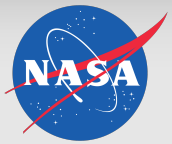
Building unstructured grids with JIGSAW

Darren Engwirda [1,2] {darren.engwirda@columbia.edu}

MPAS-Ocean team [3], COMPAS team [4]



[1] Center for Climate Systems Research, Columbia University, [2] NASA Goddard Institute for Space Studies, [2] Los Alamos National Laboratory, [4] Oceans and Atmospheres, CSIRO

International Workshop on Multi-scale (Un)-structured mesh Modelling for coastal, shelf and global ocean dynamics: Santa Fe, September 2019

What is in this session?

Workshop / tutorial-session building various global and coastal unstructured meshes using `JIGSAW`.

We'll work through set of example problems using your favourite scripting language: either `MATLAB` or `Python`.

Throughout, I'll provide brief background on the algorithms / methods running behind-the-scenes.

Please ask-questions / discuss throughout!

See `github.com/dengwirda/jigsaw-geo-tutorial` for tutorial files/data.

Getting Started (compiling):

The first step is to choose your scripting language and compile/configure the code. **You will need to have the following available**:

- a C++ compiler (`g++`, `clang++`, `msvc`, etc) + `cmake`.
- either: `MATLAB`, or `Python3` (`numpy`, `scipy`, `pathlib`) + `Paraview`.
- the `jigsaw-matlab` and/or `jigsaw-python` package, via `github`.

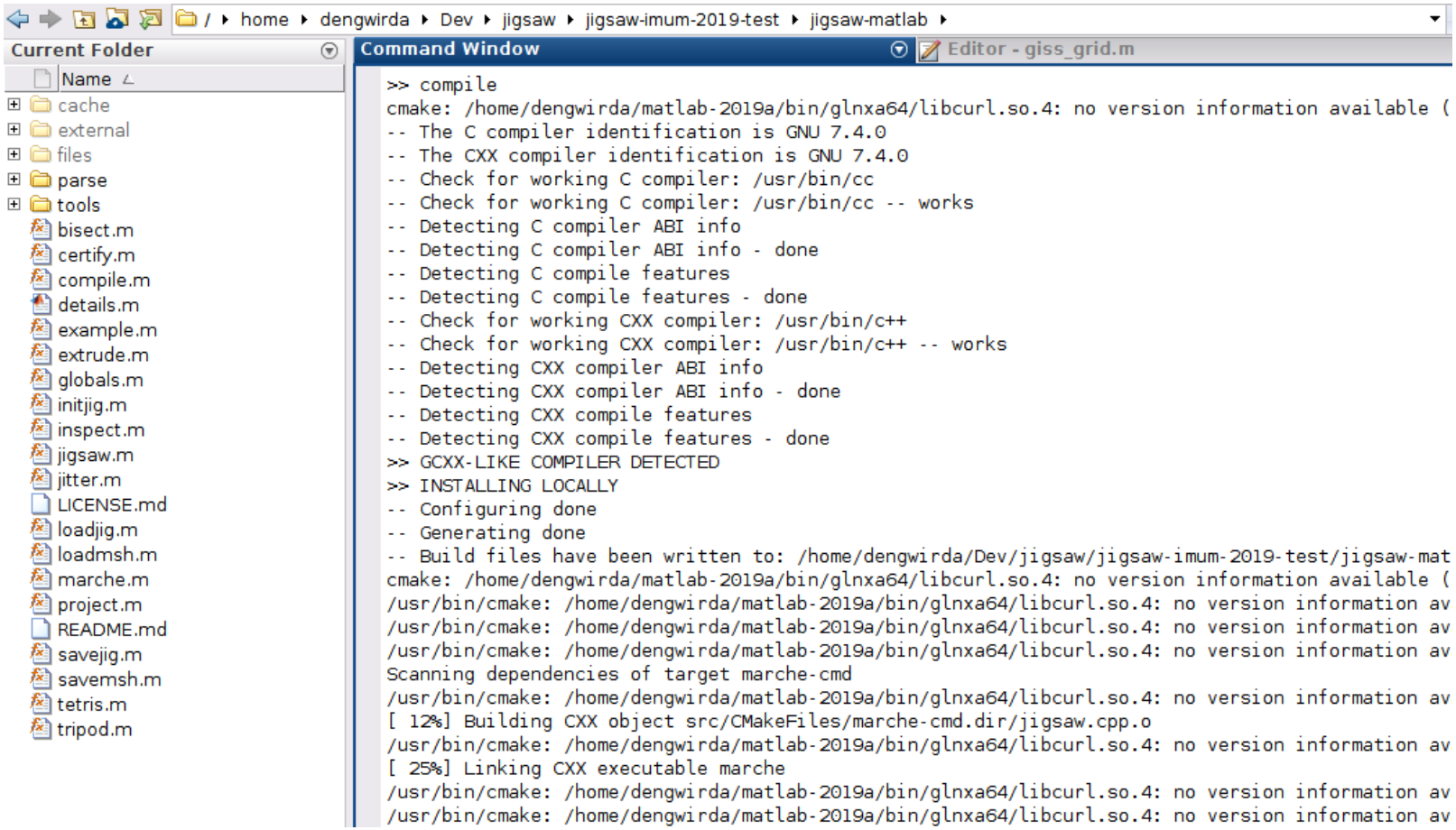
```
MATLAB (in cmd window):

cd jigsaw-matlab

compile;



(will run cmake script, etc)
```

```
Python (in system shell):

cd jigsaw-python

(sudo if needed...)
python3 setup.py build_external
python3 setup.py install

(will run cmake script, etc)
```

These steps will install the `jigsaw-matlab` and/or `jigsaw-python` environments on your system.

Getting Started (compiling):

In `MATLAB`, the installation should run via `cmake`, creating a set of executables in `../jigsaw-matlab/external/jigsaw/bin`:



Getting Started (compiling):

In `Python`, the installation should run via `cmake`, creating a set of executables in `../jigsaw-python/jigsawpy/bin_`:

```
dengwirda@dengwirda:~/Dev/jigsaw/jigsaw-imum-2019-test/jigsaw-python$ python3 se
tup.py build_external
running build_external
cmake config.
-- The C compiler identification is GNU 7.4.0
-- The CXX compiler identification is GNU 7.4.0
-- Check for working C compiler: /usr/bin/cc
-- Check for working C compiler: /usr/bin/cc -- works
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Detecting C compile features
-- Detecting C compile features - done
-- Check for working CXX compiler: /usr/bin/c++
-- Check for working CXX compiler: /usr/bin/c++ -- works
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Detecting CXX compile features
-- Detecting CXX compile features - done
>> GCXX-LIKE COMPILER DETECTED
>> INSTALLING LOCALLY
-- Configuring done
-- Generating done
-- Build files have been written to: /home/dengwirda/Dev/jigsaw/jigsaw-imum-2019
-test/jigsaw-python/external/jigsaw/tmp
cmake complie
Scanning dependencies of target tripod-cmd
[ 12%] Building CXX object src/CMakeFiles/tripod-cmd.dir/jigsaw.cpp.o
[ 25%] Linking CXX executable tripod
[ 25%] Built target tripod-cmd
Scanning dependencies of target jigsaw-lib
[ 37%] Building CXX object src/CMakeFiles/jigsaw-lib.dir/jigsaw.cpp.o
[ 50%] Linking CXX shared library libjigsaw.so
[ 50%] Built target jigsaw-lib
Scanning dependencies of target jigsaw-cmd
```

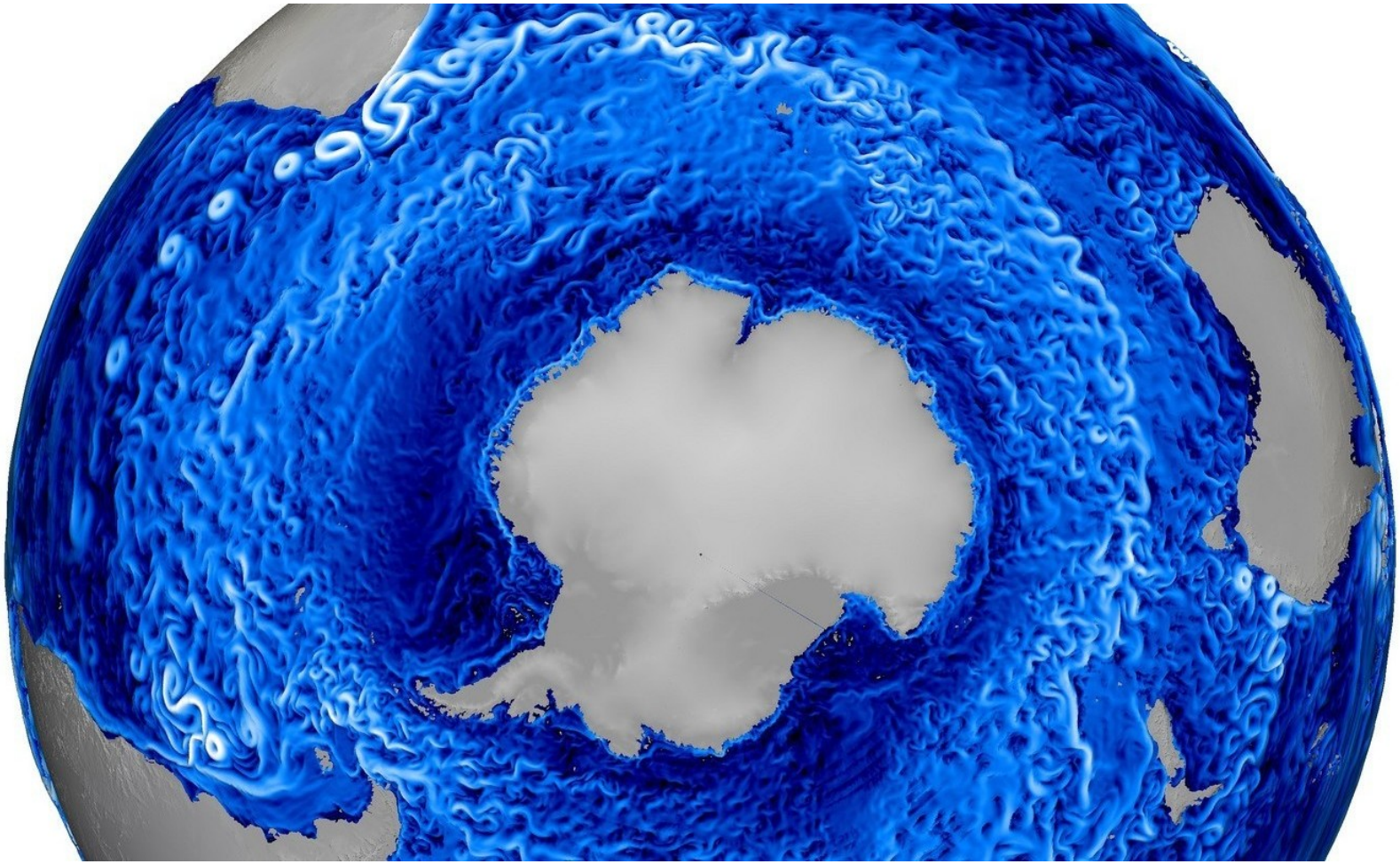
`jigsawpy` is then installed as a 'normal' Python package.

Background

Variable-resolution approaches for ocean modelling.

Motivation: building unstructured grids for geophysical flows

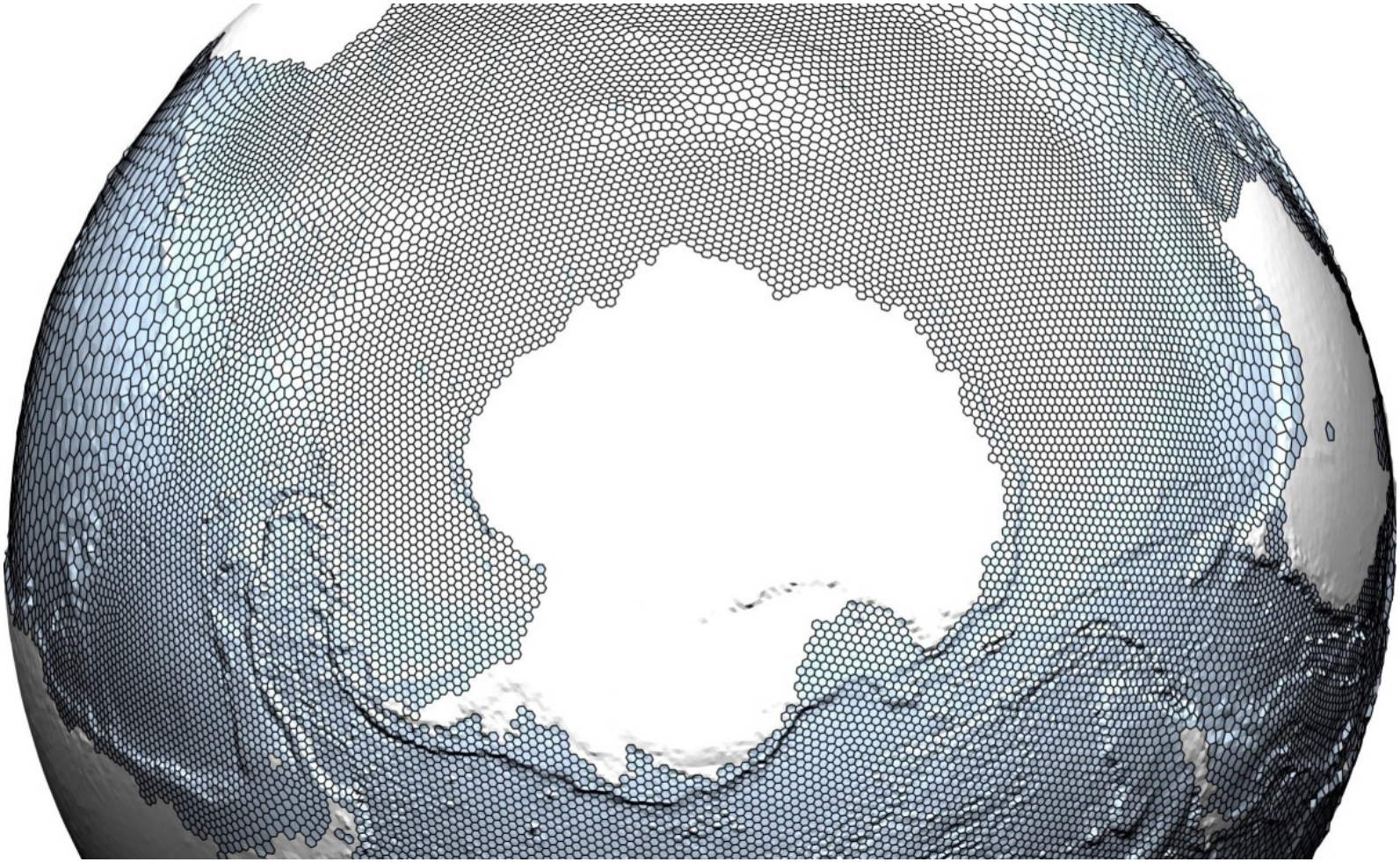
Develop flexible, variable resolution methods for geophysical flows based on unstructured meshes: **global ocean modelling**, **climate dynamics**, etc...



**Eddy-permitting global ocean, showing dynamics in the ACC (MPAS-O team, LANL).

Motivation: building unstructured grids for geophysical flows

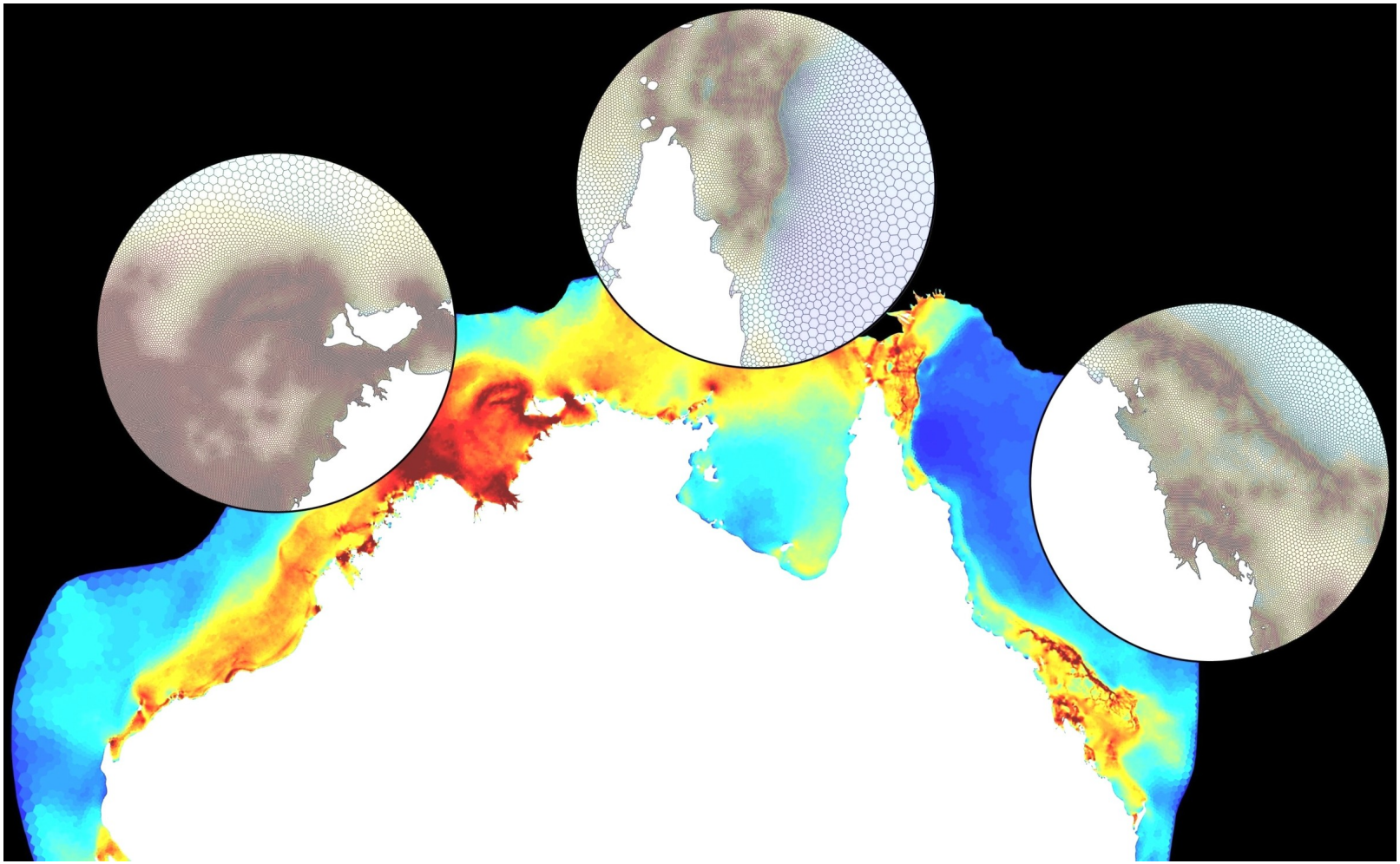
Develop flexible, variable resolution methods for geophysical flows based on unstructured meshes: **global ocean modelling**, **climate dynamics**, etc...



**Adapt mesh resolution to dynamics of mesoscale eddies / ocean-state variability?

Motivation: building unstructured grids for geophysical flows

Develop flexible, variable resolution methods for geophysical flows based on unstructured meshes: **coastal modelling**, **environmental assessment**, etc...



**Herzfeld et al: Estimation of tidal flow rate on the Australian shelf.

What is JIGSAW?

`JIGSAW`: a general-purpose meshing library designed to generate high-quality **Delaunay** / **Voronoi** grids for 2- and 3-dim. computational simulation.

Open source package, primarily written in `C++`, with common `MATLAB`, `Python` and `API` interfaces:

https://github.com/dengwirda/jigsaw

Thanks to the MPAS-Ocean (LANL) and COMPAS (CSIRO) teams for on-going collaboration / development / feedback / etc!

There are also many other excellent meshing packages available: Gmsh, Distmesh(Oceanmesh2D, ADmesh), etc...

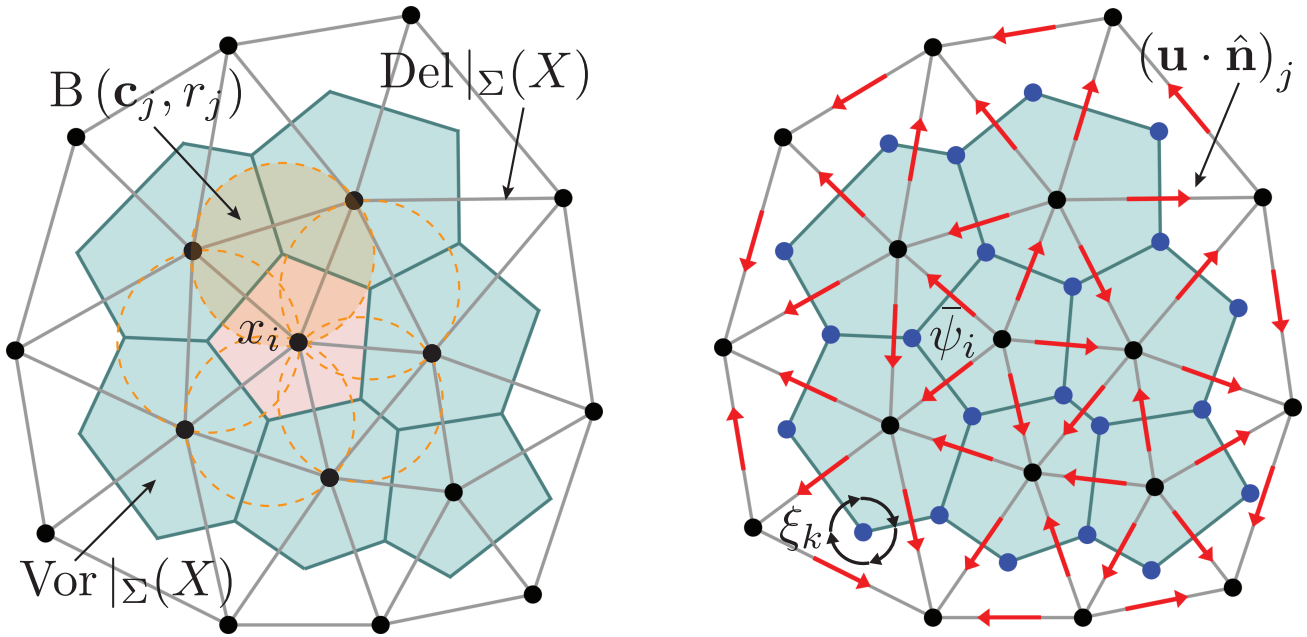
Engwirda: Generalised primal-dual grids for unstructured co-volume schemes, JCP, 2018.

Engwirda: JIGSAW-GEO (1.0): locally orthogonal staggered unstructured grid generation for general circulation modelling on the sphere, GMD, 2017.

Formulation: staggered unstructured finite-volumes

Unstructured generalisation of Arwkawa-type schemes:

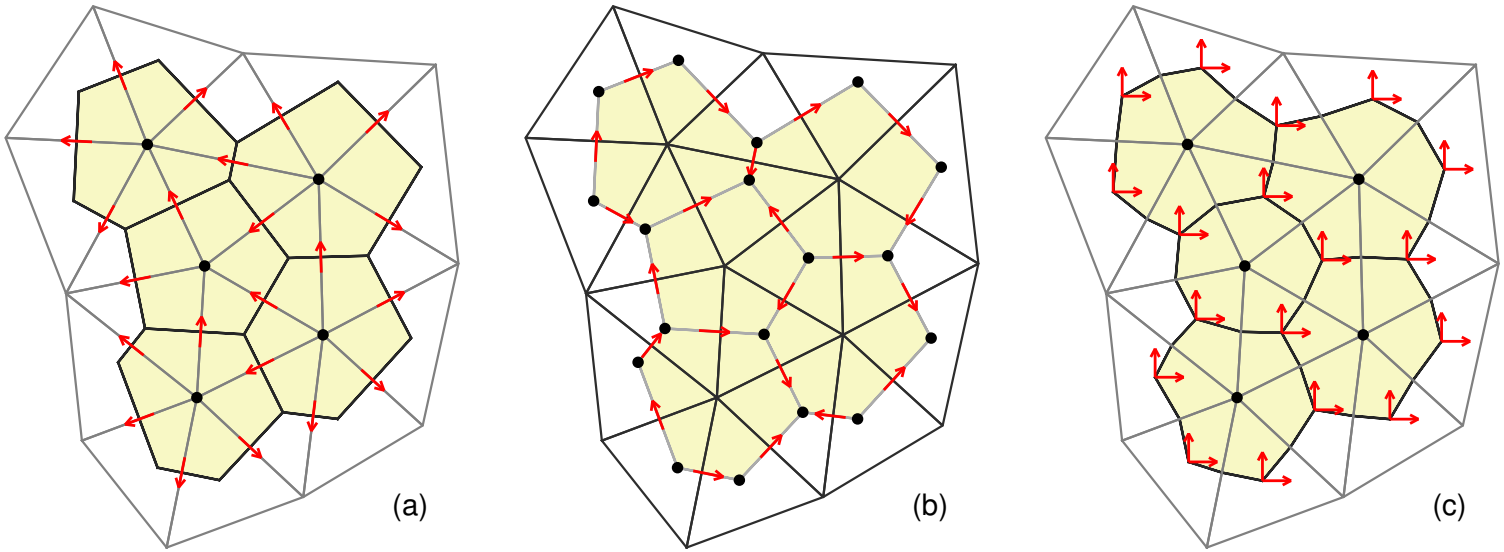
- Staggered finite-difference/finite-volume mimetic schemes.
- Vector components staggered on primal cell edges.
- Conserved quantities centred in dual control volumes.



**Staggered, mimetic finite-volume scheme ('TRiSK') used in the Model for Prediction Across Scales (MPAS-O), Ringler et al., 2013, and the Coastal Marine Prediction Across Scales (COMPAS), Herzfeld et al., 2019.

Formulation: staggered unstructured finite-volumes

'Hex' C-grid (TRiSK) **'Tri' C-grid (ICON)** **B-grid (FESOM2)**



**Danilov et al.: The Finite-volumE Sea-ice Ocean Model (FESOM2), 2017; Korn et al.: Formulation of an unstructured grid model for global ocean dynamics (ICON), 2017.

Discretisation error is a strong function of grid-cell regularity — need cells with 'nice' distribution of angles, edge-lengths, cell-areas, triangle/dual staggering, etc $\rightarrow$ **multiple coupled constraints**!

Formulation: staggered unstructured finite-volumes

Requirements on meshing strategies:

- Produce unstructured triangulations / dual-meshes for spherical domains, regional/coastal domains with complex boundaries.
- Follow non-uniform 'mesh-spacing' (i.e. resolution) functions: $\bar{h}(\mathbf{x})$.
- Produce orthogonal meshes (i.e. Delaunay/Voronoi grids) for C-grid models.
- Produce 'Centroidal Voronoi' meshes (i.e. 'nice' primal-dual meshes).
- Heavy optimisation of mesh quality: max. $\theta_\tau \leq 90^\circ$, no short edges in dual mesh, 'equilibrate' staggering between triangles and polygons, smooth variation in cell-area, etc.
- Produce 'large' meshes (i.e. $\geq$ 1M cells) with reasonable efficiency.

Example 1:

Building uniform resolution global grids.

(Any questions / issues wrt. installation?)

Example 1:

The first example is a simple global grid with uniform 150km resolution:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_1;

(will run example-1)
(and display in MATLAB)
```

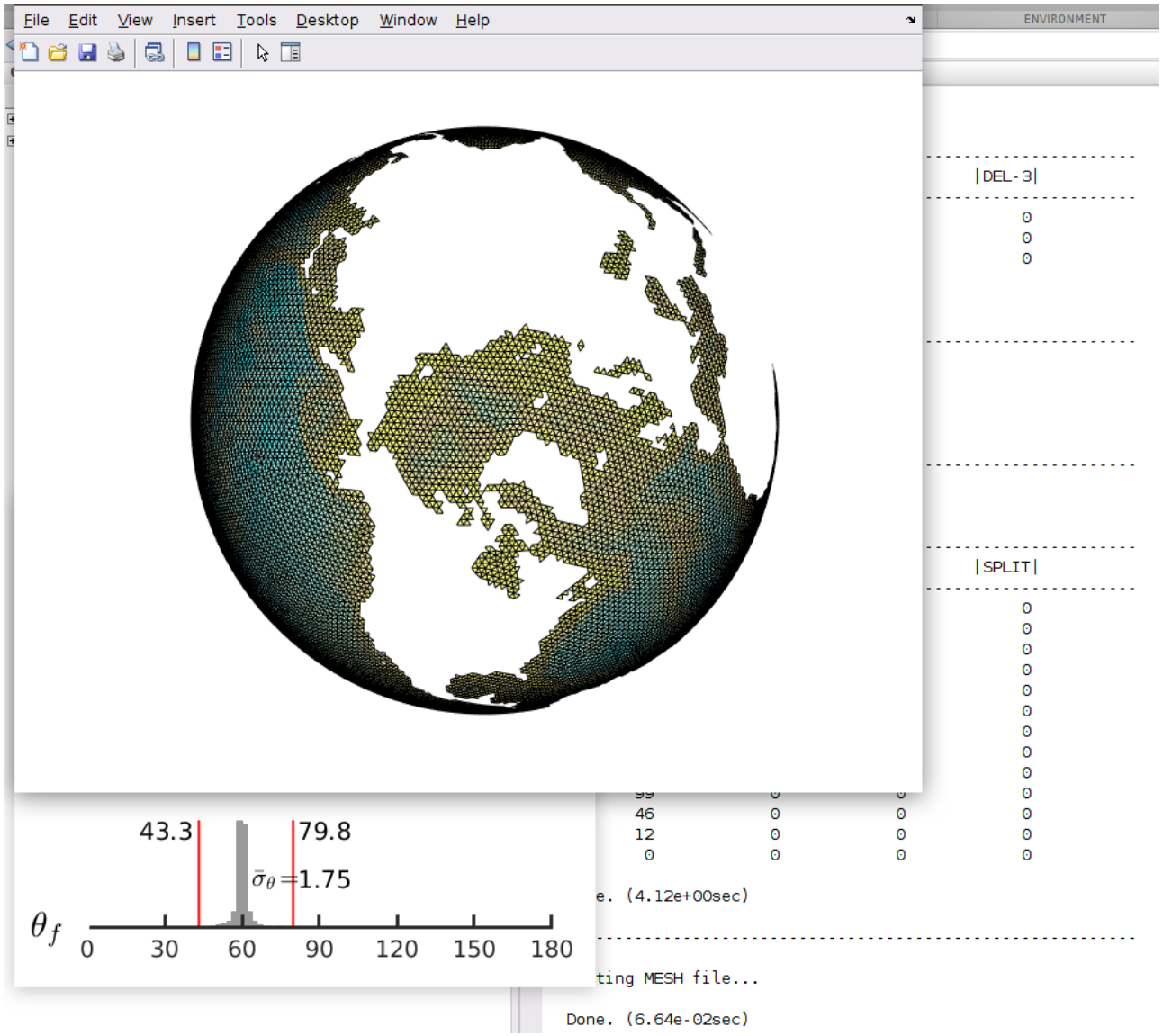
```
Python (in system shell):

cd ..
cd examples-python

python3 ex_1.py

(will run example-1)
(and save ex_1.vtk file)

open ex_1.vtk in Paraview
```

All examples in this session will follow a similar workflow: run a MATLAB / Python script and then visualise / post-process.

Hopefully, these scripts can form templates for subsequent meshing tasks — please feel free to 'hack-around' with the content throughout!

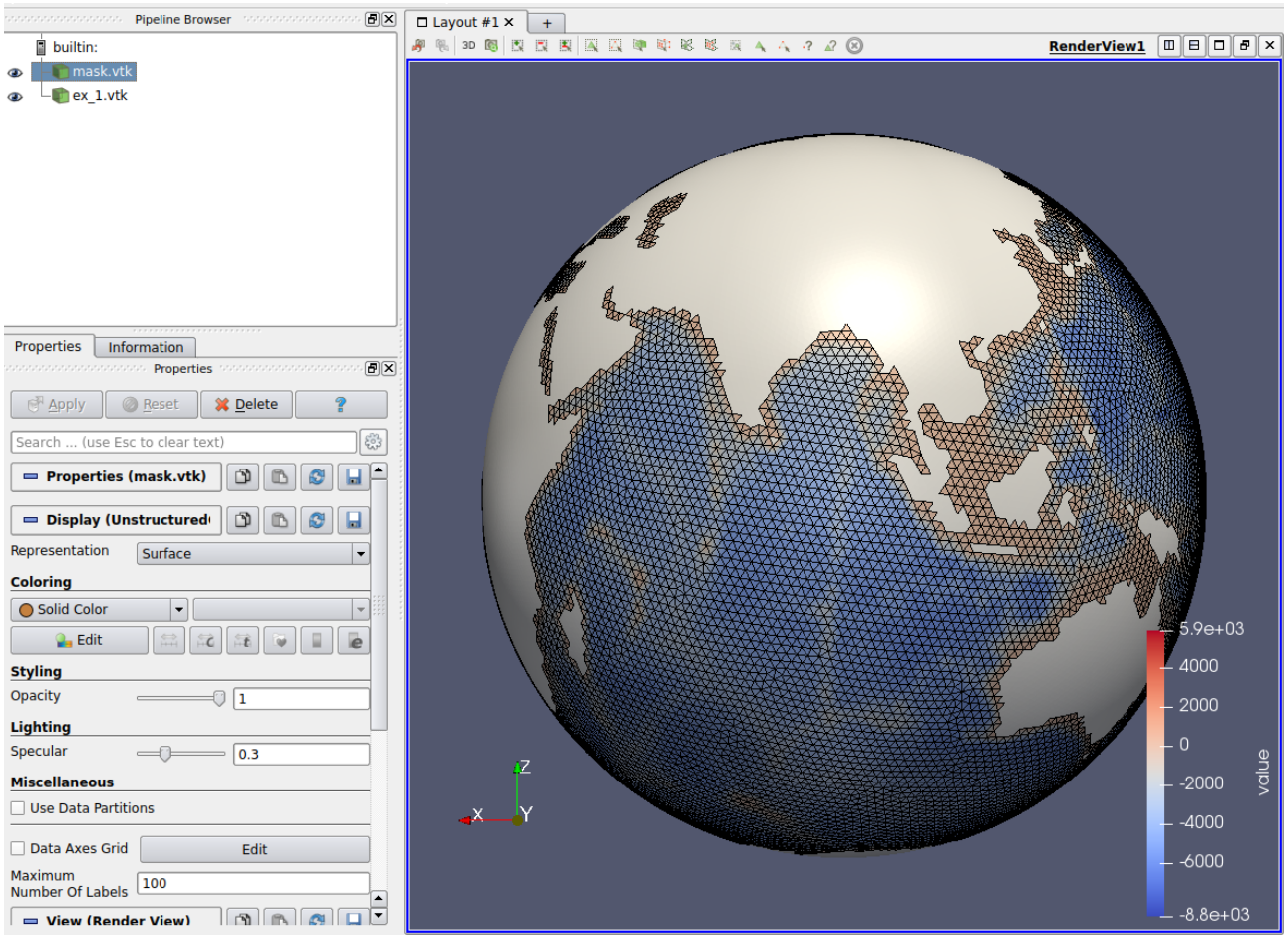
Example 1:

In MATLAB, a global grid + mesh statistics should be shown:



Example 1:

In `Python`, the script should save a `*.vtk` file for `Paraview`:



Also load `../examples-python/files/mask.vtk` as a land-mask.

Example 1: workflow

JIGSAW passes information between its MATLAB / Python layers and the C++ back-end via structured data-types:

JIGSAW works with two basic data-types:

- `jig_t // *.jig-file`: set of user-defined options.
- `msh_t // *.msh-file`: mesh/geom data-structure.

We will use short MATLAB and/or Python scripts to manage the meshing process — defining various `jig_t` and `msh_t` objects to control the meshing workflow.

Such scripts offer a good way to manage the 'history' of various meshing configurations — fully specifying all inputs, options, etc.

Example 1: workflow

Working through `ex_1.py` in detail:

```
11 # DEMO-1: generate a uniform resolution (150KM) global grid.
12
13     src_path = os.path.join(
14         os.path.abspath(
15         os.path.dirname(__file__)),"files")
16
17     dst_path = os.path.join(
18         os.path.abspath(
19         os.path.dirname(__file__)),"cache")
20
21
22     opts = jigsawpy.jigsaw_jig_t()
23
24     topo = jigsawpy.jigsaw_msh_t()
25
26     geom = jigsawpy.jigsaw_msh_t()
27     mesh = jigsawpy.jigsaw_msh_t()
28
29 #---------------------------------- setup files for JIGSAW
30
31     opts.geom_file = \
32         str(Path(dst_path)/"earth.msh") # GEOM file
33
34     opts.jcfg_file = \
35         str(Path(dst_path)/"globe.jig") # JCFG file
36
37     opts.mesh_file = \
38         str(Path(dst_path)/"globe.msh") # MESH file
39
```

Setup `jig_t` and `msh_t` objects, and define filenames.

Example 1: workflow

Working through `ex_1.py` in detail:

```
40 #---------------------------------- define JIGSAW geometry
41
42     geom.mshID = "ellipsoid-mesh"
43     geom.radii = np.full(3, 6371.,
44         dtype=jigsawpy.jigsaw_msh_t.REALS_t)
45
46     jigsawpy.savemsh(opts.geom_file, geom)
47
```

Define spherical geometry with $R = 6371$km.

```
48 #---------------------------------- make mesh using JIGSAW
49
50     opts.hfun_scal = "absolute"
51     opts.hfun_hmax = +150.          # uniform at 150km
52
53     opts.mesh_dims = +2             # 2-dim. simplexes
54
55     opts.optm_qlim = +9.5E-01       # tighter opt. tol
56     opts.optm_iter = +32
57     opts.optm_qtol = +1.0E-05
58
59     jigsawpy.cmd.jigsaw(opts, mesh)
60
```

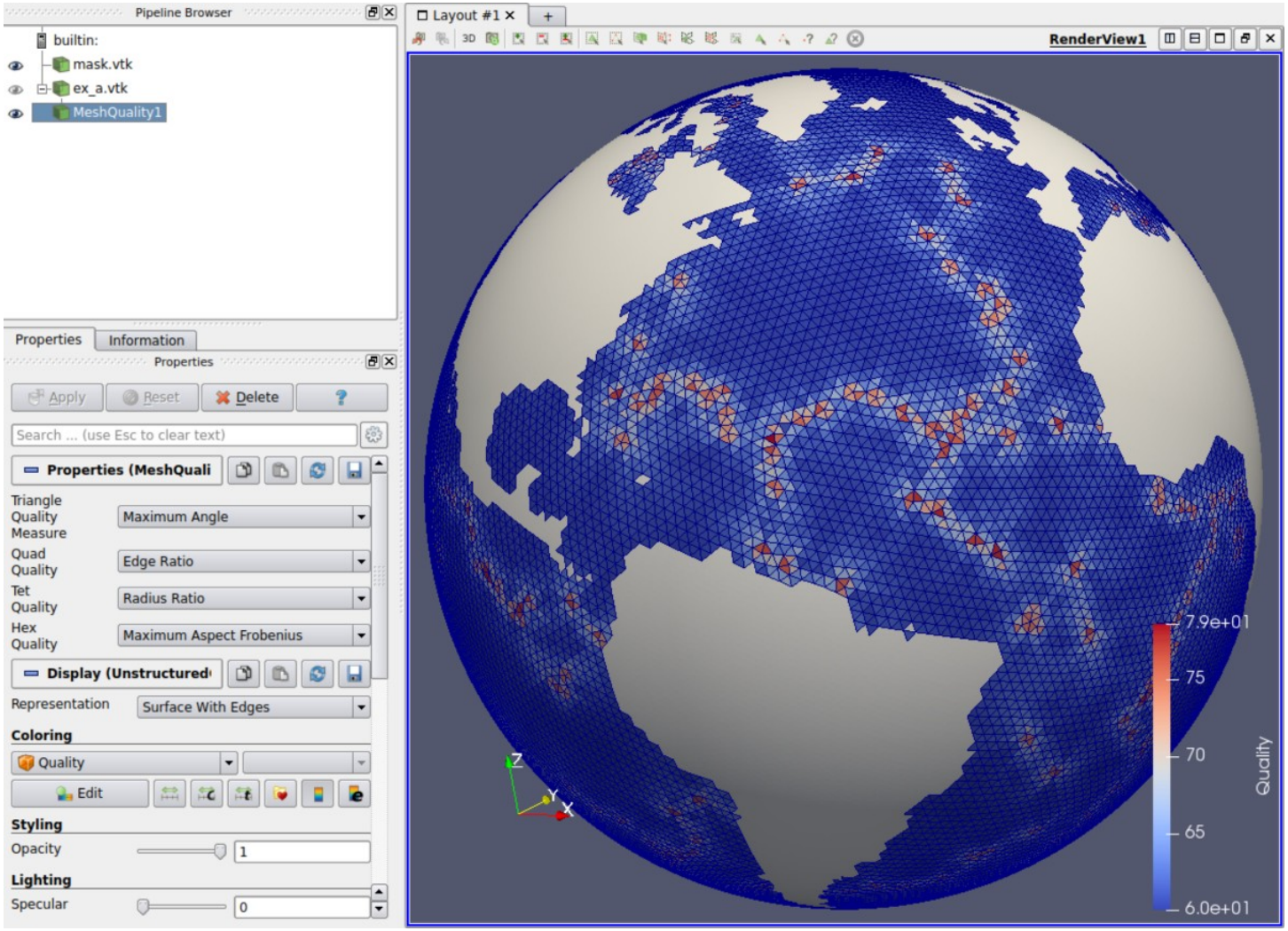
Define `JIGSAW` options and call mesh generator (`msh_t.py` or `help jigsaw`).

Example 1: workflow

```
61 #----------------------------------- save mesh for Paraview
62
63     jigsawpy.loadmsh(
64         str(Path(src_path)/"topo.msh"), topo)
65
66 #----------------------------------- a very rough land mask
67
68     apos = jigsawpy.R3toS2(
69         geom.radii, mesh.point["coord"][:])
70
71     apos = apos * 180./np.pi
72
73     zfun = interpolate.RectBivariateSpline(
74         topo.ygrid, topo.xgrid, topo.value)
75
76     mesh.value = zfun(
77         apos[:, 1], apos[:, 0], grid=False)
78
79     zmsk = \
80     mesh.value[mesh.tria3["index"][:,0]] + \
81     mesh.value[mesh.tria3["index"][:,1]] + \
82     mesh.value[mesh.tria3["index"][:,2]]
83     zmsk = zmsk / +3.0
84
85     mesh.tria3 = mesh.tria3[zmsk < +0.]
86
87
88     print("Writing ex_a.vtk file.")
89
90     jigsawpy.savevtk(
91         str(Path(dst_path)/"ex_a.vtk"), mesh)
92
```

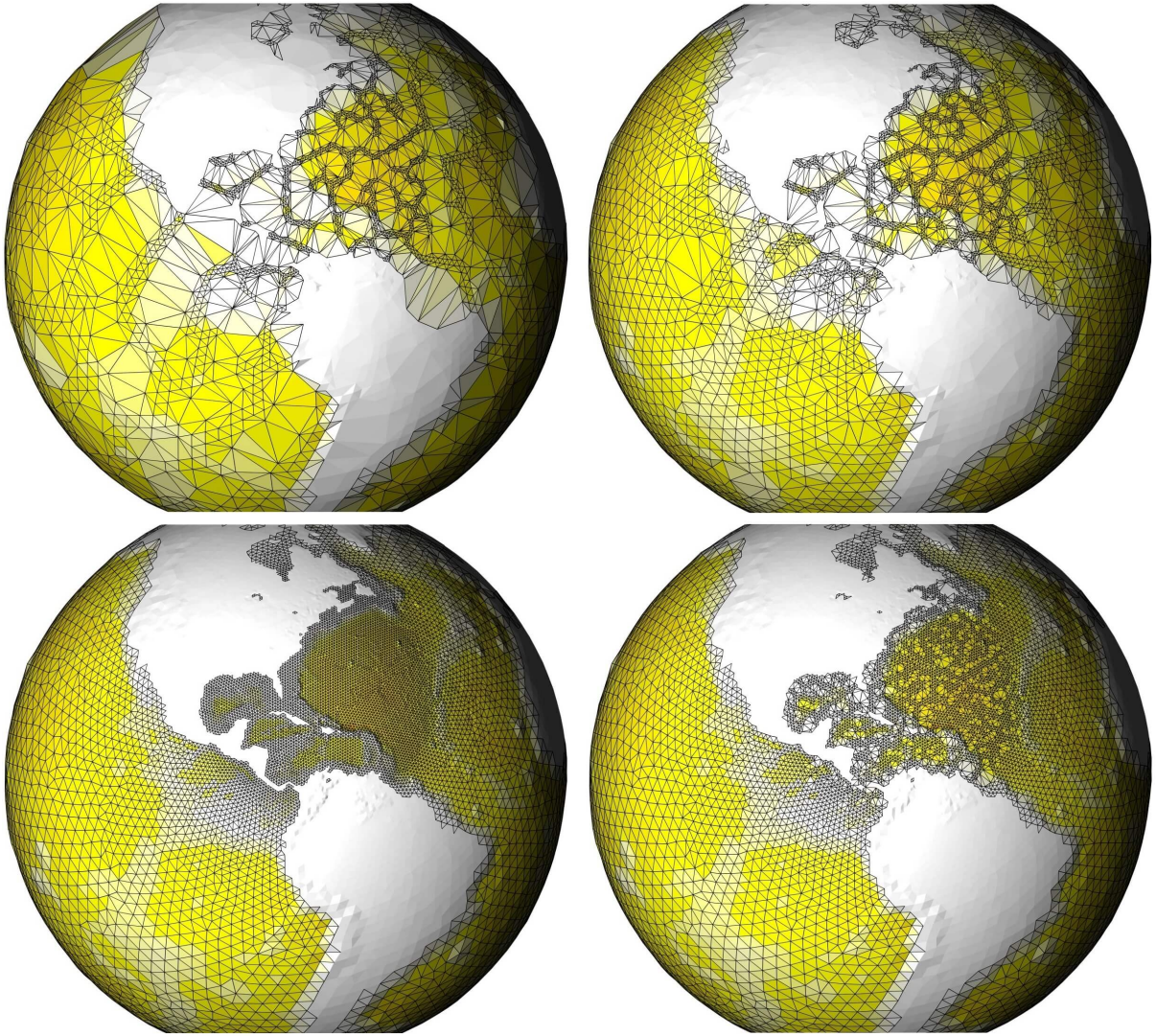
Write output for `Paraview` with a (very crude!) land mask.

Example 1: output



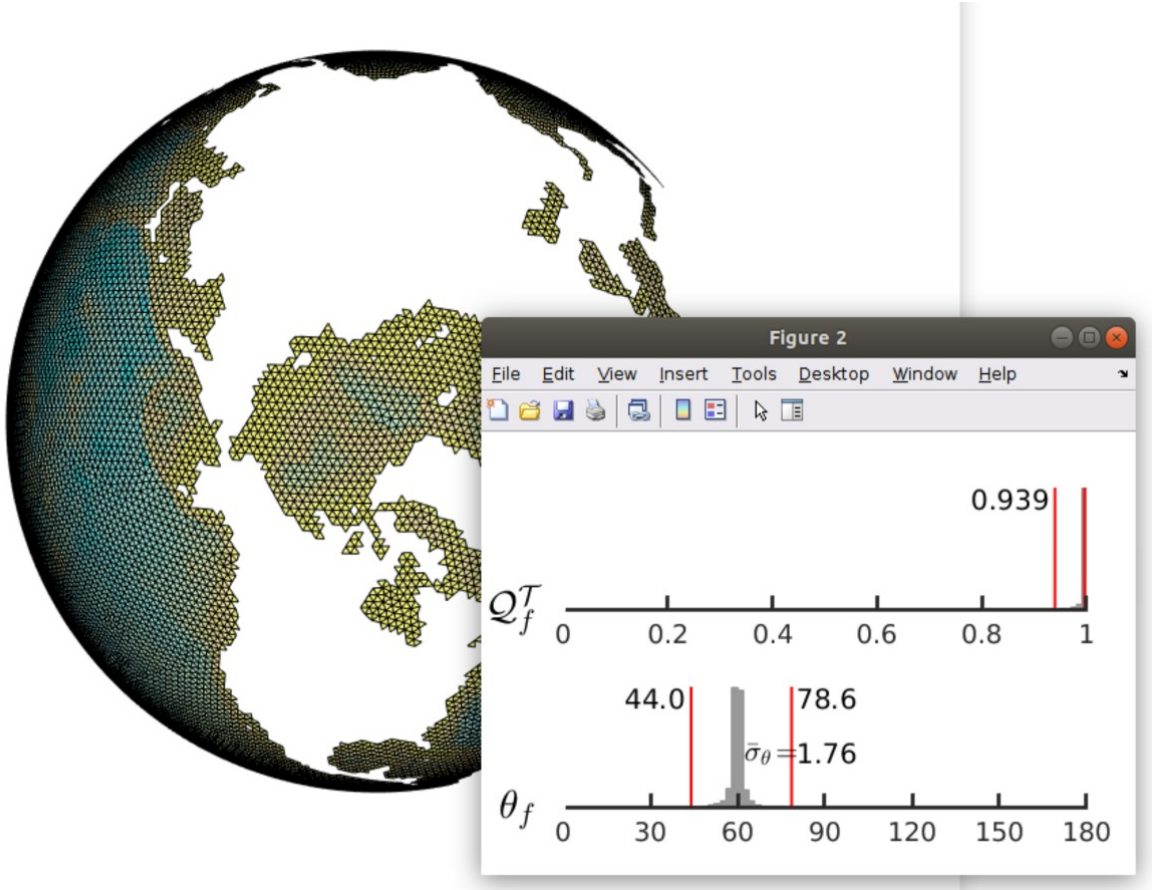
Very uniform mesh, with max. $\theta_\tau \leq 79^\circ$ (filters/alphabetical/mesh-quality).

Example 1: algorithms



JIGSAW produces meshes in two steps: (1) ‘refining‘ the mesh by inserting individual nodes (an ‘off-centre‘ Delaunay refinement technique).

Example 1: algorithms



`JIGSAW` produces meshes in two steps: (2) 'improving' the mesh via optimisation (a 'penalty' Optimal-Delaunay / Centroidal-Voronoi approach).

Typically generates very high-quality meshes, with $40^\circ \leq \theta_\tau \leq 80^\circ$.

Example 2:

Building variable resolution global grids.

Example 2:

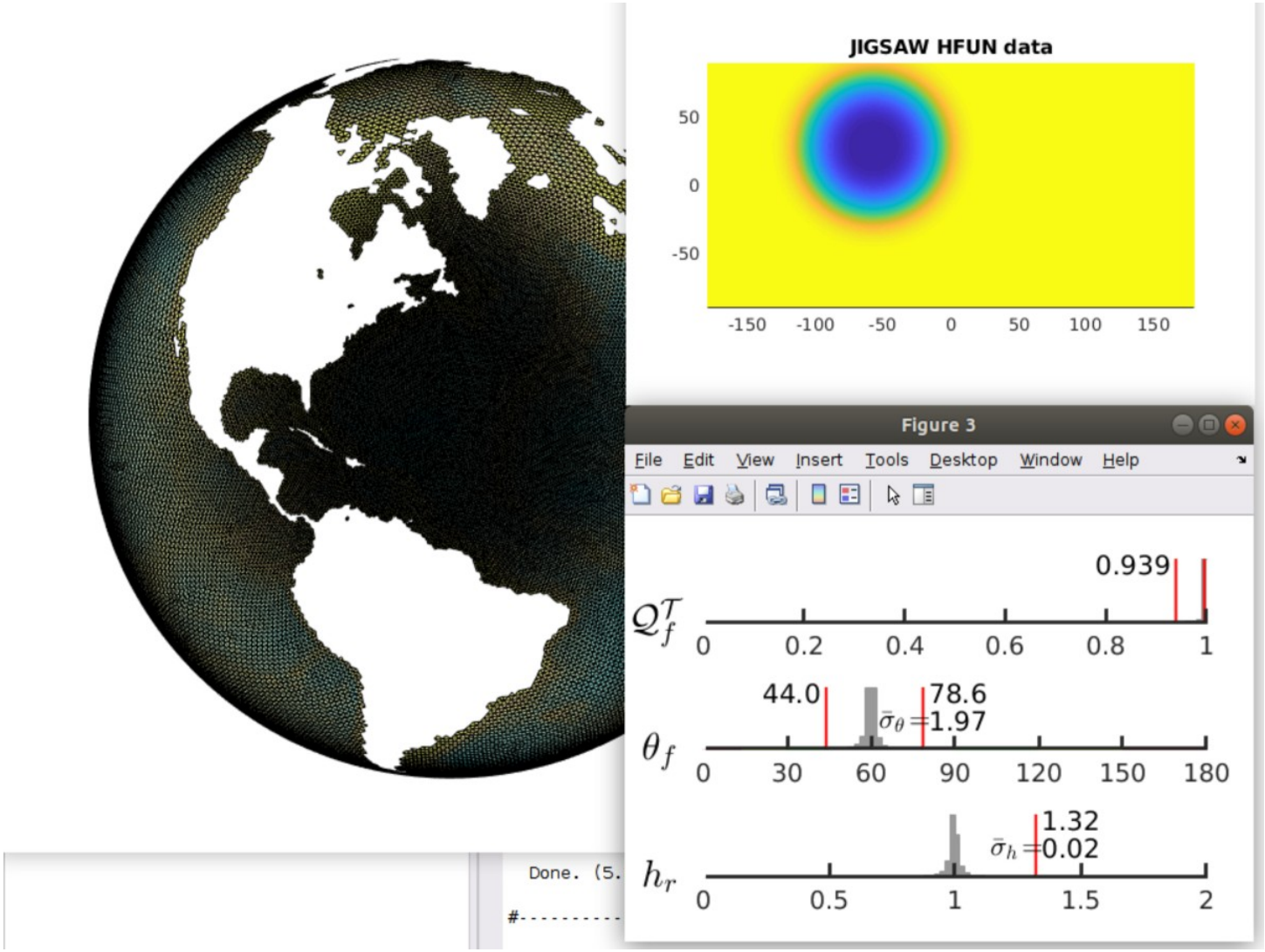
The second example is a global grid with a simple resolution pattern:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_2;

(will run example-2)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_2.py

(will run example-2)
(and save ex_2.vtk file)

open ex_2.vtk in Paraview
```

Defining smooth analytical resolution dependence can be a good way to achieve simple regional refinement. Mesh spacing is specified via the 'HFUN' argument: $\bar{h}(\mathbf{x})$.

Example 2: output



Regional refinement in the Atlantic / Gulf Stream region.

Example 2: workflow

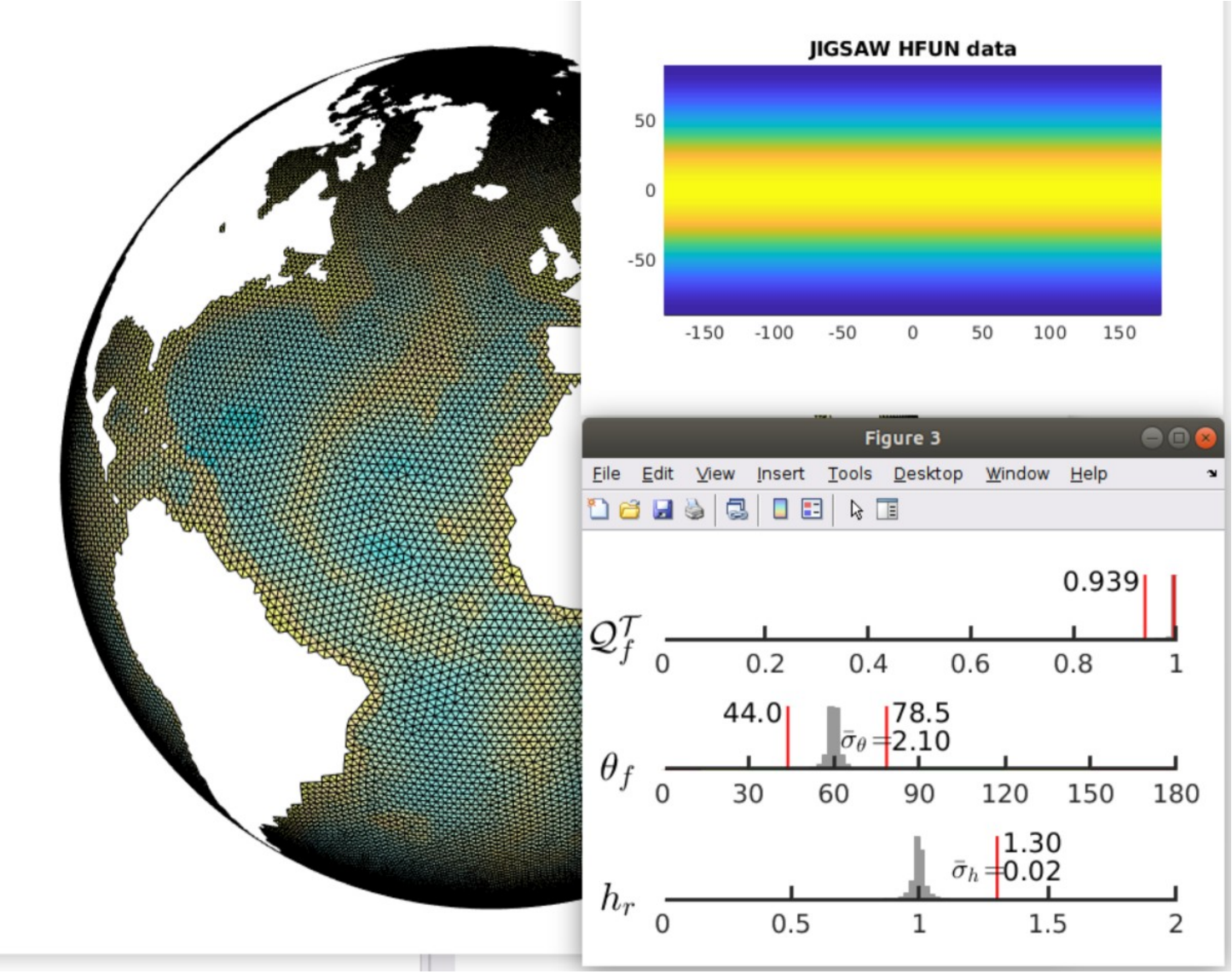
Working through `ex_2.m` in detail:

```
%---------------------------------- define spacing pattern

    hfun.mshID = 'ELLIPSOID-GRID' ;
    hfun.radii = geom.radii;

    hfun.point.coord{1} = linspace( ...
        -1.*pi, +1.*pi, 720) ;

    hfun.point.coord{2} = linspace( ...
        -.5*pi, +.5*pi, 360) ;

   [xmat, ymat] = meshgrid ( ...
        hfun.point.coord{1}, ...
        hfun.point.coord{2}) ;

    hfun.value = +150. - 100. * exp( ...
    -1.5*((xmat+1.).^2+(ymat-.5).^2).^2 ...
              ) ;

    savemsh (opts.hfun_file,hfun) ;

```

Define resolution distribution as an analytical function on a lon-lat grid.

Try different options: $\bar{h}(\mathbf{x}) = 150 - 100\sin^2(\phi)$, with ϕ latitude.

Example 2: output



Regional refinement in the polar regions: $\bar{h}(\mathbf{x})$ can be arbitrarily complex!

Example 3:

Building ‘aggressively‘ non-uniform global grids.

Example 3:

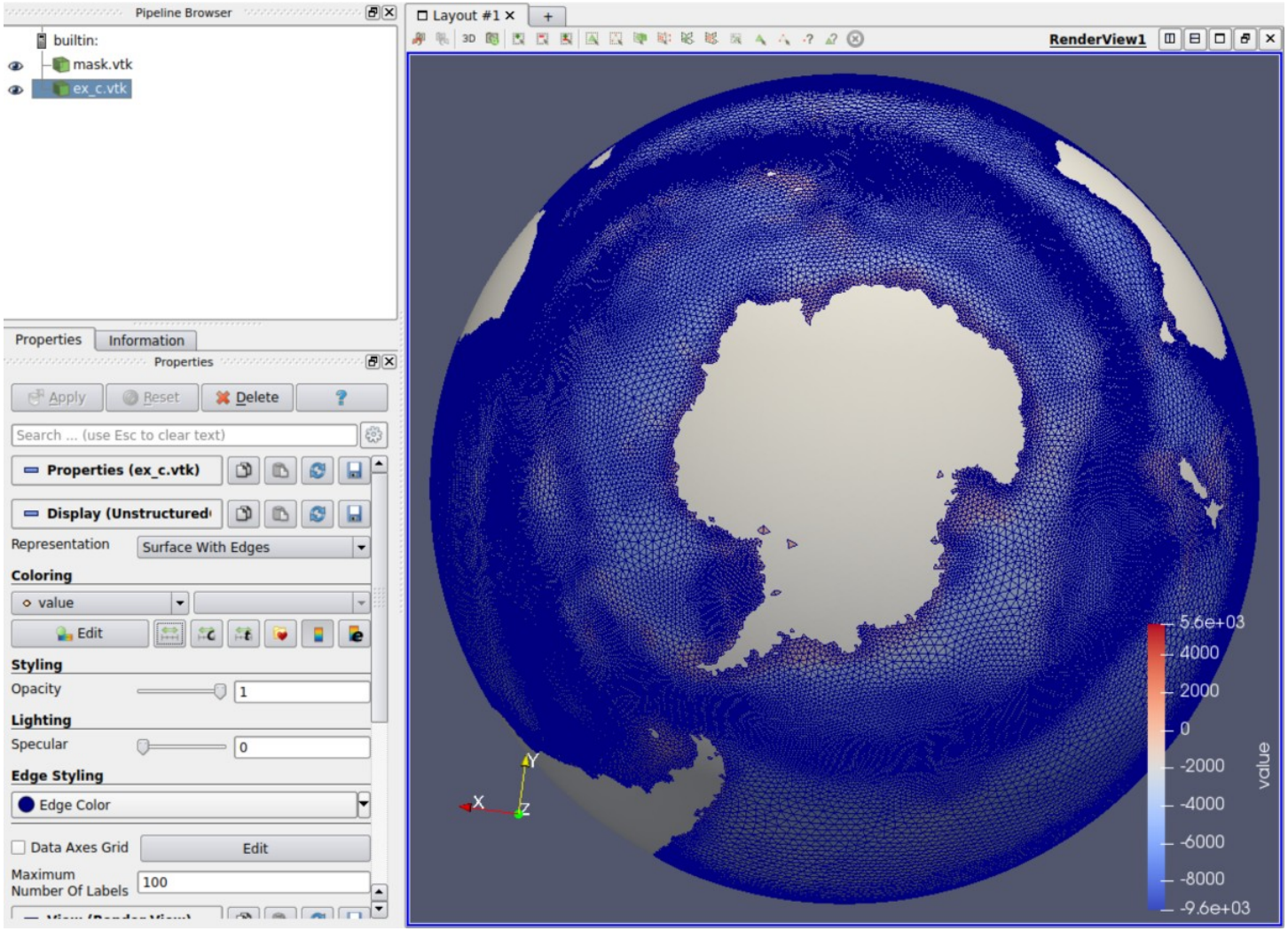
The third example is a global grid with very non-uniform mesh spacing:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_3;

(will run example-3)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_3.py

(will run example-3)
(and save ex_3.vtk file)

open ex_3.vtk in Paraview
```

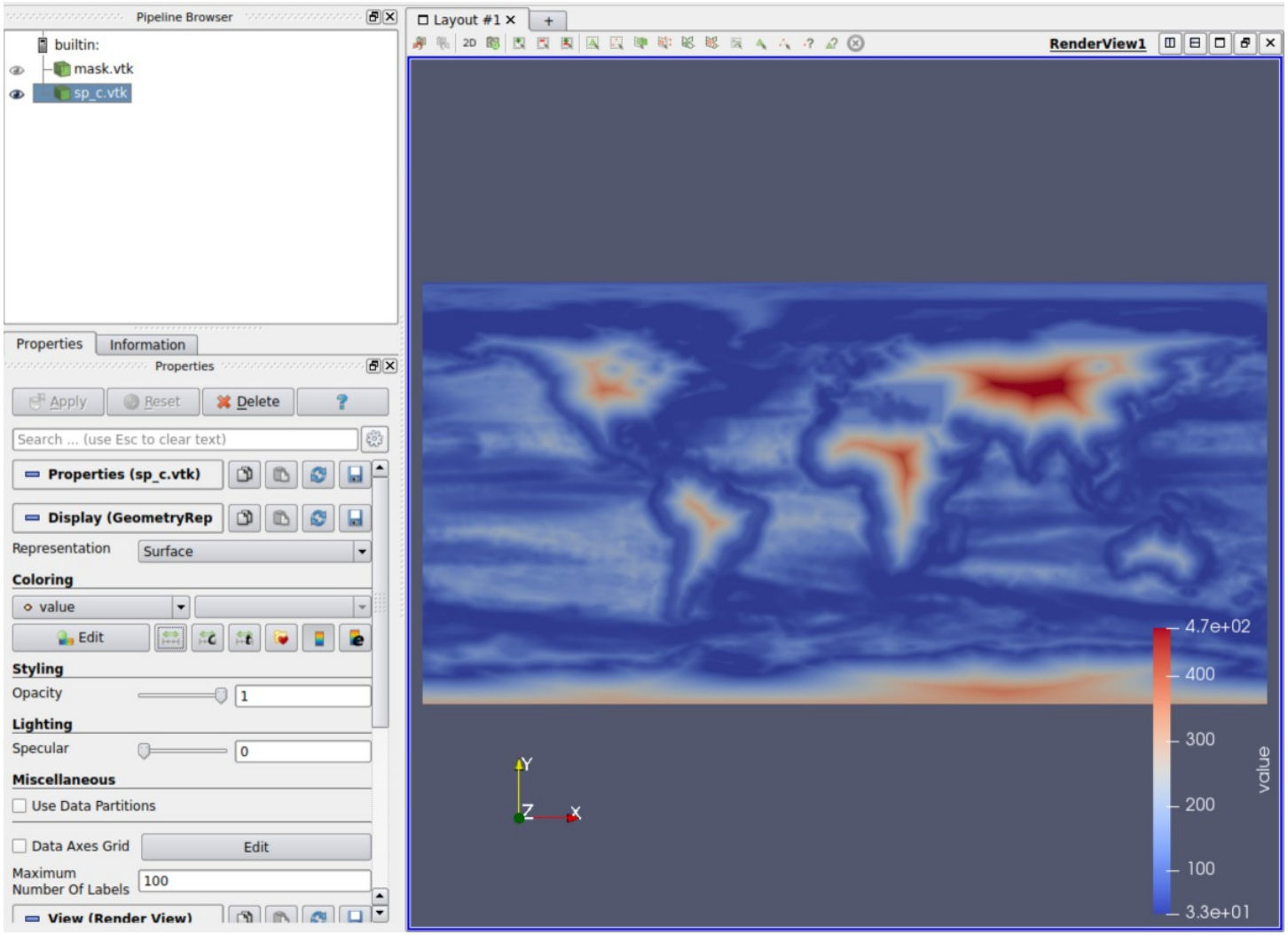
The ‘HR’ resolution pattern used in this example is based on an ‘SSH-variability’ metric, developed by the FESOM team at AWI, as per Sein et al.: Designing variable ocean model resolution based on the observed ocean variability, JAMES, 2016.

Example 3: output



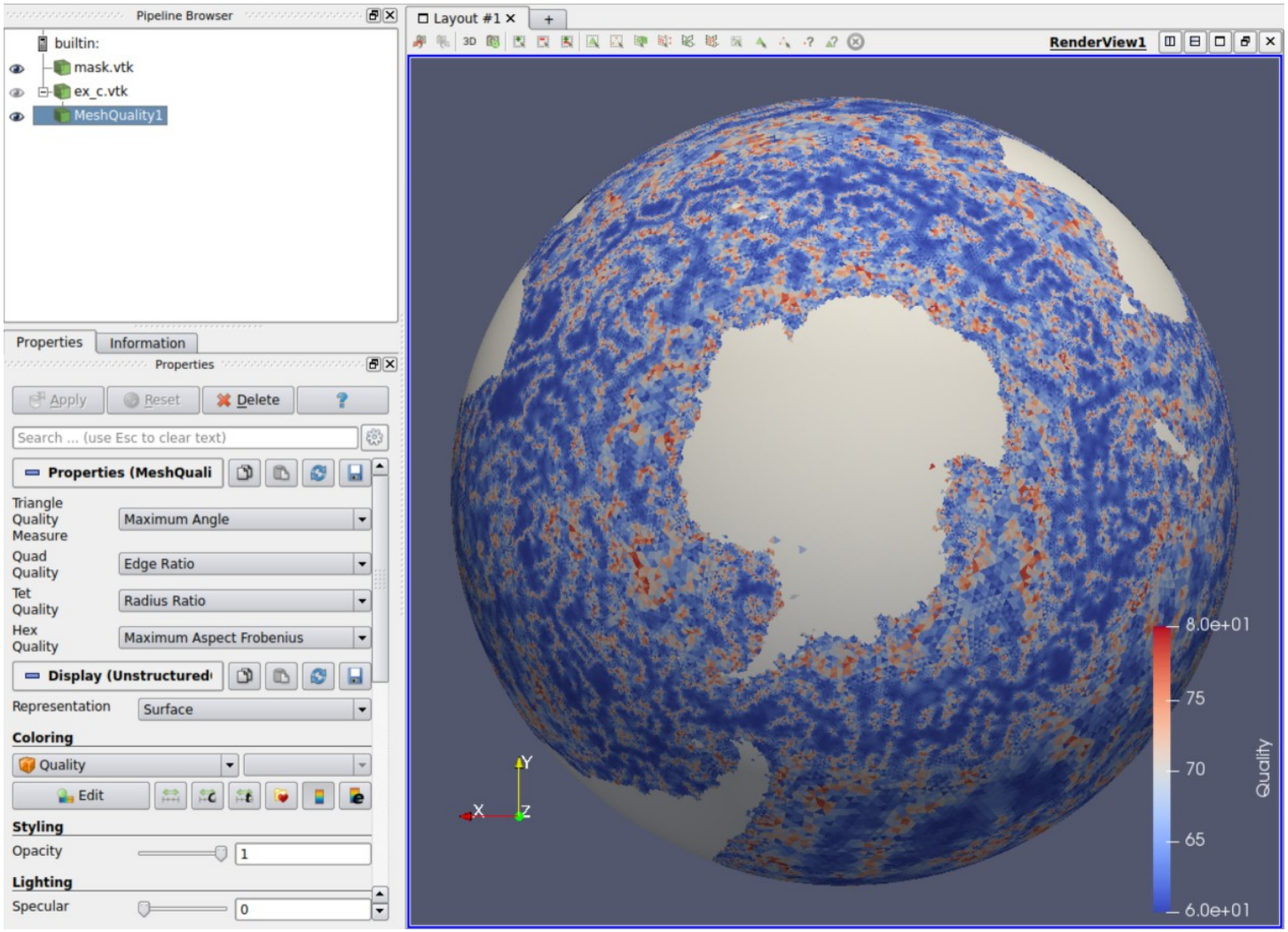
A complex refinement pattern is induced, leading to an 'eddy-enhanced' variable resolution configuration. Simulation quality was assessed using FESOM, as per Sein et al., 2016.

Example 3: output



The resolution pattern can be visualised in `Paraview` in lon-lat space.

Example 3: output



Despite the complexity, mesh-quality is still very high, with max. $\theta_\tau \leq 80^\circ$.

Example 4:

Improved regularity via ‘multi-level’ mesh generation.

Example 4:

The fourth example uses JIGSAW's (new) multi-level meshing scheme, TETRIS:

MATLAB (in cmd window):

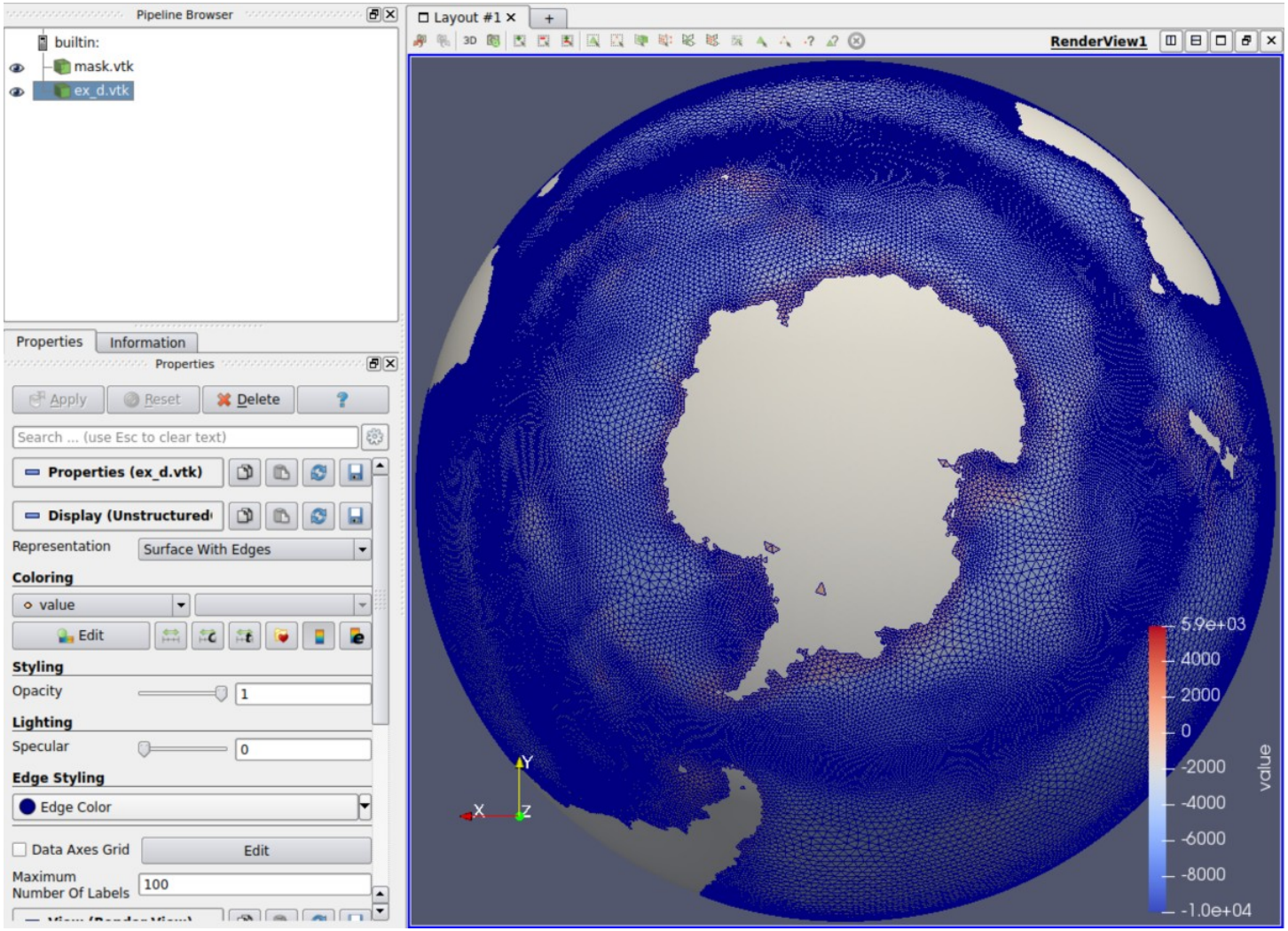
```
cd ..
cd examples-matlab

ex_4;

(will run example-4)
(and display in MATLAB)
```

Python (in system shell):

```
cd ..
cd examples-python

python3 ex_4.py

(will run example-4)
(and save ex_4.vtk file)

open ex_4.vtk in Paraview
```

By making multiple calls to the underlying JIGSAW solver, TETRIS assembles global grids with improved topology (i.e. more hexagons, etc).

Example 4: output



Mesh topology is more regular — larger number of $|\text{degree}| = 6$ nodes $\rightarrow$ hexagonal dual (Voronoi) cells.

Example 4: analysis

Comparing `ex_3.py` and `ex_4.py`, the only difference is the use of the `JIGSAW` vs `TETRIS` commands:

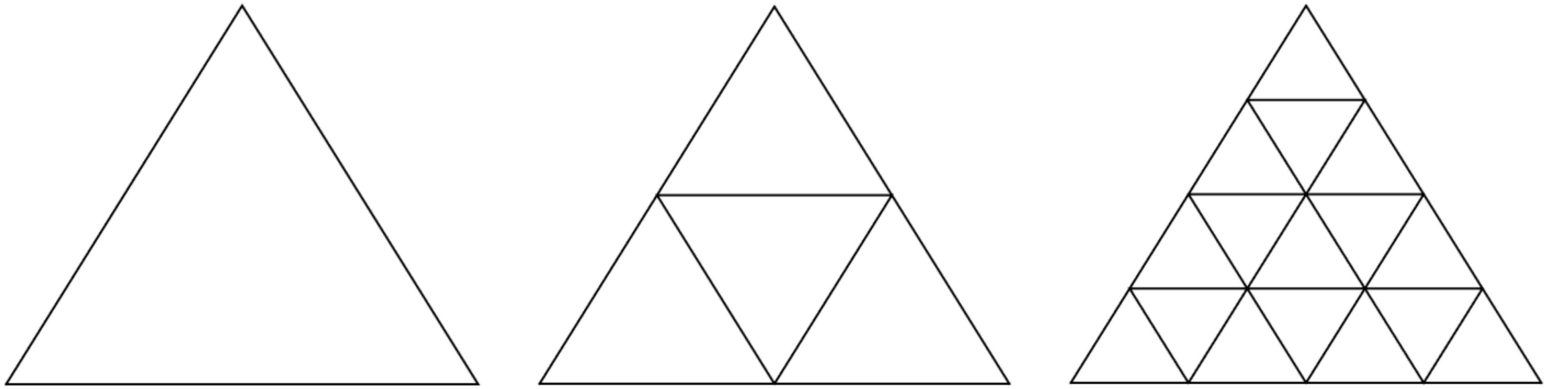
```
73    #jigsawpy.cmd.jigsaw(opts, mesh)
74     jigsawpy.cmd.tetris(opts, 2, mesh)
75
```

`TETRIS` can be called using identical inputs to `JIGSAW`, while also specifying the number of 'semi-structured' refinement operations to be performed (here, $N = 2$).

With `TETRIS`, meshes are generated as a 'hierarchy' — building a 'coarse' mesh using `JIGSAW` that is **bisected**, **refined** and **optimised** over N levels. The 'standard' `JIGSAW` solver is called at each level to do the refinement + optimisation.

Because bisection is a 'structured' operation (each triangle is split into 4 sub-triangles), the overall mesh topology can become more regular.

Example 4: analysis



'Bisection', wrt. triangle meshes, means splitting each triangle into 4 sub-triangles in a structured fashion.

In `TETRIS`, bisection operations are sandwiched between `JIGSAW`'s typical refinement + optimisation passes.

Example 4: analysis

Looking at the output for `ex_3.py` / `ex_4.py` in detail:

`TETRIS` (N = 0): 90.0% hexagons
`TETRIS` (N = 1): 95.9% hexagons
`TETRIS` (N = 2): 97.3% hexagons
`TETRIS` (N = 3): 97.4% hexagons

Eventually, too many bisection levels tend to compromise the (geometric) quality of the mesh $\rightarrow$ the grid becomes 'too-structured' and can no longer change resolution effectively.

N = 2, 3 seems to be a reasonable choice in practice.

Example 5:

Building high-quality mesh-spacing functions.

Example 5:

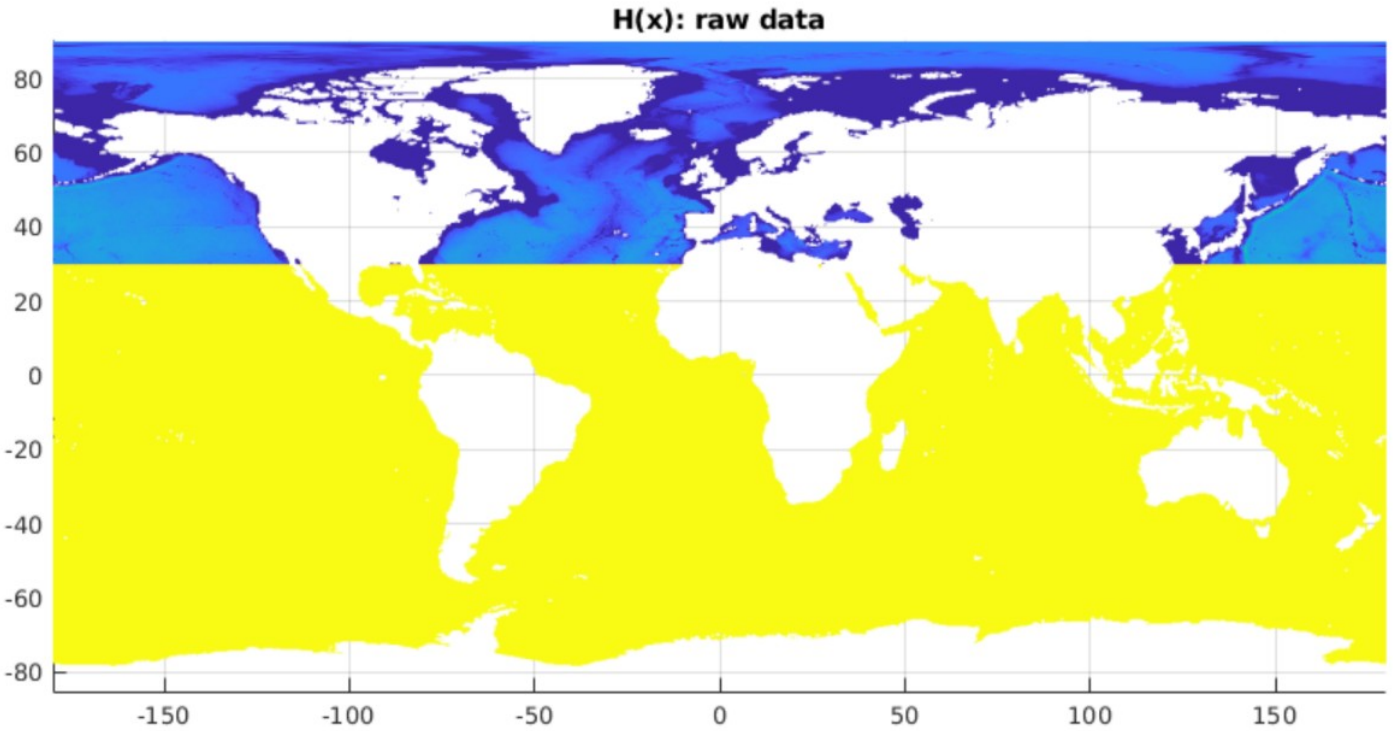
The fifth example uses JIGSAW's (new-ish) 'gradient-limiting' solver, MARCHE:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_5;

(will run example-5)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_5.py

(will run example-5)
(and save ex_5.vtk file)

open ex_5.vtk in Paraview
```

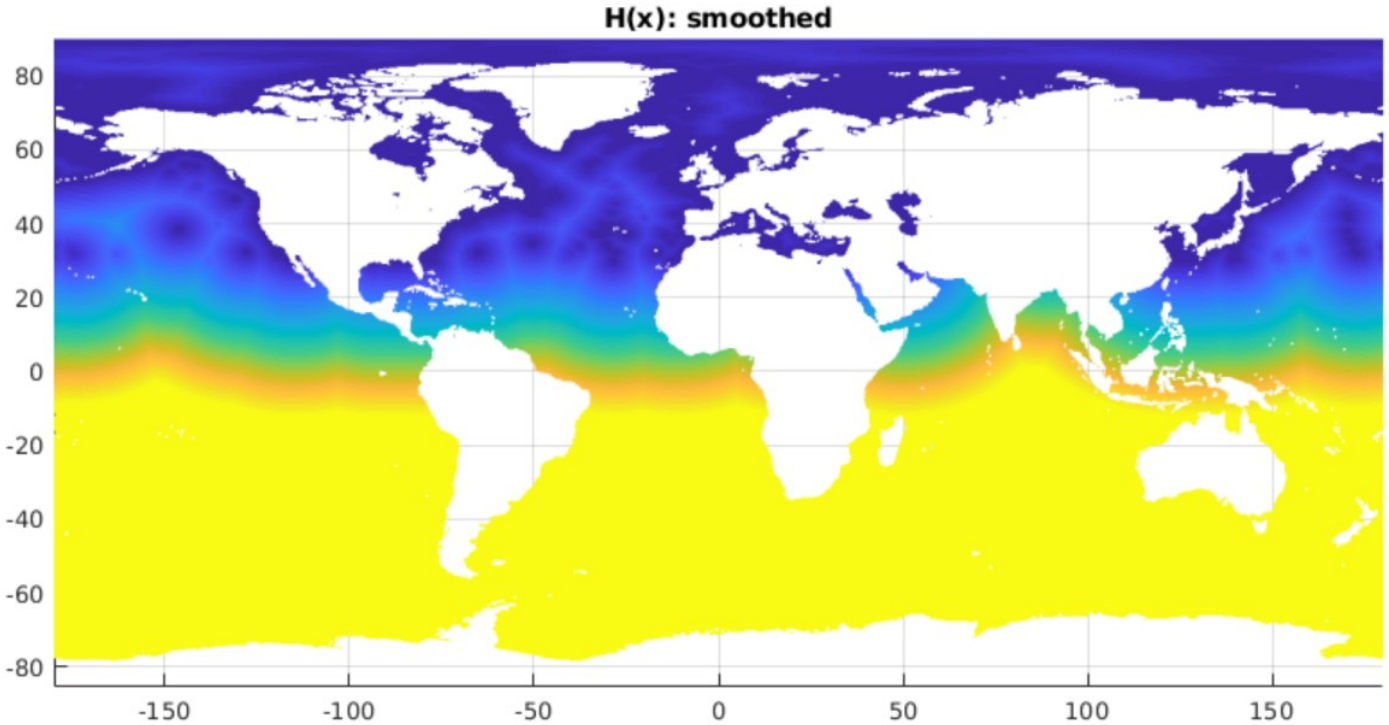
By solving the Eikonal equations via a 'fast-marching' method, MARCHE can be used to generate smooth, high-quality mesh-spacing distributions $\bar{h}(\mathbf{x})$ from 'rough' input data.

Example 5: output



A piecewise discontinuous spacing definition — refine with bathymetry in the polar region + constant spacing elsewhere.

Example 5: output



Imposing a 'gradient-limit' $|\mathrm{grad}(\bar{h})| \leq \beta(\mathbf{x})$ leads to a smooth, well-behaved function with bounded variability.

Example 5: workflow

Working through `ex_5.m` in detail:

```
35 %---------------------------------- define spacing pattern
36
37     topo = loadmsh( ...
38         fullfile(rootpath, ...
39             'files', 'topo.msh')) ;
40
41     xpos = topo.point.coord{1};
42     ypos = topo.point.coord{2};
43     zlev = topo.value;
44
45    [XPOS,YPOS] = meshgrid (xpos,ypos) ;
46
47     hfn0 = +150. ;                      % global spacing
48     hfn2 = +33.;                        % adapt. spacing
49     hfn3 = +99.;                        % arctic spacing
50
51     hfun = +hfn0*ones(size(zlev)) ;
52
53     htop = sqrt(max(-zlev(:),eps))/1. ;
54     htop = max(htop,hfn2);
55     htop = min(htop,hfn3);
56     htop(zlev>0.) = hfn0 ;
57
58     hfun(YPOS>=30.) = htop(YPOS>=30.) ;
59
```

Load global topography from file, scale resolution with depth $\alpha\sqrt{gH}$, and impose constant spacing south of 30°N and over land $\rightarrow$ discontinuous $\bar{h}(\mathbf{x})$.

Example 5: workflow

Set up mesh-spacing $\bar{h}(\mathbf{x})$ on sphere:

```
60 %---------------------------------- set HFUN grad.-limiter
61
62     dhdx = +.025;                        % max. gradients
63
64     hraw.mshID = 'ELLIPSOID-GRID' ;
65     hraw.radii = geom.radii ;
66     hraw.point.coord{1} = xpos*pi/180 ;
67     hraw.point.coord{2} = ypos*pi/180 ;
68     hraw.value = single(hfun) ;
69     hraw.slope = dhdx*ones(size(hfun));
70
71     savemsh (opts.hfun_file,hraw) ;
72
```

Call `MARCHE` (see `help marche`, etc) to solve $|\mathrm{grad}(\bar{h})| \leq \beta(\mathbf{x})$:

```
73 %---------------------------------- set HFUN grad.-limiter
74
75     hlim = marche(opts) ;
76
```

Try changing the `dhdx` $= \beta(\mathbf{x})$ threshold.

Offers an easy way to control variation in $\bar{h}(\mathbf{x}) \rightarrow$ 'smoothness' of mesh.

Example 6:

Building global grids using a MARCHE-limited mesh-spacing function.

Example 6:

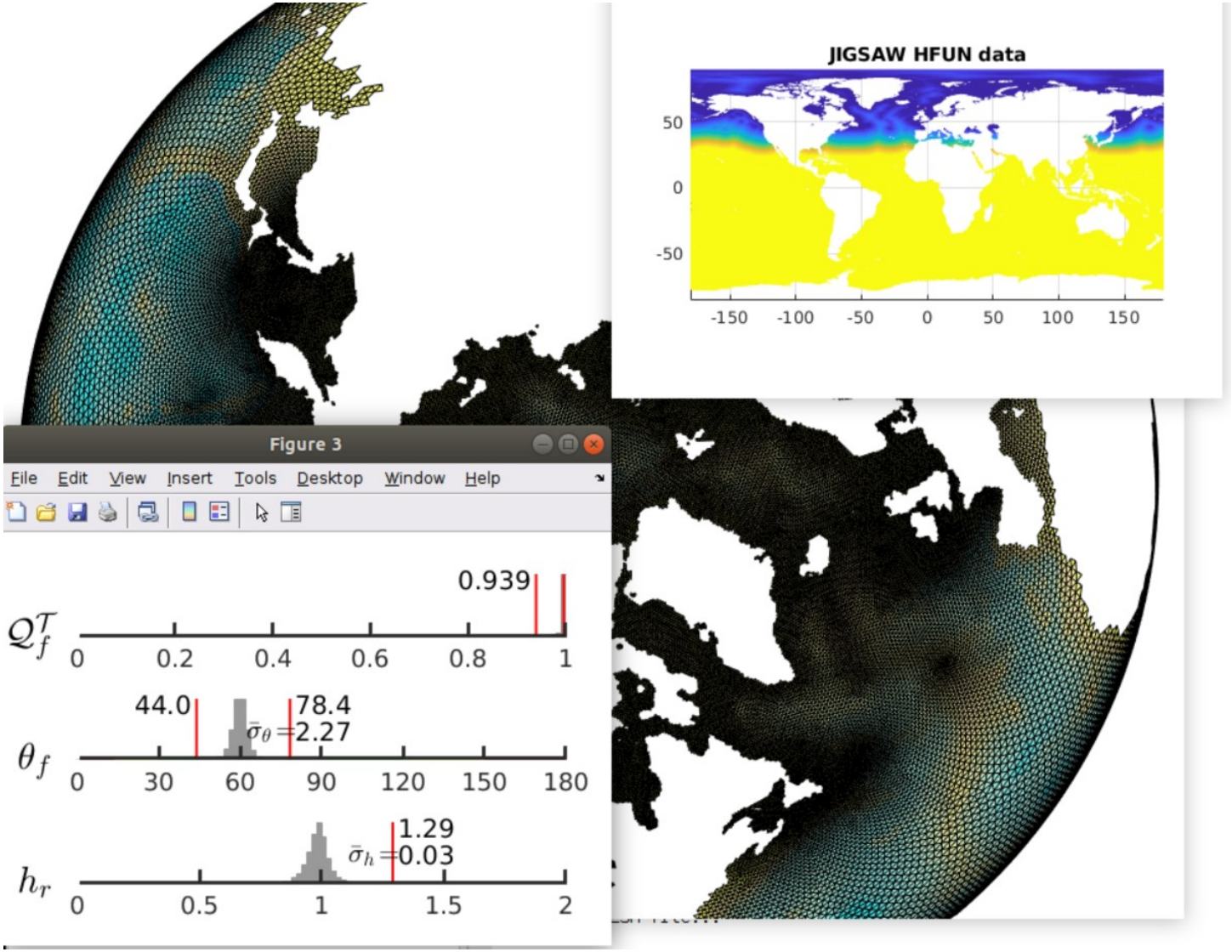
The sixth example uses the last result to build a 'shelf-enhanced' arctic mesh:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_6;

(will run example-6)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_6.py

(will run example-6)
(and save ex_6.vtk file)

open ex_6.vtk in Paraview
```

This example constitutes a typical 'full' JIGSAW workflow for global grids: define geometry, setup 'raw' $\bar{h}(\mathbf{x})$ data, smooth using MARCHE, and mesh using TETRIS.

Example 6: output



Try changing the threshold `dhdx` $= \beta(\mathbf{x})$ and explore how the mesh changes.

Example 7:

Building grids with boundaries in the plane.

Example 7:

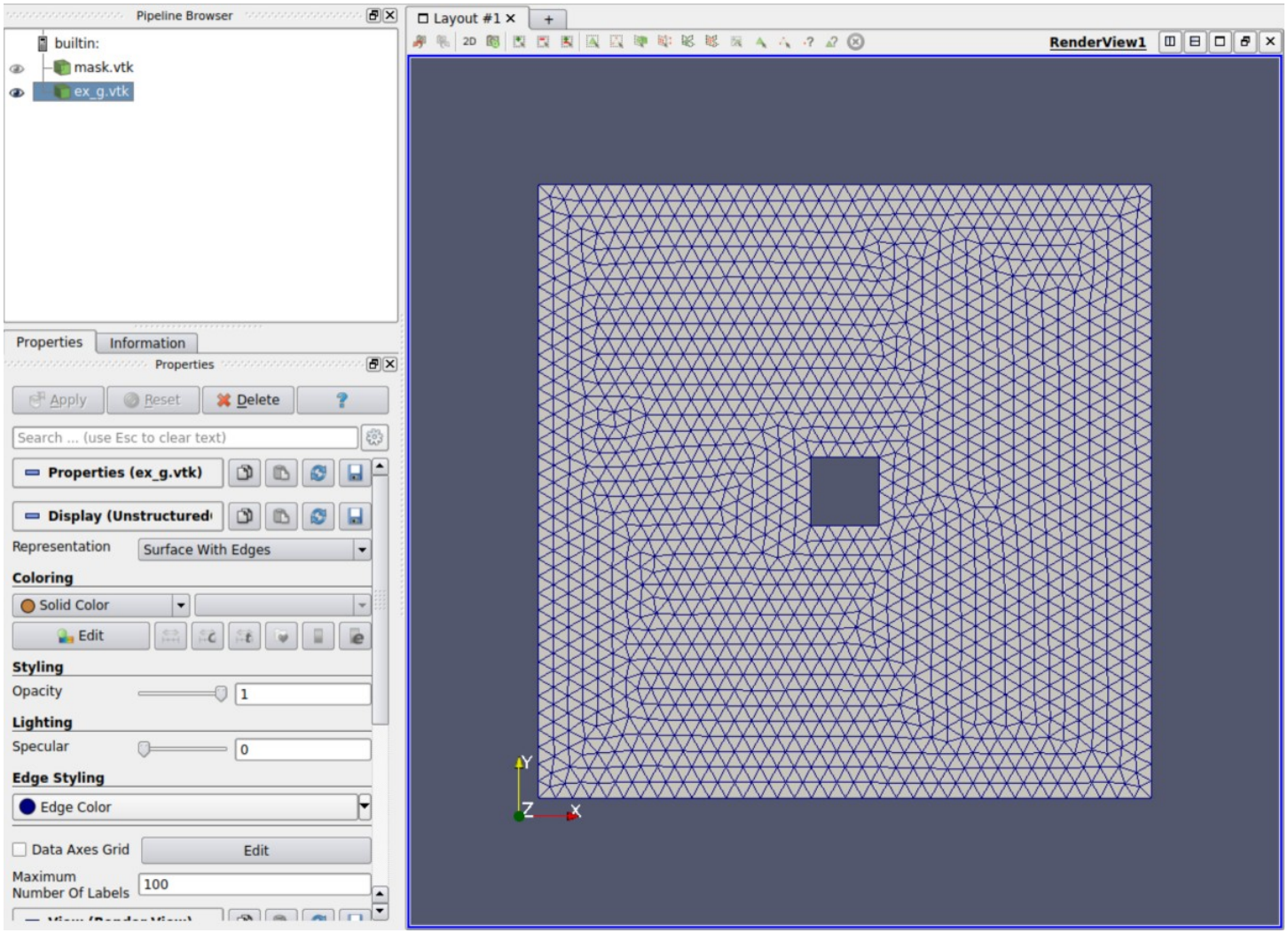
The seventh example deals with the definition of boundaries in planar meshes:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_7;

(will run example-7)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_7.py

(will run example-7)
(and save ex_7.vtk file)

open ex_7.vtk in Paraview
```

This example introduces very simple boundaries as collections of vertices and edges.

Example 7: output



A planar grid that conforms to the piecewise linear boundary definition.

Example 7: workflow

Working through `ex_7.py` in detail:

```
38 #---------------------------------- define JIGSAW geometry
39
40     geom.mshID = "euclidean-mesh"
41     geom.ndims = +2
42     geom.vert2 = np.array([ # list of xy "node" coordinate
43       ((0, 0), 0),          # outer square
44       ((9, 0), 0),
45       ((9, 9), 0),
46       ((0, 9), 0),
47       ((4, 4), 0),          # inner square
48       ((5, 4), 0),
49       ((5, 5), 0),
50       ((4, 5), 0)] ,
51     dtype=jigsawpy.jigsaw_msh_t.VERT2_t)
52
53     geom.edge2 = np.array([ # list of "edges" between vert
54       ((0, 1), 0),          # outer square
55       ((1, 2), 0),
56       ((2, 3), 0),
57       ((3, 0), 0),
58       ((4, 5), 0),          # inner square
59       ((5, 6), 0),
60       ((6, 7), 0),
61       ((7, 4), 0)] ,
62     dtype=jigsawpy.jigsaw_msh_t.EDGE2_t)
63
```

The geometry is specified as a 'piecewise linear complex' — a collection of vertices and edges. Each edge is define as a connection between two vertices (numbered implicitly based on their position in the array).

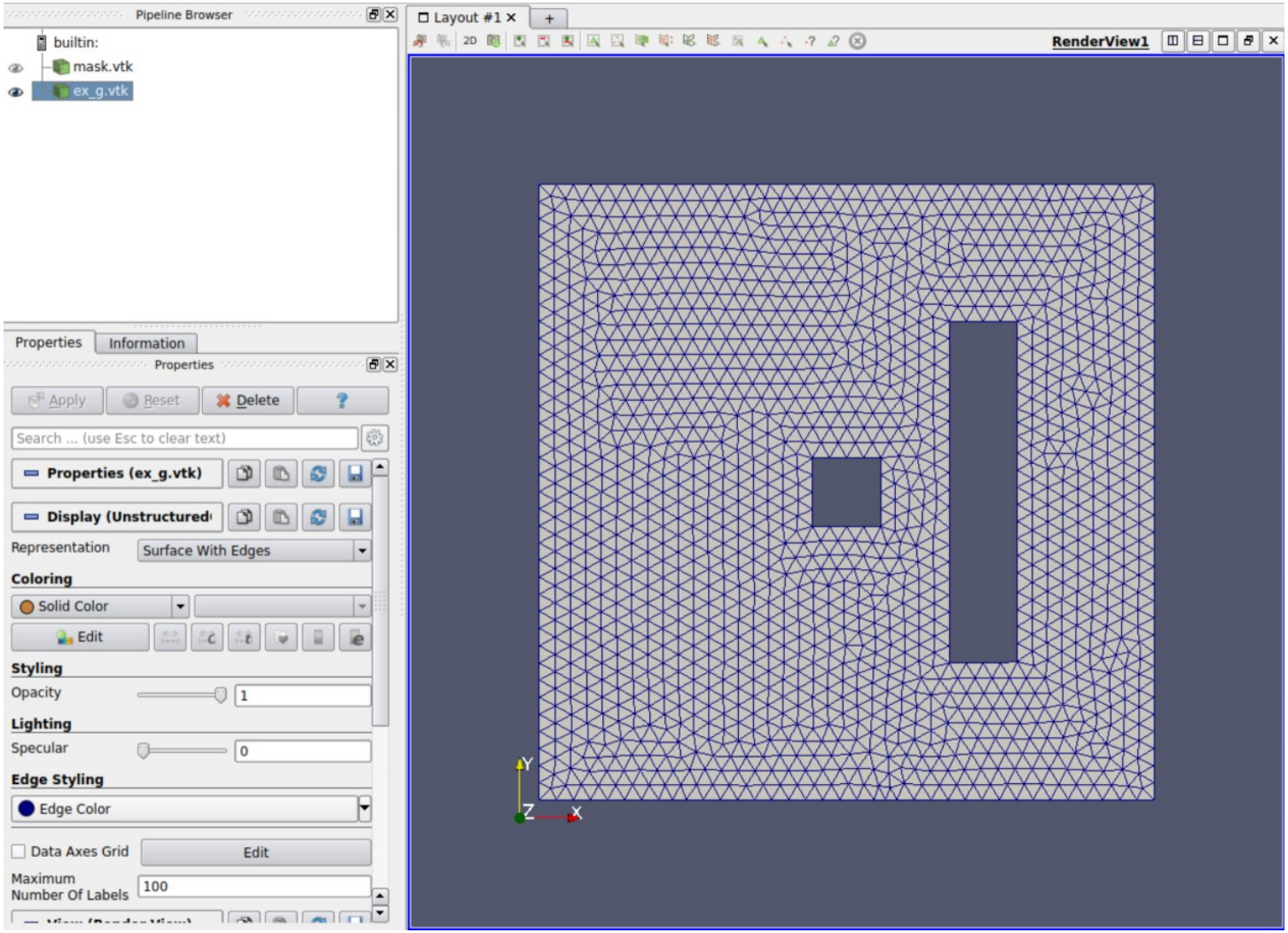
Example 7: workflow

Working through `ex_7.py` in detail:

```
38 #---------------------------------- define JIGSAW geometry
39
40     geom.mshID = "euclidean-mesh"
41     geom.ndims = +2
42     geom.vert2 = np.array([ # list of xy "node" coordinate
43       ((0, 0), 0),          # outer square
44       ((9, 0), 0),
45       ((9, 9), 0),
46       ((0, 9), 0),
47       ((4, 4), 0),          # inner square
48       ((5, 4), 0),
49       ((5, 5), 0),
50       ((4, 5), 0)] ,
51     dtype=jigsawpy.jigsaw_msh_t.VERT2_t)
52
53     geom.edge2 = np.array([ # list of "edges" between vert
54       ((0, 1), 0),          # outer square
55       ((1, 2), 0),
56       ((2, 3), 0),
57       ((3, 0), 0),
58       ((4, 5), 0),          # inner square
59       ((5, 6), 0),
60       ((6, 7), 0),
61       ((7, 4), 0)] ,
62     dtype=jigsawpy.jigsaw_msh_t.EDGE2_t)
63
```

The last 'column' in the vertex / edge arrays is an `IDtag`. This can be used to propagate integer 'tags' to a mesh. Typically, set `IDtag` $= 0$.

Example 7: output



Try adding / removing geometry components by modifying the vertex and edge arrays.

Example 8:

Building grids for regional domains (I): basic setup.

Example 8:

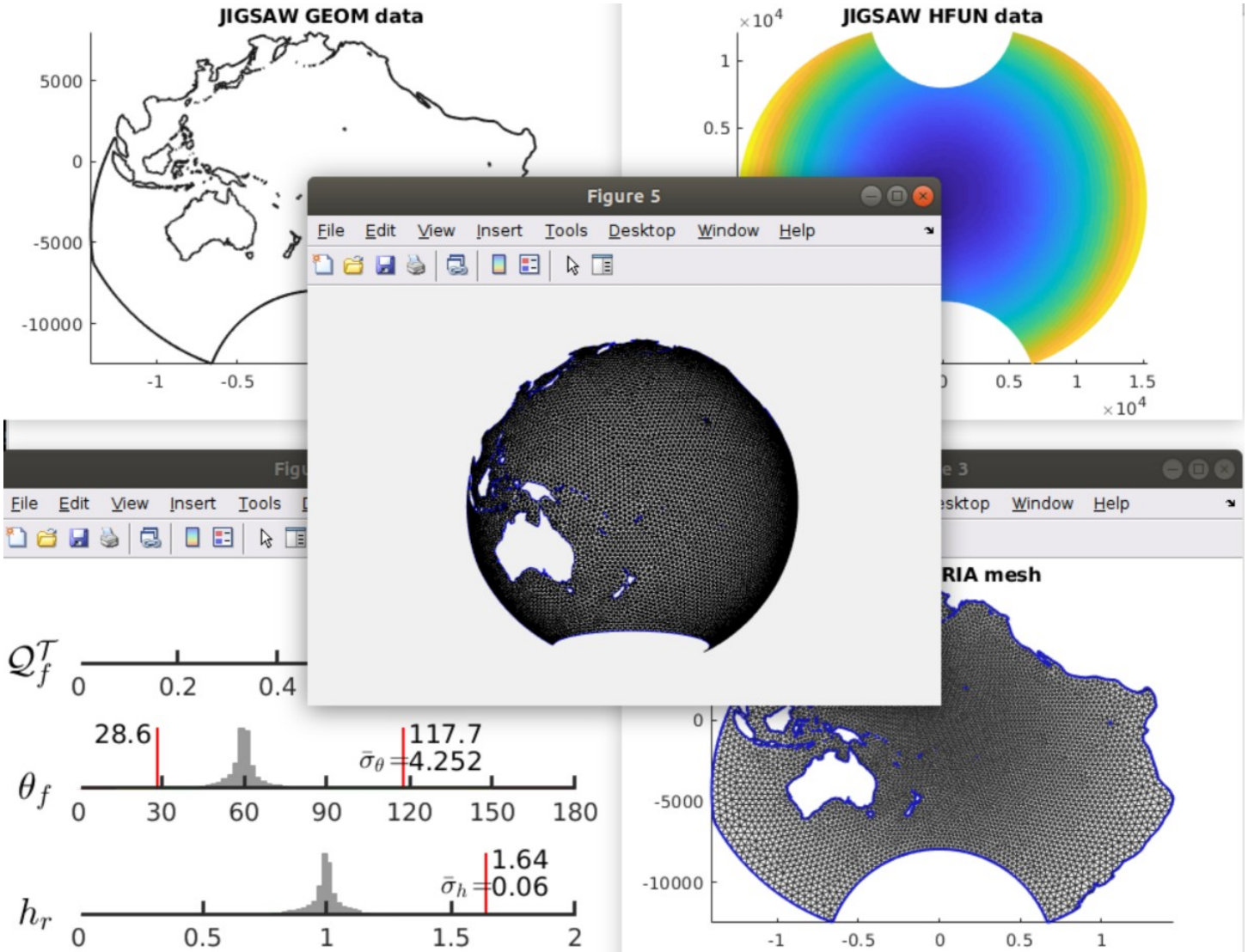
The eighth example deals with regional domains and projections:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_8;

(will run example-8)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_8.py

(will run example-8)
(and save ex_8.vtk file)

open ex_8.vtk in Paraview
```

This example uses stereographic projections to create meshes for the pacific basin.

Example 8: output



A range of output! Spherical + stereographic meshes, geometry, $\bar{h}(\mathbf{x})$, etc.

Example 8: workflow

Working through `ex_8.m` in detail:

```
29 %---------------------------------- define JIGSAW geometry
30
31     geom = loadmsh( ...
32         fullfile(rootpath, ...
33             'files', 'paci.msh')) ;
34
35     xmin = min(geom.point.coord(:,1)) ;
36     xmax = max(geom.point.coord(:,1)) ;
37     ymin = min(geom.point.coord(:,2)) ;
38     ymax = max(geom.point.coord(:,2)) ;
39
40     xlon = linspace(xmin, xmax, 100);
41     ylat = linspace(ymin, ymax, 100);
42
43 %---------------------------------- define spacing pattern
44
45     hval = 200.;
46
47     hfun.mshID = 'ELLIPSOID-GRID';
48     hfun.radii = 6371.E+00;
49     hfun.point.coord{1} = xlon * pi/180. ;
50     hfun.point.coord{2} = ylat * pi/180. ;
51     hfun.value = ...
52     hval*ones(length(ylat),length(xlon)) ;
53
```

Load coastlines from file, and create a uniform $\bar{h}(\mathbf{x})$ in lon-lat space.

Example 8: workflow

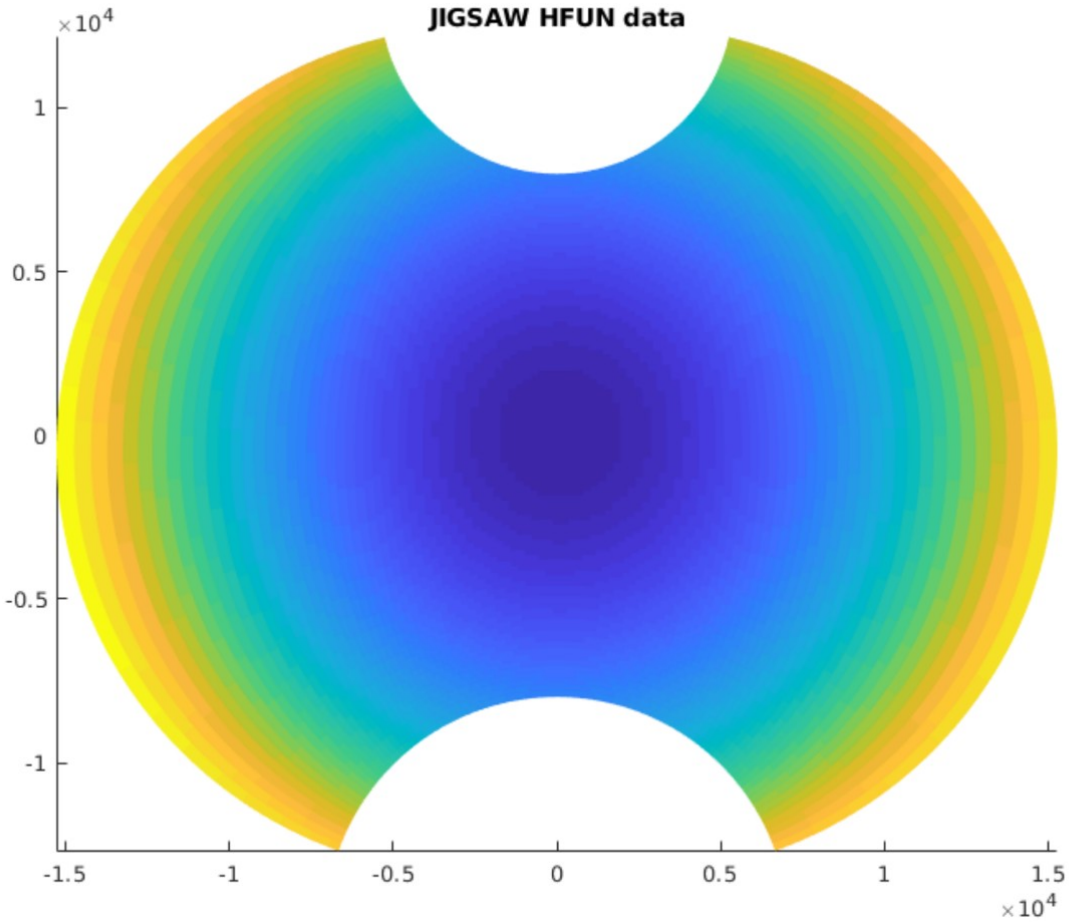
Working through `ex_8.m` in detail:

```
%---------------------------------- do stereographic proj.

    geom.point.coord(:,1:2) = ...
    geom.point.coord(:,1:2) * pi/180. ;

    proj.prjID = 'STEREOGRAPHIC' ;
    proj.radii = 6371.E+00;
    proj.xbase = .5 * ( ...
        min(geom.point.coord(:,1)) ...
      + max(geom.point.coord(:,1))) ;
    proj.ybase = .5 * ( ...
        min(geom.point.coord(:,2)) ...
      + max(geom.point.coord(:,2))) ;

    GEOM = project(geom,proj,'fwd') ;
    HFUN = project(hfun,proj,'fwd') ;

    savemsh (opts.geom_file,GEOM) ;
    savemsh (opts.hfun_file,HFUN) ;

```

Transform **both** the geometry **and** mesh-spacing using a stereographic projection.

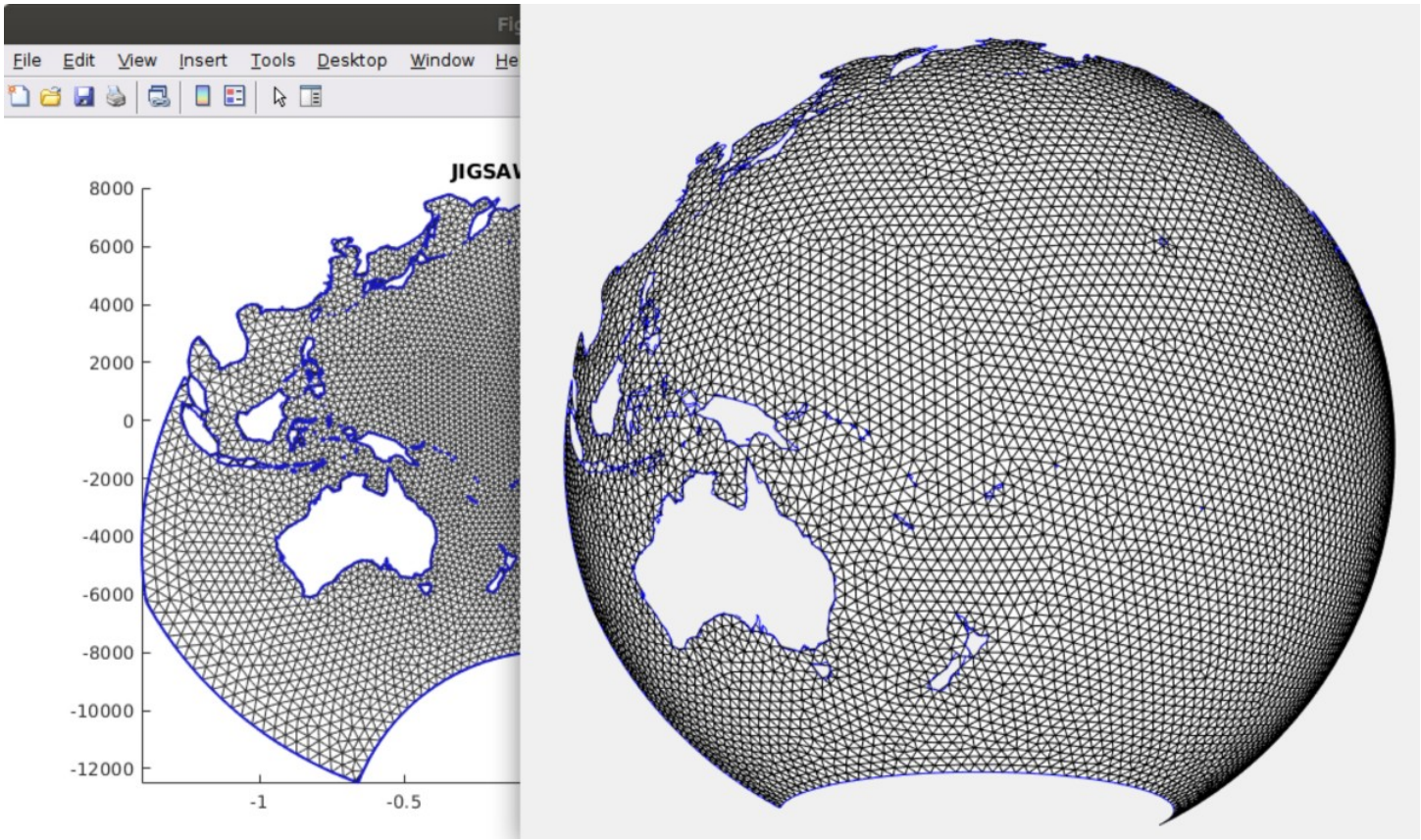
`JIGSAW` creates a mesh in stereographic space using these transformed inputs.

Example 8: output



Important to transform the spacing function to account for the 'stretching' induced by the projection — a uniform $\bar{h}(\mathbf{x})$ becomes non-uniform!

Example 8: output



This means that a **non-uniform** mesh is generated in stereographic space, which becomes **uniform** when mapped back onto the sphere.

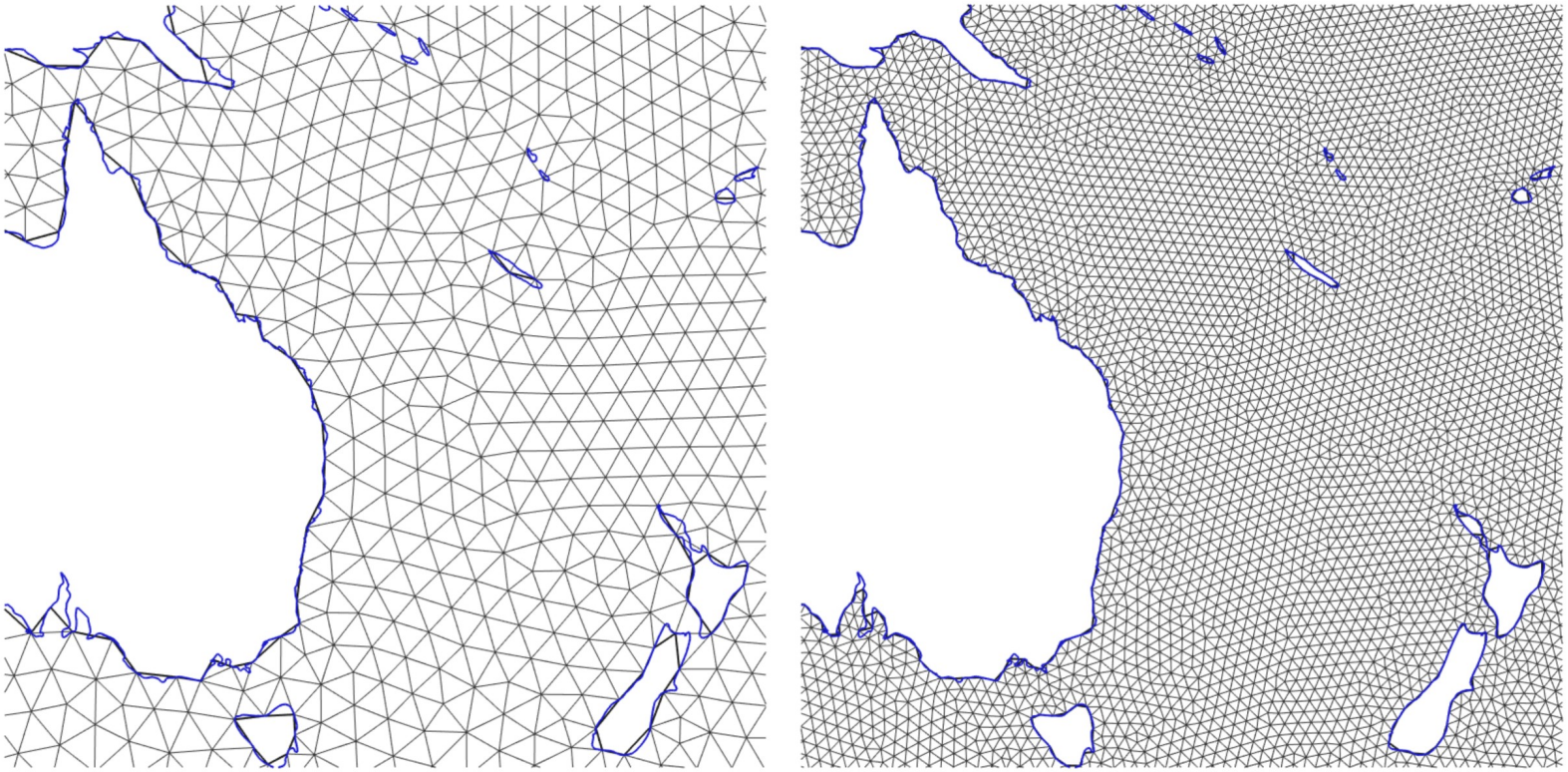
`JIGSAW` considers the projection's 'indicatrix' (i.e. differential form) to compute the stretching factors.

Example 8:

Building grids for regional domains (II): approximating coastlines?

Example 8: output

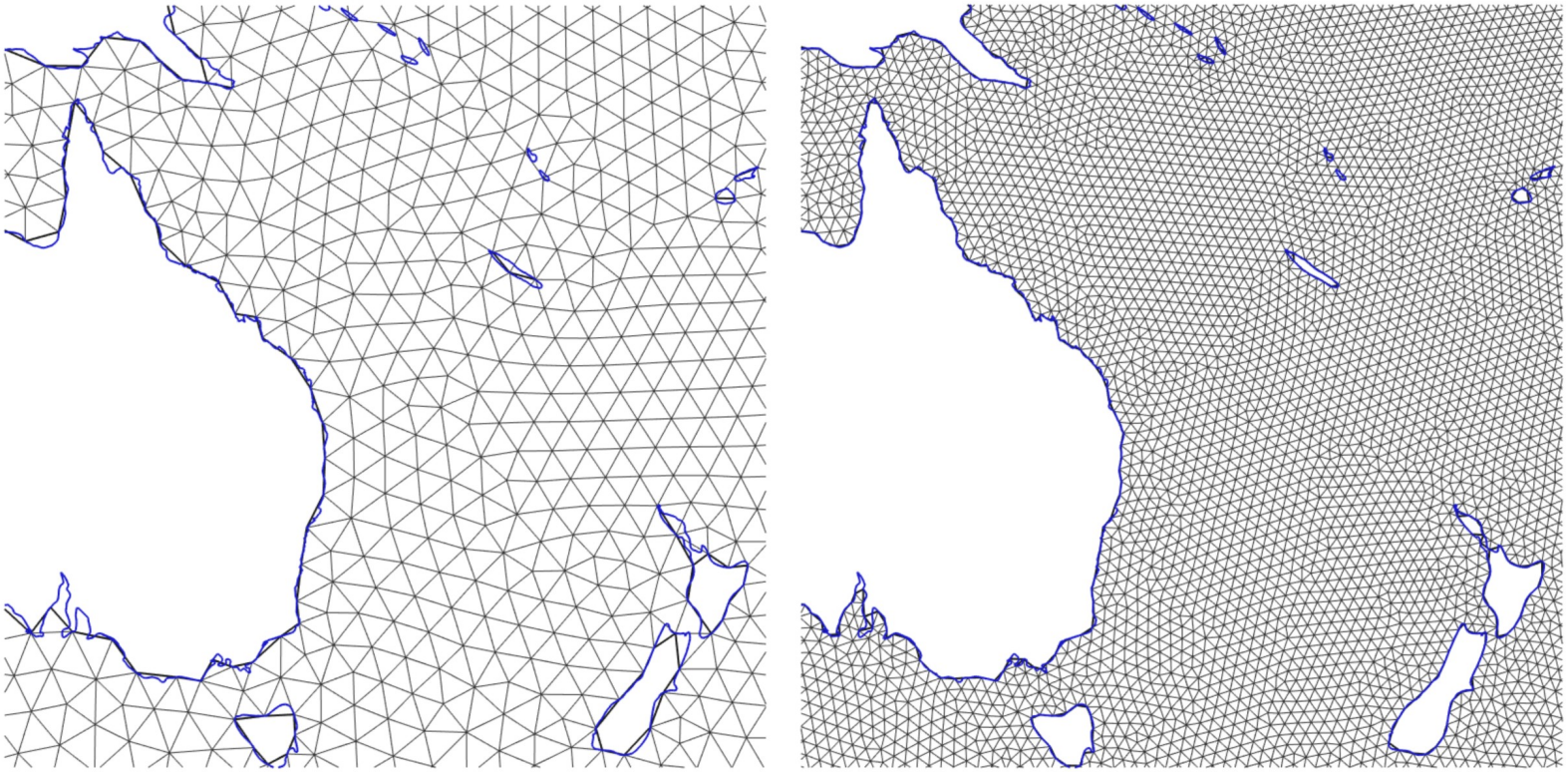
JIGSAW is based on a **multi-resolution** approach to mesh generation called 'restricted' Delaunay triangulation (compared to more conventional 'constrained' triangulation schemes).



The geometry (boundaries, coastlines, etc) is 're-sampled' at the mesh-spacing $\bar{h}(\mathbf{x})$ that's defined.

Example 8: output

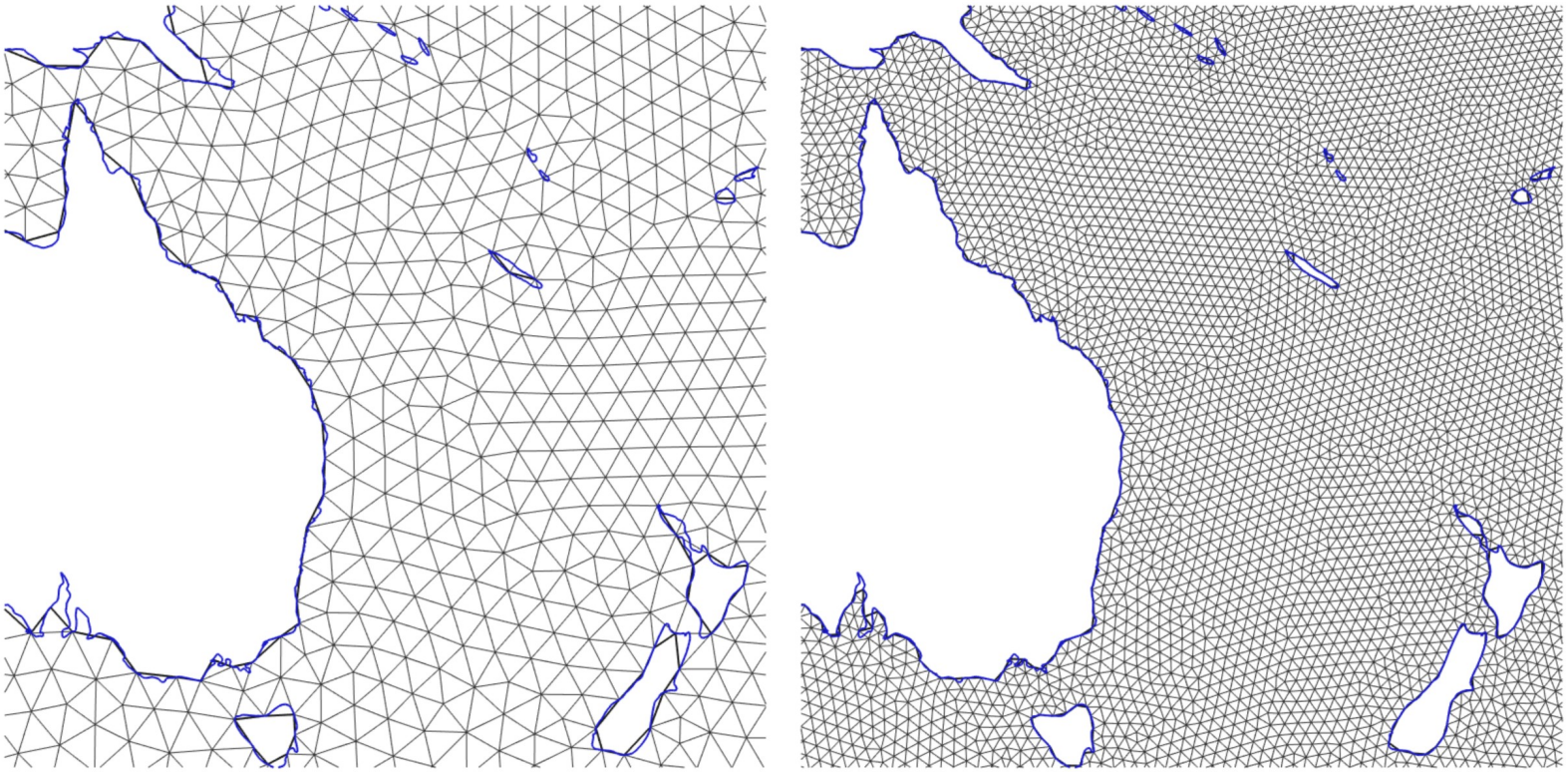
JIGSAW is based on a **multi-resolution** approach to mesh generation called 'restricted' Delaunay triangulation (compared to more conventional 'constrained' triangulation schemes).



Try changing the mesh-spacing $\bar{h}(\mathbf{x}) = 200, 100, 50$, etc.

Example 8: output

JIGSAW is based on a **multi-resolution** approach to mesh generation called 'restricted' Delaunay triangulation (compared to more conventional 'constrained' triangulation schemes).



'Multi-resolution' boundaries can help deal with some 'bad' aspects of geoscientific datasets — wrong resolution, sharp features, narrow channels, etc...

Example 9:

Building grids for coastal domains.

Example 9:

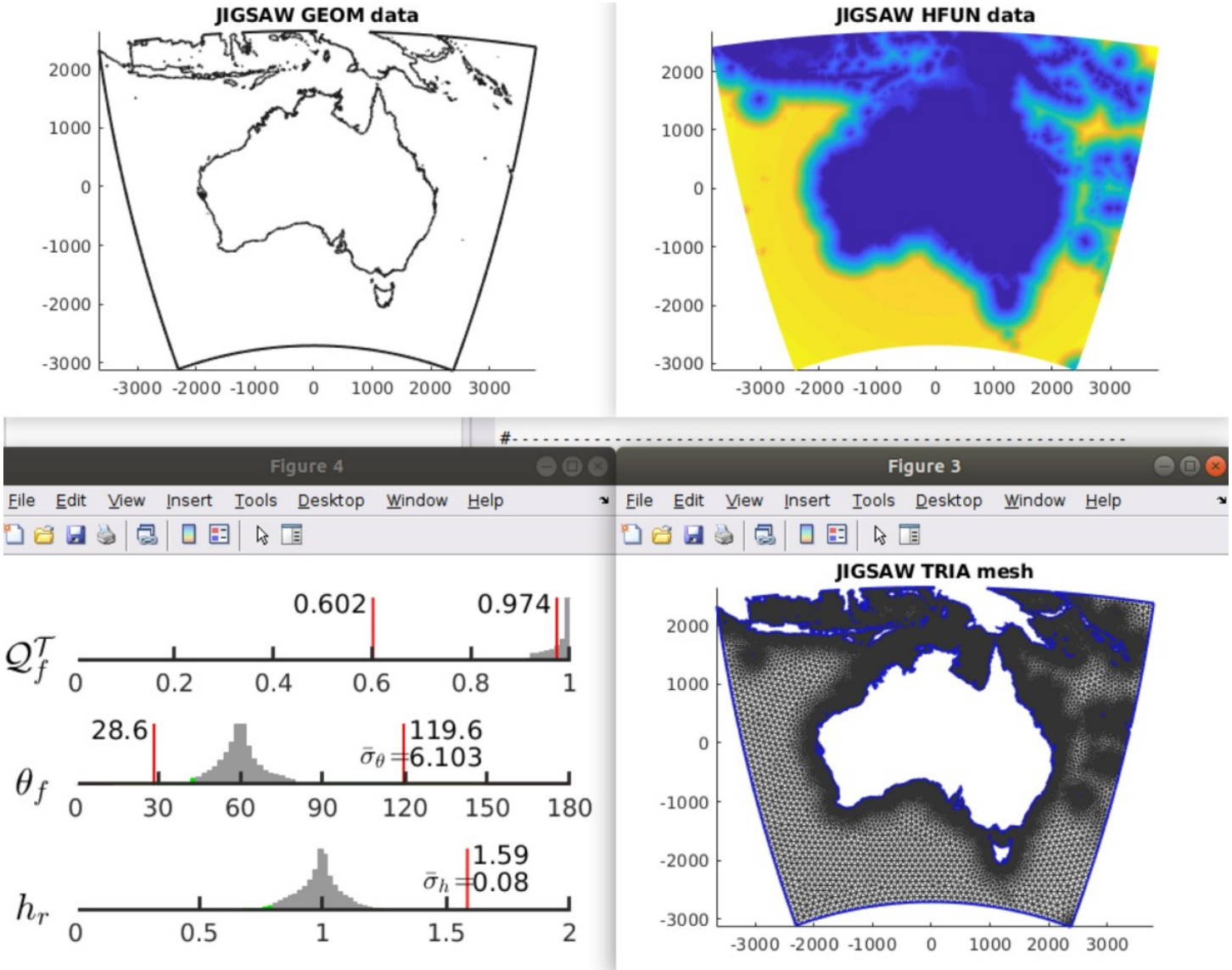
The last example deals with regional domains and projections:

```
MATLAB (in cmd window):

cd ..
cd examples-matlab

ex_9;

(will run example-9)
(and display in MATLAB)
```

```
Python (in system shell):

cd ..
cd examples-python

python3 ex_9.py

(will run example-9)
(and save ex_9.vtk file)

open ex_9.vtk in Paraview
```

This example outlines a coastal meshing workflow for the Australian region.

Example 9: output



As per `ex_8.m`, spherical + stereographic meshes, geometry, $\bar{h}(\mathbf{x})$, etc.

Example 9: workflow

Working through `ex_9.m` in detail:

```
%---------------------------------- define JIGSAW geometry

    geom = loadmsh(fullfile( ...
        rootpath,'files','aust.msh')) ;

    topo = loadmsh(fullfile( ...
        rootpath,'files','topo.msh')) ;

    xmin = min(geom.point.coord(:,1)) ;
    xmax = max(geom.point.coord(:,1)) ;
    ymin = min(geom.point.coord(:,2)) ;
    ymax = max(geom.point.coord(:,2)) ;

    xmsk = topo.point.coord{1} >= xmin ...
         & topo.point.coord{1} <= xmax ;
    ymsk = topo.point.coord{2} >= ymin ...
         & topo.point.coord{2} <= ymax ;

    xlon = topo.point.coord{1}(xmsk) ;
    ylat = topo.point.coord{2}(ymsk) ;
    zlev = topo.value(ymsk,xmsk) ;

```

Load coastlines and (coarse) bathymetry for the region.

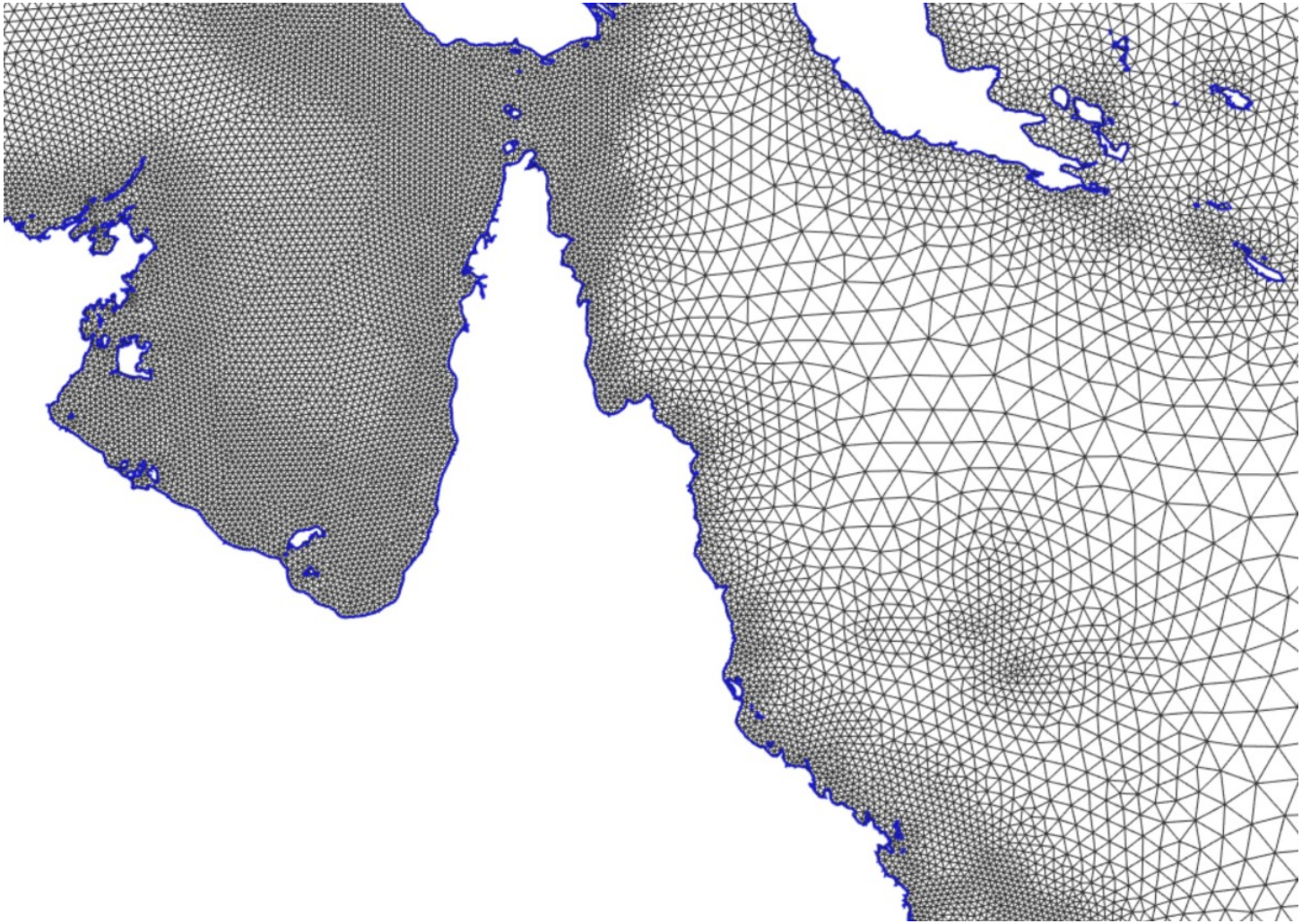
Example 9: workflow

Working through `ex_9.m` in detail:

```
53 %---------------------------------- define spacing pattern
54
55     hmin = +10. ;                          % min. H(X) [deg.]
56     hmax = +100. ;                         % max. H(X)
57
58     hmat = sqrt(max(-zlev,eps))/.5 ;       % scale with H^1/2
59     hmat = max(hmat,hmin);
60     hmat = min(hmat,hmax);
61
62     dhdx = +.175 *ones(size(hmat)) ;       % smoothing limits
63
64     hfun.mshID = 'ELLIPSOID-GRID';
65     hfun.radii = 6371.E+00;
66     hfun.point.coord{1} = ...
67                 xlon * pi / 180. ;
68     hfun.point.coord{2} = ...
69                 ylat * pi / 180. ;
70     hfun.value = hmat ;
71     hfun.slope = dhdx ;
72
```

Scale $\bar{h}(\mathbf{x})$ with estimated column-depth and call `MARCHE` to limit $\mathrm{grad}(\bar{h})$.

Example 9: output



Scaling $\bar{h}(\mathbf{x})$ with column-depth is a very simple heuristic for mesh design, typically, this would be combined with other metrics: distance-to-coast, bathymeric gradients, tidal amplitude, EKE, vorticity, shear stresses, etc!

Summary

That's it!

Hopefully, you now know how to build global, regional + coastal meshes with JIGSAW.

JIGSAW is a project under active development: quad-meshes, high-order elements, parallelism, more aggressive optimisation, etc...

Let me know if you have questions:

darren.engwirda@columbia.edu

https://github.com/dengwirda/jigsaw