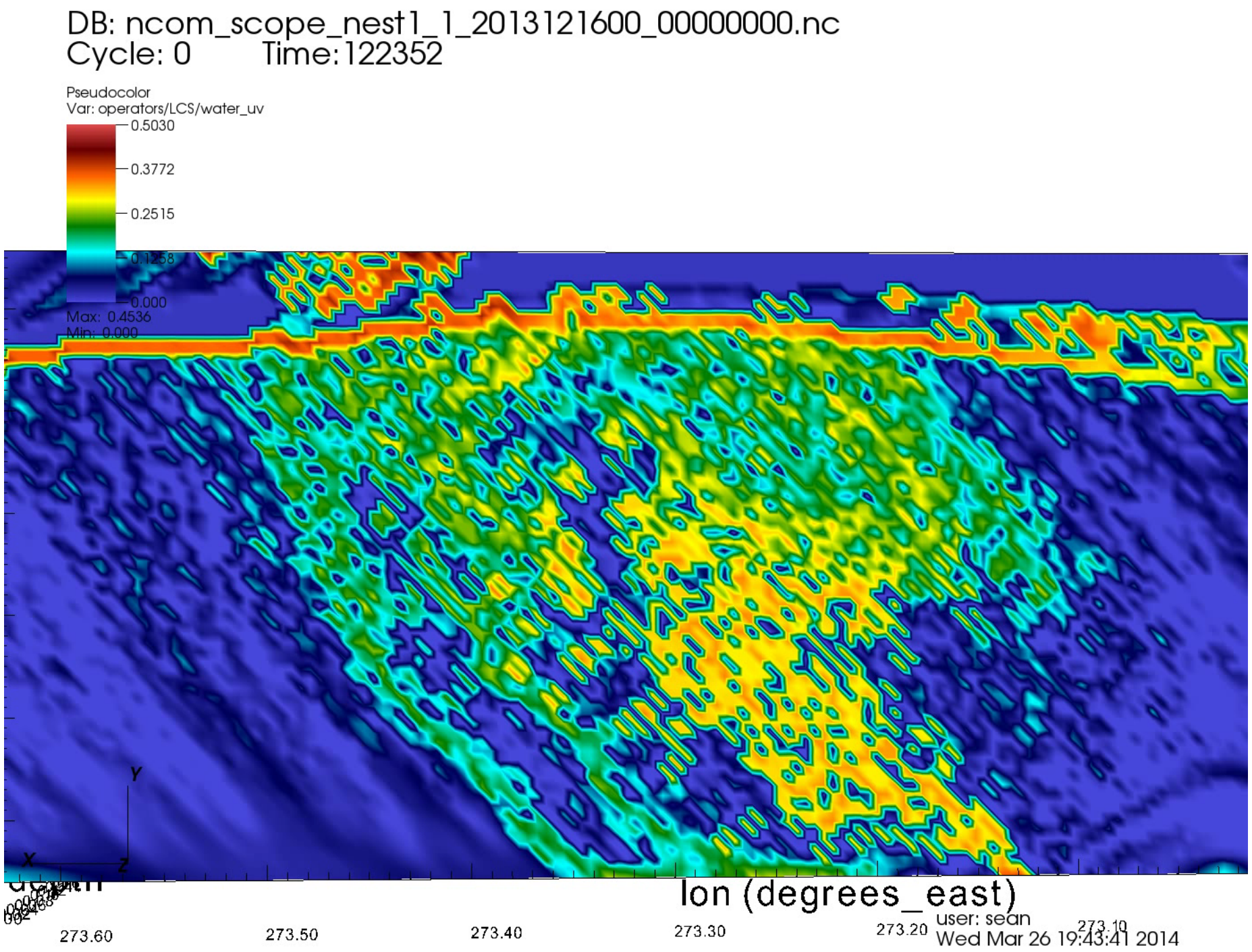


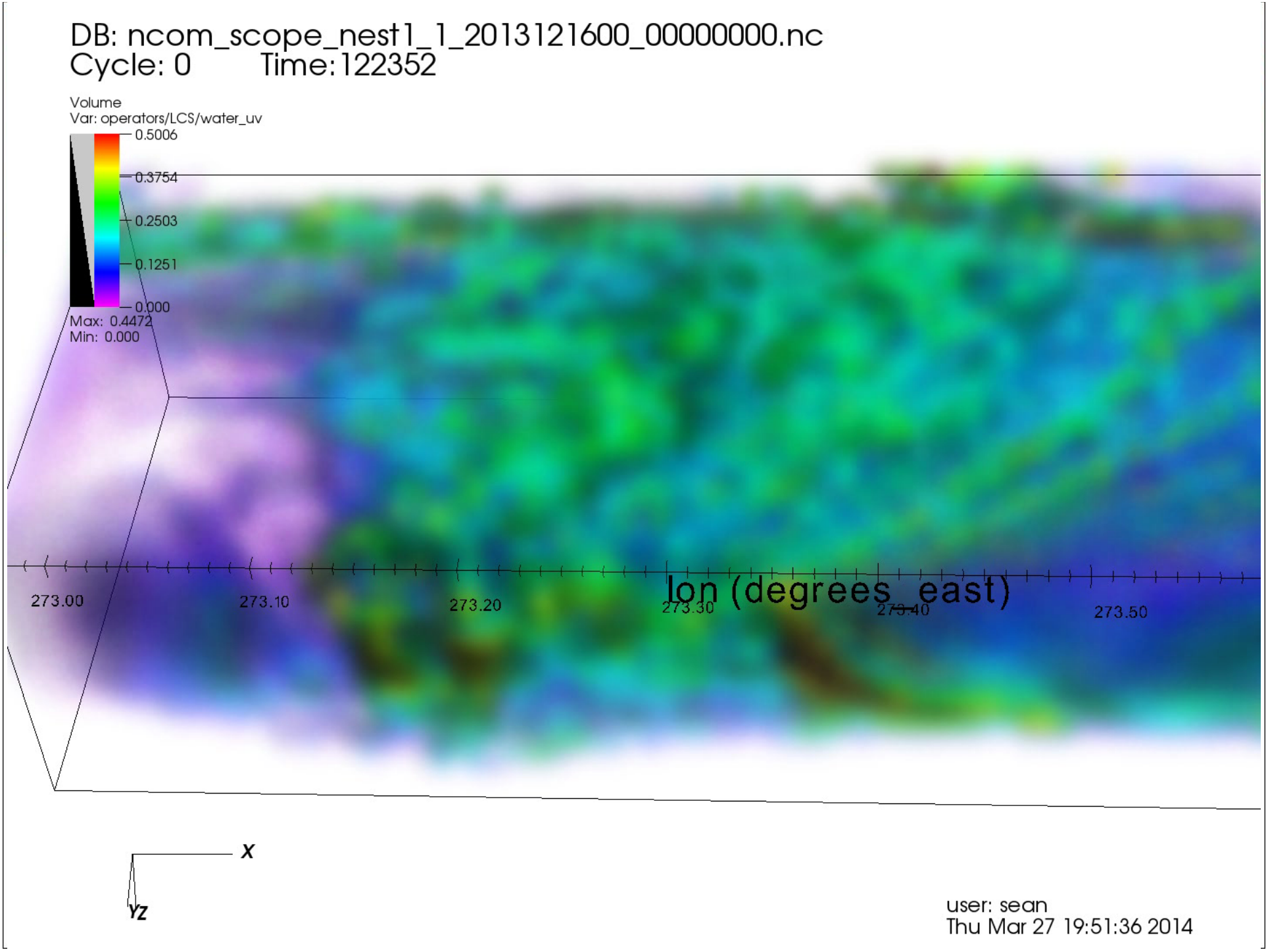
Case Study: NCOM (cont)




ncom_scope_nest1
2013120600
Hour: 0


scope_nest1_1_2013121600_00010000.nc
Time:122353
sh701x637/src_water_temp
30.50
30.40
30.30
30.20
30.10
lat (degrees_north)
273.20
273.30
273.40
273.50
273.60
273.70
lon (degrees_east)


DB: ncom_scope_nest1_1_2013121600_00000000.nc
Cycle: 0 Time:122352
Pseudocolor
Var: operators/LCS/water_uv
0.5030
0.3772
0.2515
0.1258
0.000
Max: 0.4536
Min: 0.000
Y
X
Z
lon (degrees_east)
273.60
273.50
273.40
273.30
273.20
273.10
user: sean
Wed Mar 26 19:43:41 2014


DB: ncom_scope_nest1_1_2013121600_00000000.nc
Cycle: 0 Time:122352
Volume
Var: operators/LCS/water_uv
0.5006
0.3754
0.2503
0.1251
0.000
Max: 0.4472
Min: 0.000
273.00
273.10
273.20
273.30
273.40
273.50
lon (degrees_east)
X
YZ
user: sean
Thu Mar 27 19:51:36 2014

What's Still Needed

- More rectilinear grid support
  - E.g., Gradient Filter in ParaView
- Some consideration of units (grid, time, etc.)
  - Grid: horizontal units (degrees), vertical (m)
    - VisIt: LCS regularly sampled seeds

What's Still Needed

- Separation of computational & display grids
  - Or a singular transform filter that handles many inputs
  - Grid & display coords fundamentally different, consider doing operations in grid coords then doing transform
    - VisIt handles this well when applying Operators to all Plots

Acknowledgements

- Jackie Kinney (Naval Postgraduate School)
- Jose Renteria (PETTT On-site @ NPS)
- Pat Gallacher (Naval Research Labs)
- Gregg Jacobs (Naval Research Labs)

Questions?

- Sean Ziegeler
  - sean.ziegeler.ctr@nrlssc.navy.mil
  - sean.ziegeler@engilitycorp.com