

Generating meshes for MPAS-Atmosphere

Michael G. Duda
NSF NCAR/MMM



NCAR
OPERATED BY UCAR

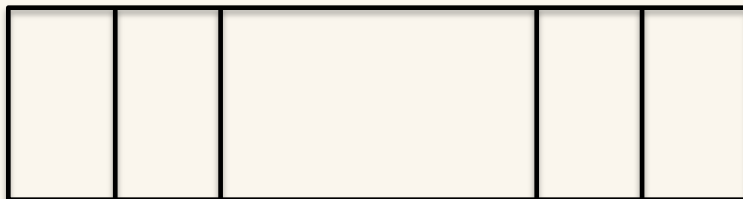
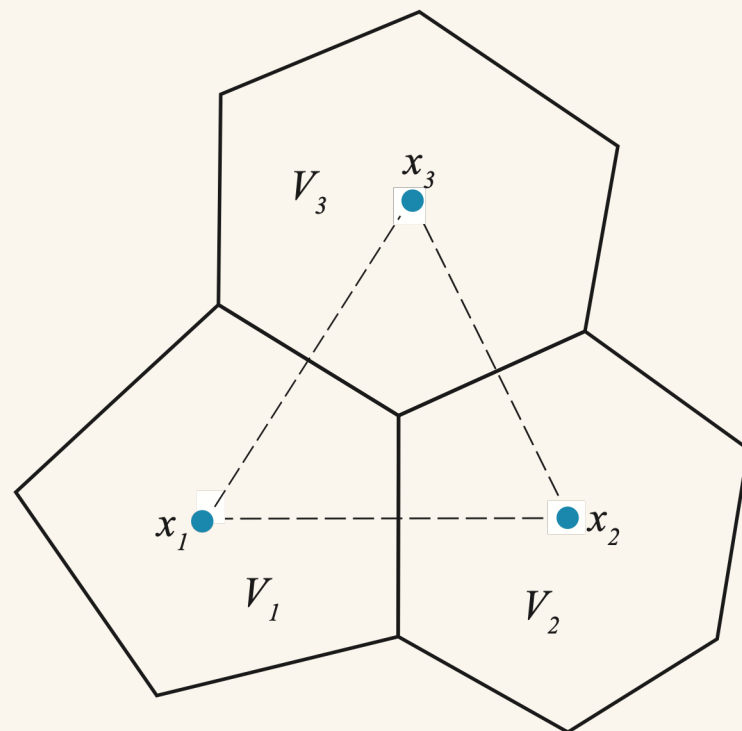
Outline

1. A quick refresher on Voronoi tessellations
2. The goals of mesh generation
3. Existing mesh tools
4. A simple JIGSAW-based workflow
5. Generating variable-resolution meshes
6. Generating quasi-uniform meshes

A quick refresher on Voronoi tessellations

Voronoi tessellation = each grid volume (cell) V_i is uniquely associated with a *generating point* x_i such that all points within V_i are closer to x_i than to any other x_j

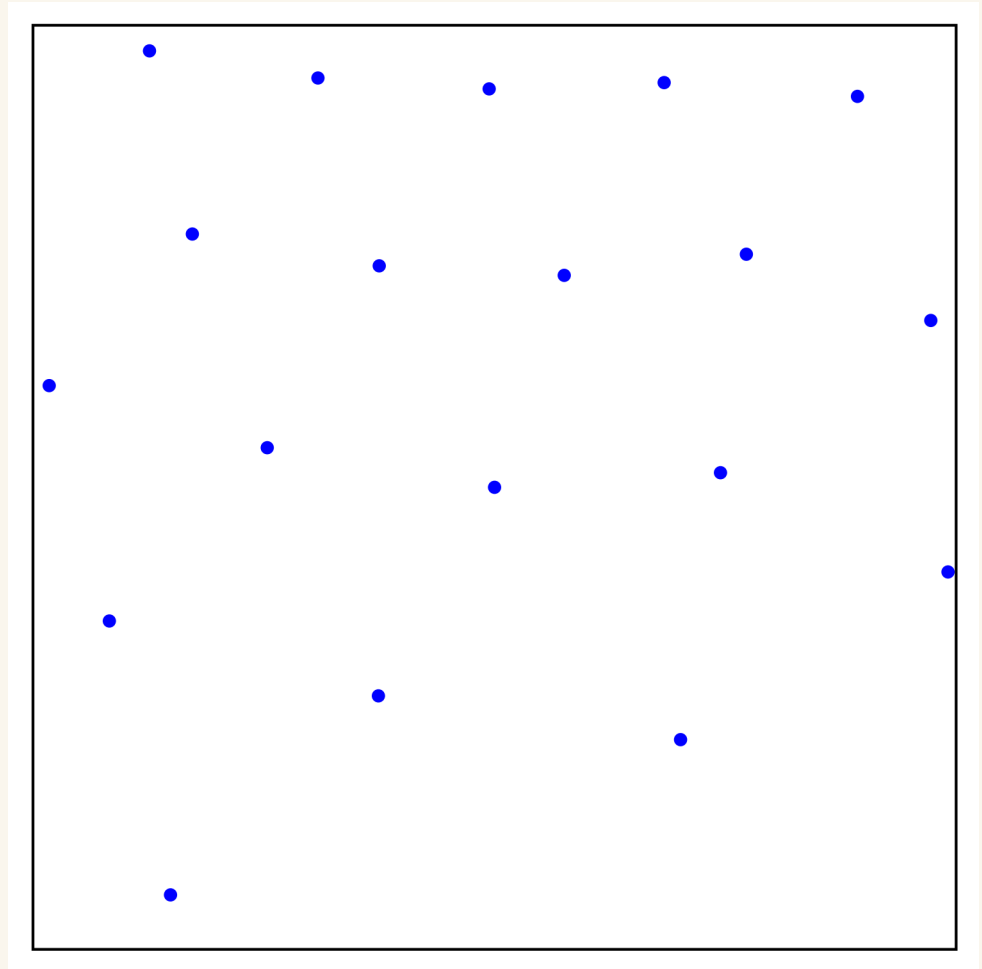
Not every tessellation is a Voronoi tessellation – it's the generating point set that defines a Voronoi tessellation, rather than the other way around.



Left: Does there exist a placement of generating points within regions such that Voronoi property holds?

Construction of C-staggered Voronoi tessellations

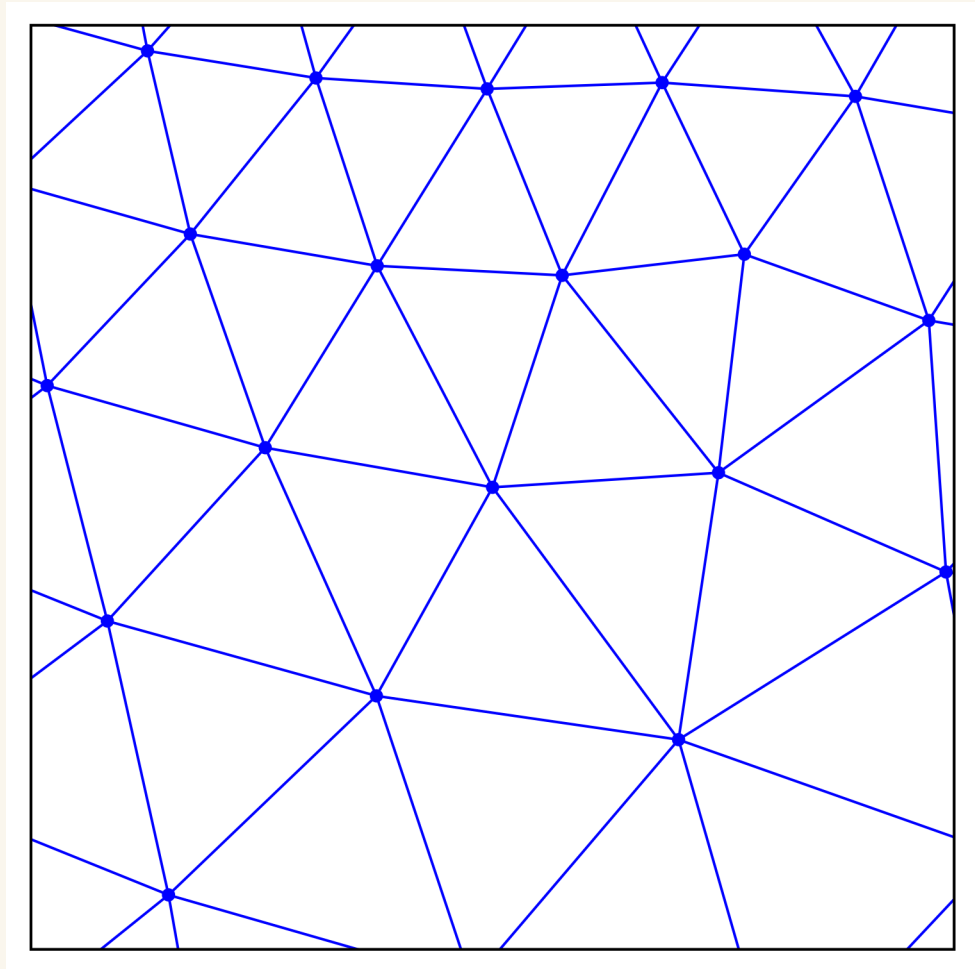
Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

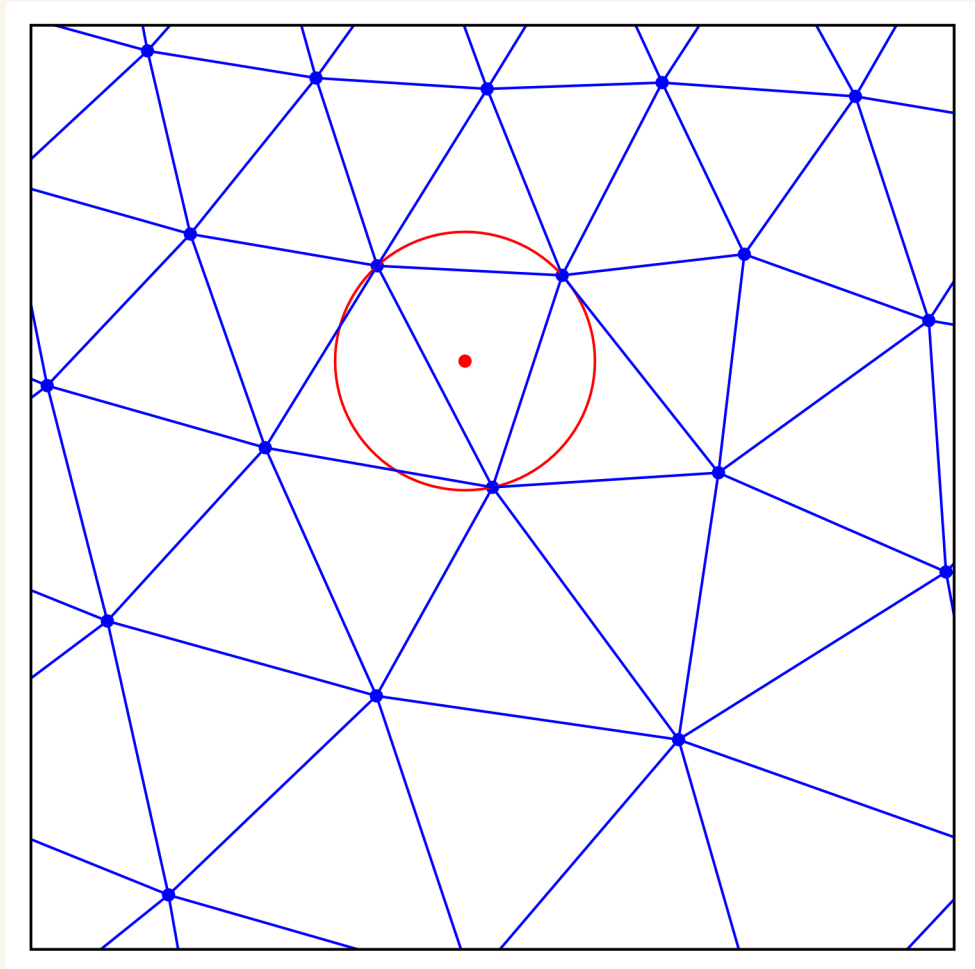
- Find the Delaunay triangulation of the generating points



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

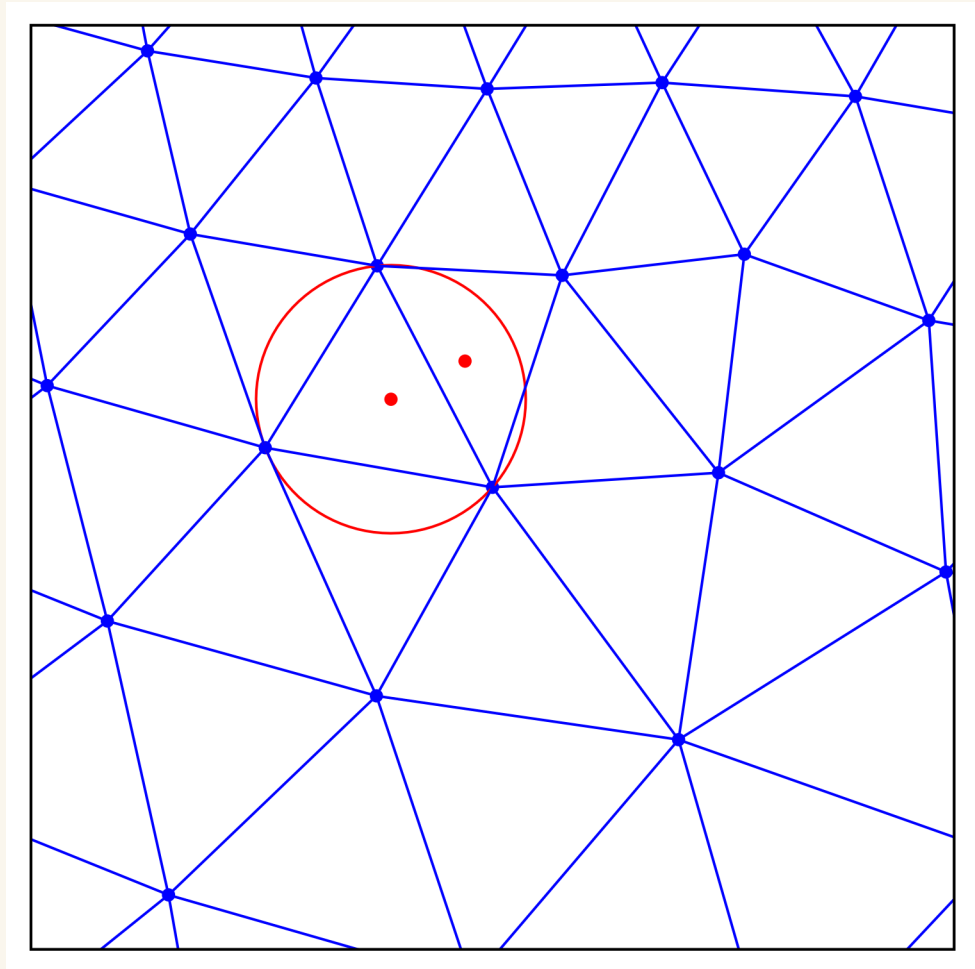
- Find the Delaunay triangulation of the generating points
- Compute the circumcenters of the Delaunay triangles



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

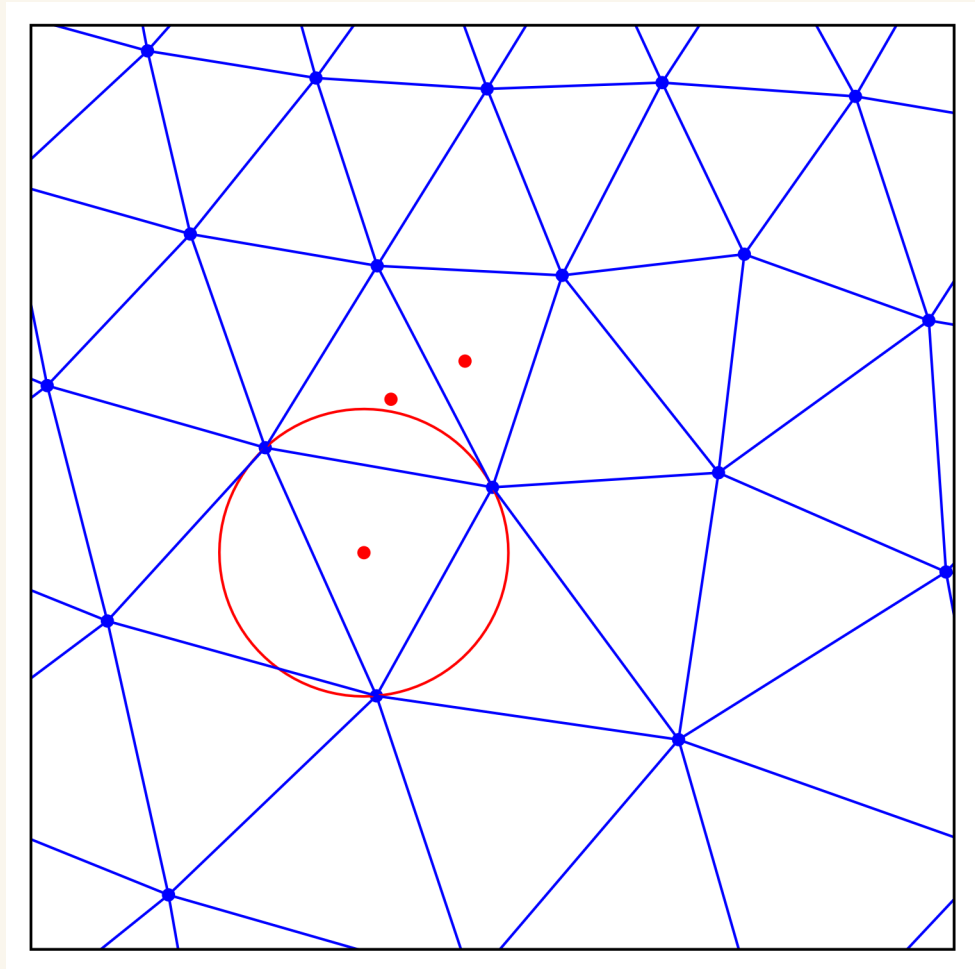
- Find the Delaunay triangulation of the generating points
- Compute the circumcenters of the Delaunay triangles



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

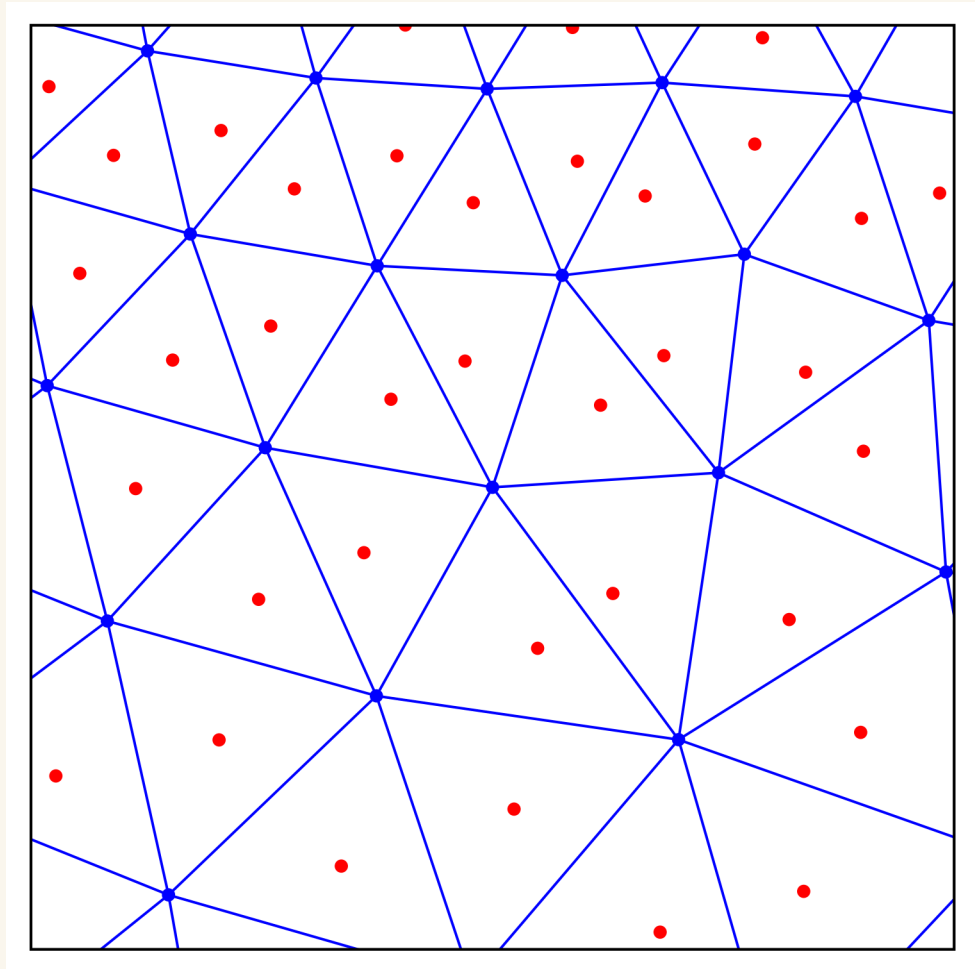
- Find the Delaunay triangulation of the generating points
- Compute the circumcenters of the Delaunay triangles



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

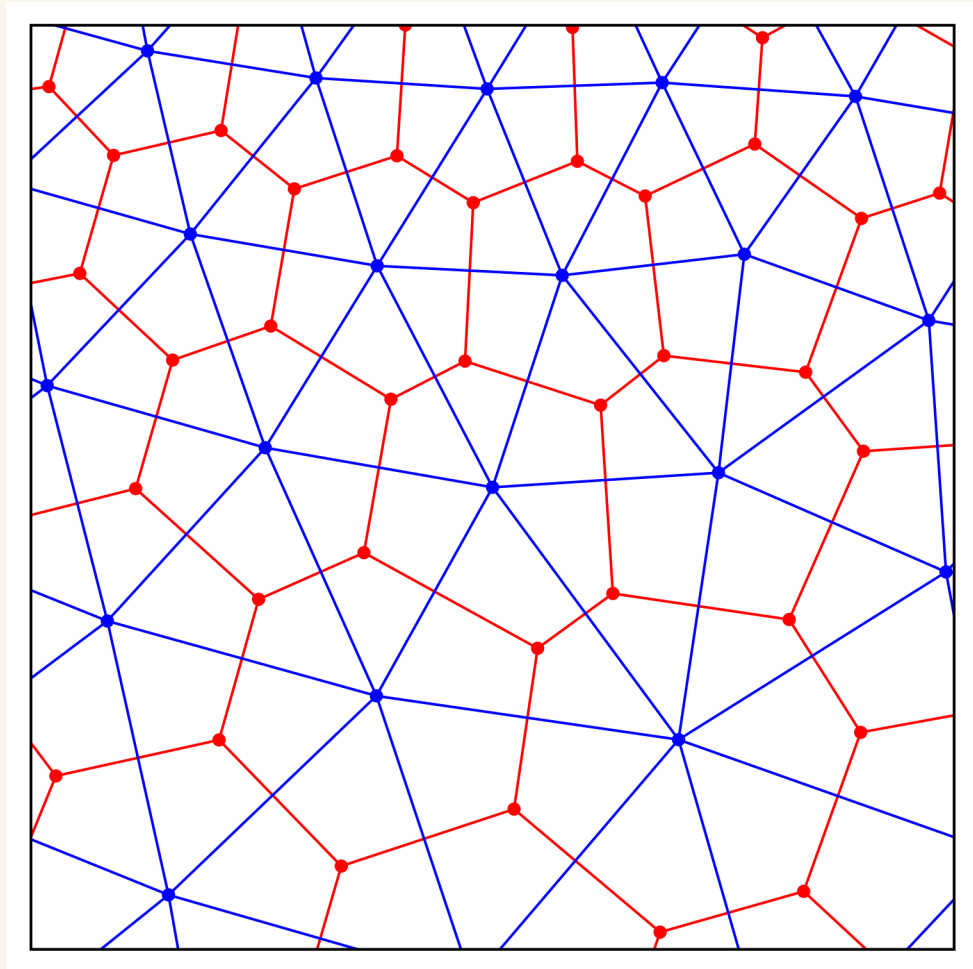
- Find the Delaunay triangulation of the generating points
- Compute the circumcenters of the Delaunay triangles



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

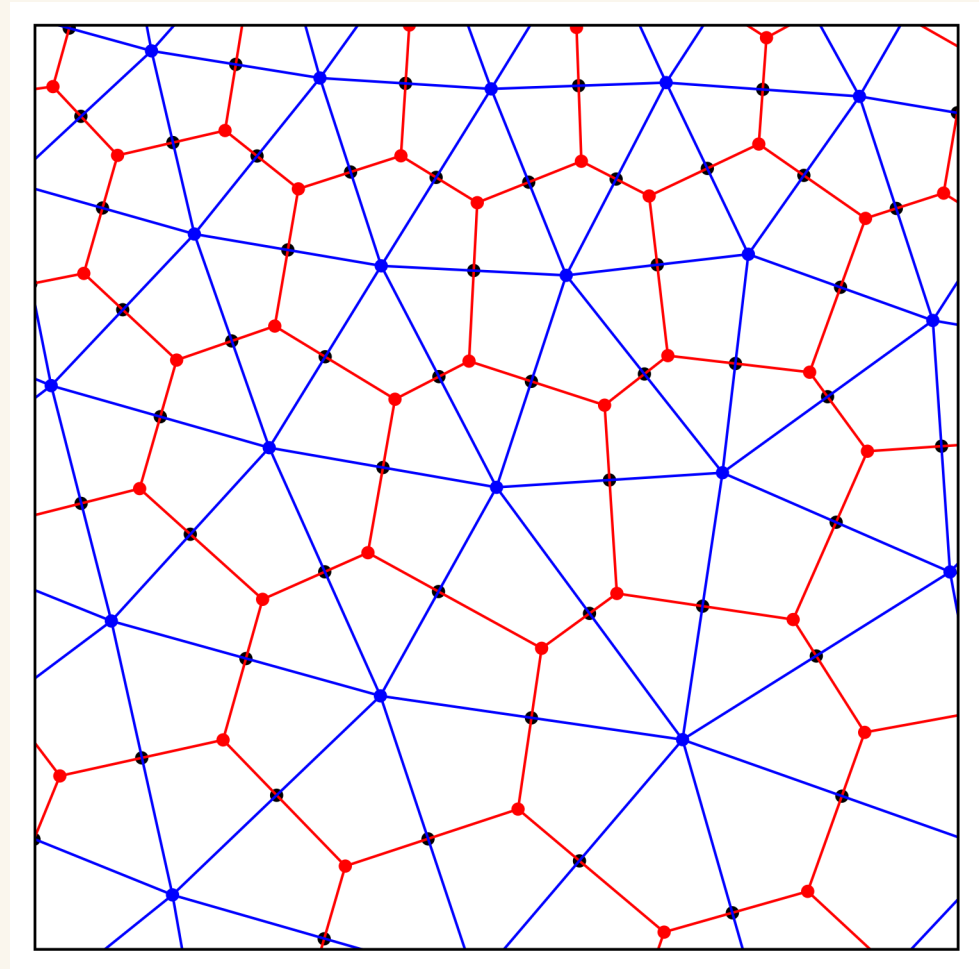
- Find the Delaunay triangulation of the generating points
- Compute the circumcenters of the Delaunay triangles
- The Voronoi regions are defined by these circumcenters



Construction of C-staggered Voronoi tessellations

Given a set of generating points, constructing a C-staggered Voronoi mesh is straightforward

- Find the Delaunay triangulation of the generating points
- Compute the circumcenters of the Delaunay triangles
- The Voronoi regions are defined by these circumcenters
- The normal components of the horizontal velocity in our C-staggered mesh lie at the intersections of Delaunay and Voronoi faces



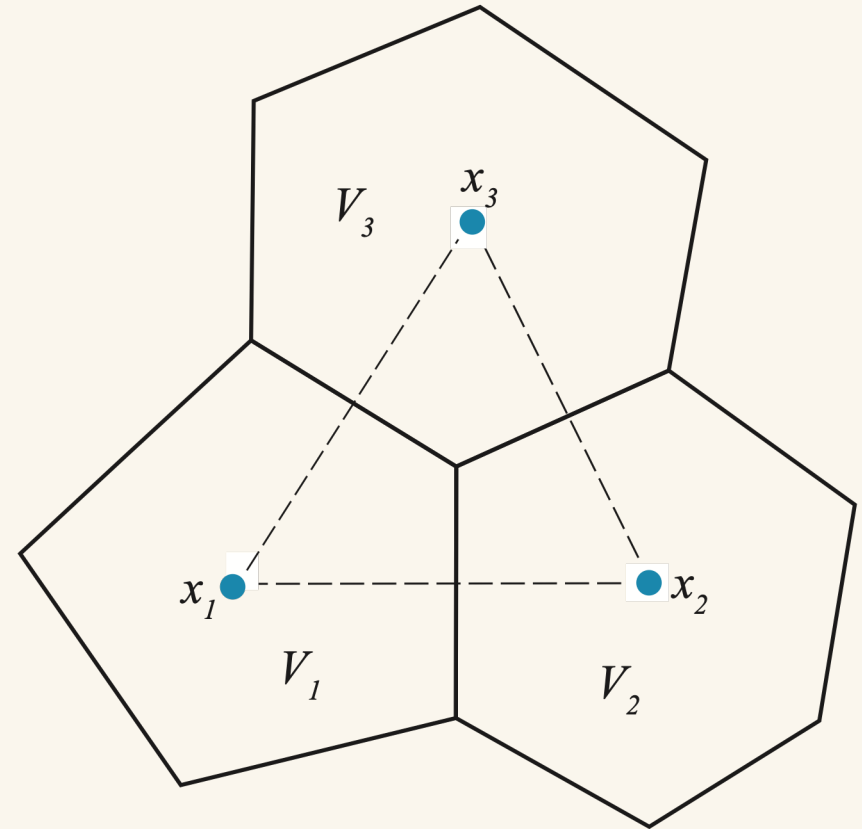
Necessity of the Voronoi property

Voronoi = each grid volume (cell)

V_i is uniquely associated with a *generating point* $\mathbf{x}_i$ such that all points within V_i are closer to $\mathbf{x}_i$ than to any other $\mathbf{x}_j$

Consequently, lines joining generating points of adjacent cells are

1. bisected by the shared cell face; and
2. intersect the shared cell face at a right angle.



Meshes for MPAS-A must be Voronoi tessellations

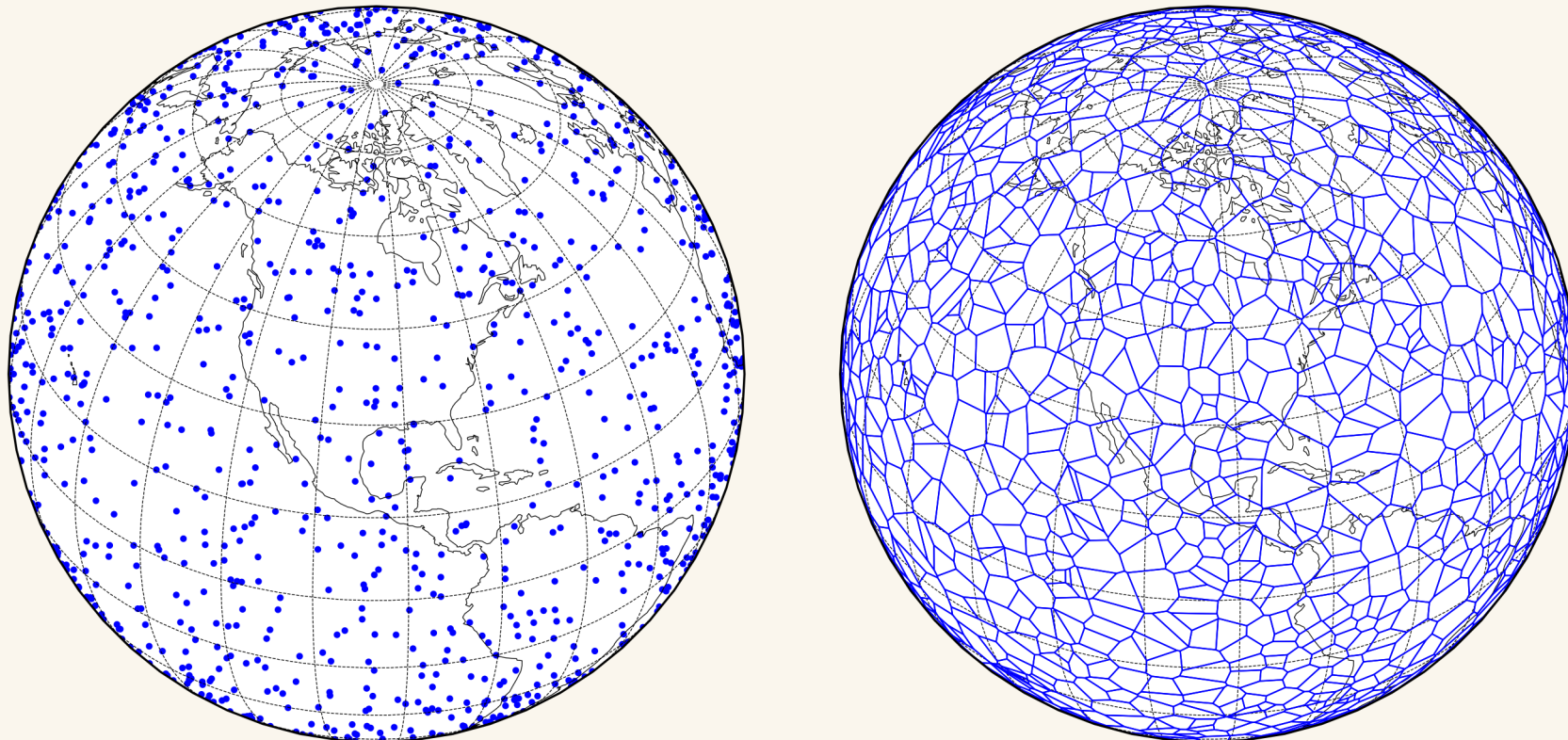
Outline

1. A quick refresher on Voronoi tessellations

2. The goals of mesh generation



The Voronoi property alone is not sufficient



Above: A set of randomly chosen points on the sphere (left) and its Voronoi tessellation (right).

Besides the Voronoi property, there are additional quality metrics to consider for MPAS-A meshes!

The fundamental task of generating a mesh

Fundamentally, our job is to arrive at a “good” placement of generating points (i.e., nominal cell centers)

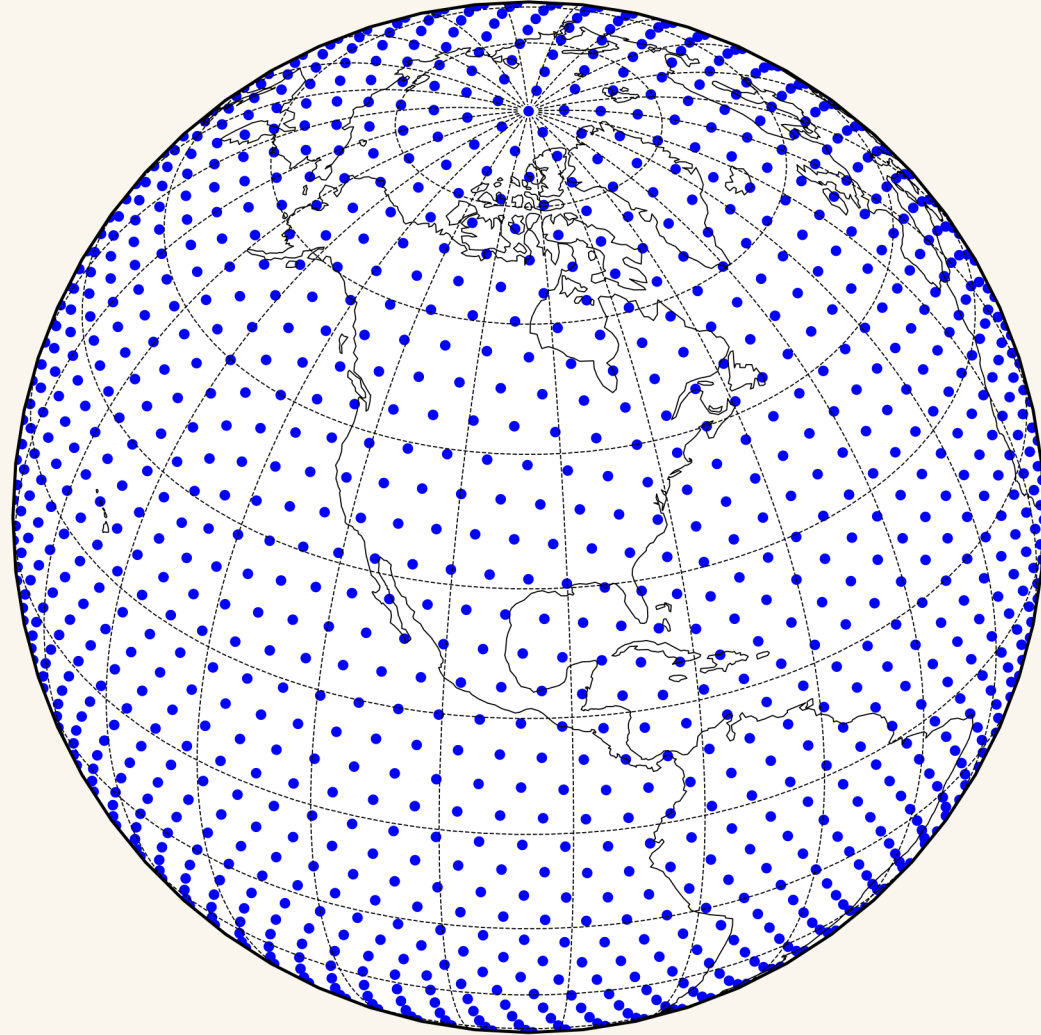
- From those generating points, we can immediately (and efficiently!) construct a Voronoi tessellation



The fundamental task of generating a mesh

Fundamentally, our job is to arrive at a “good” placement of generating points (i.e., nominal cell centers)

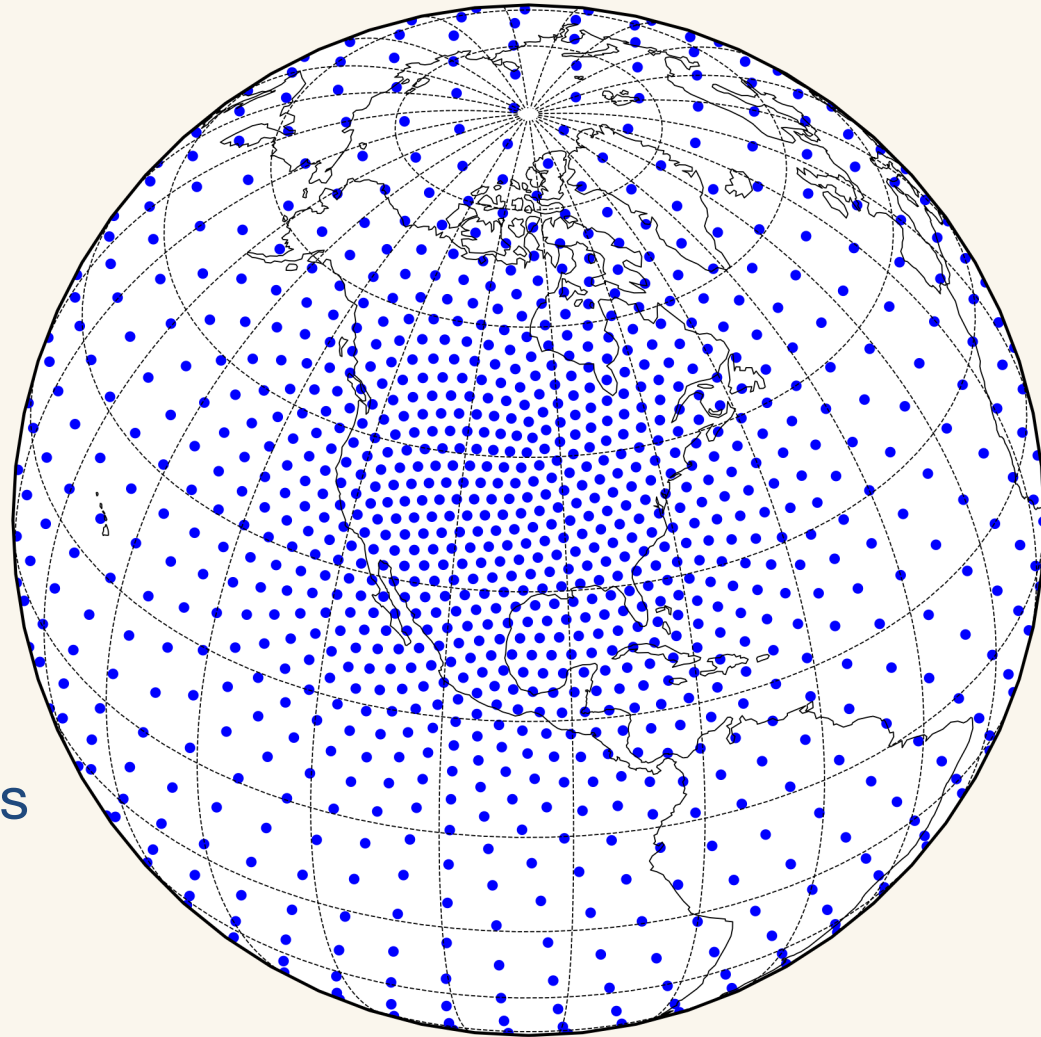
- We could employ a quasi-uniform arrangement based on the icosahedron



The fundamental task of generating a mesh

Fundamentally, our job is to arrive at a “good” placement of generating points (i.e., nominal cell centers)

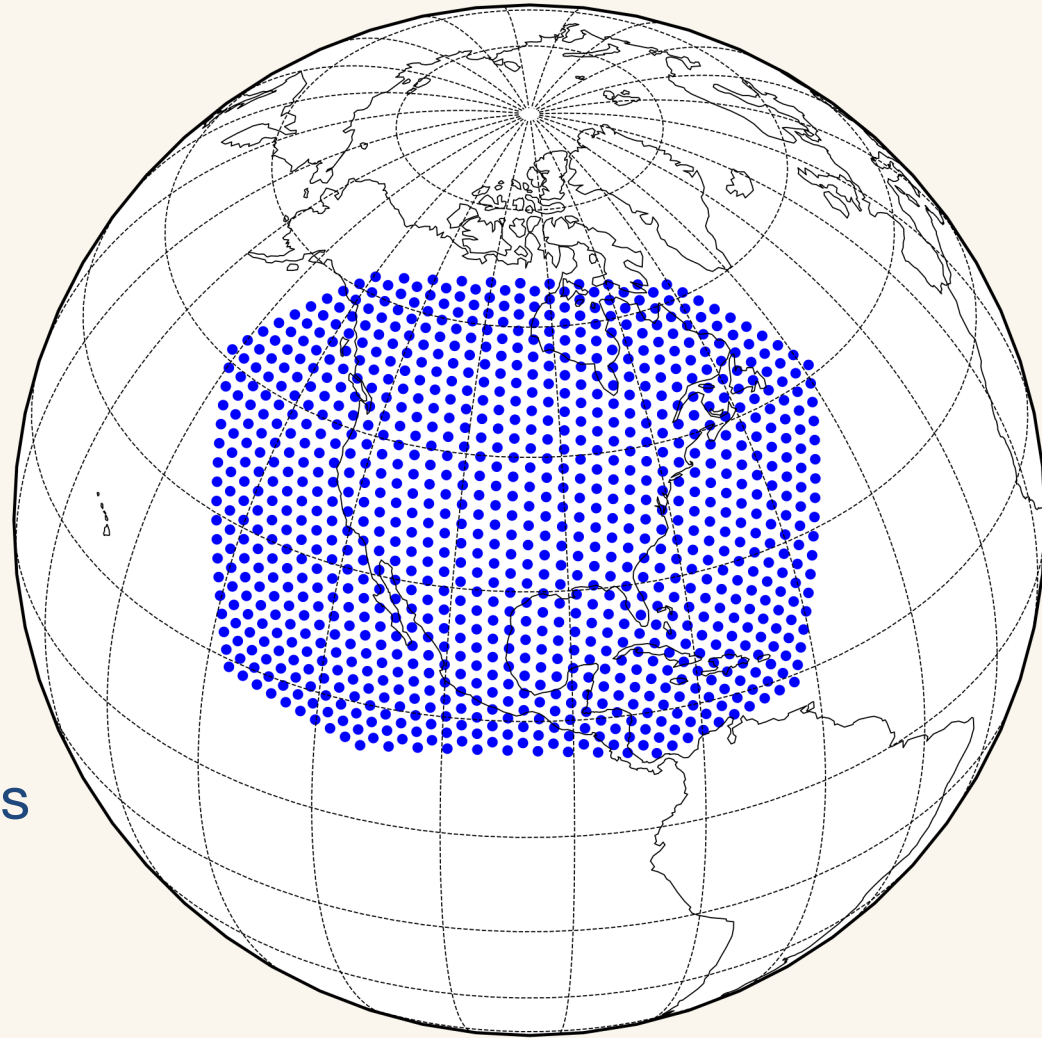
- We could employ a quasi-uniform arrangement based on the icosahedron
- Or a variable-resolution arrangement that minimizes some cost function



The fundamental task of generating a mesh

Fundamentally, our job is to arrive at a “good” placement of generating points (i.e., nominal cell centers)

- We could employ a quasi-uniform arrangement based on the icosahedron
- Or a variable-resolution arrangement that minimizes some cost function
- Or a grid of hexagons laid out on a map projection

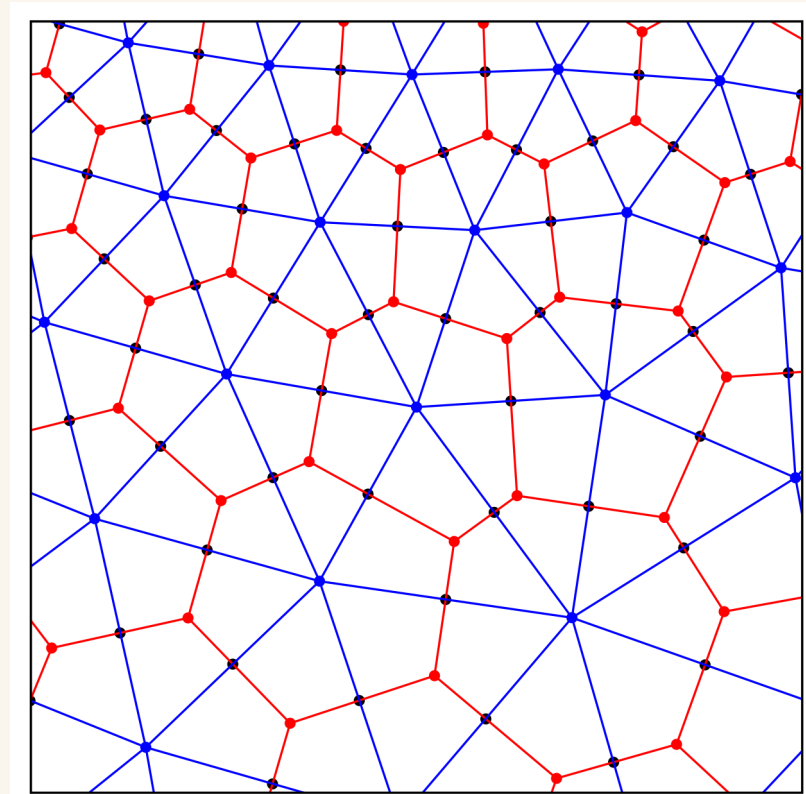


Our end goal is a `grid.nc` netCDF file

Regardless of how we arrange generating points — from which we can produce a Voronoi tessellation — what MPAS-A ultimately needs is a netCDF file with a well-defined set of arrays.

These netCDF `grid.nc` files contain:

- Coordinates of **cells** (blue dots), **vertices** (red dots), and **edges** (black dots)
- Distances, areas, and angles
- Connectivity among neighboring cells, vertices, and edges
- Nominal minimum grid distance
- Mesh density function
- Several other fields



Fields that comprise a `grid.nc` netCDF file

Coordinates

{x,y,z}Cell, {lat,lon}Cell

{x,y,z}Vertex, {lat,lon}Vertex

{x,y,z}Edge, {lat,lon}Edge

Distances

dvEdge, dcEdge

Areas

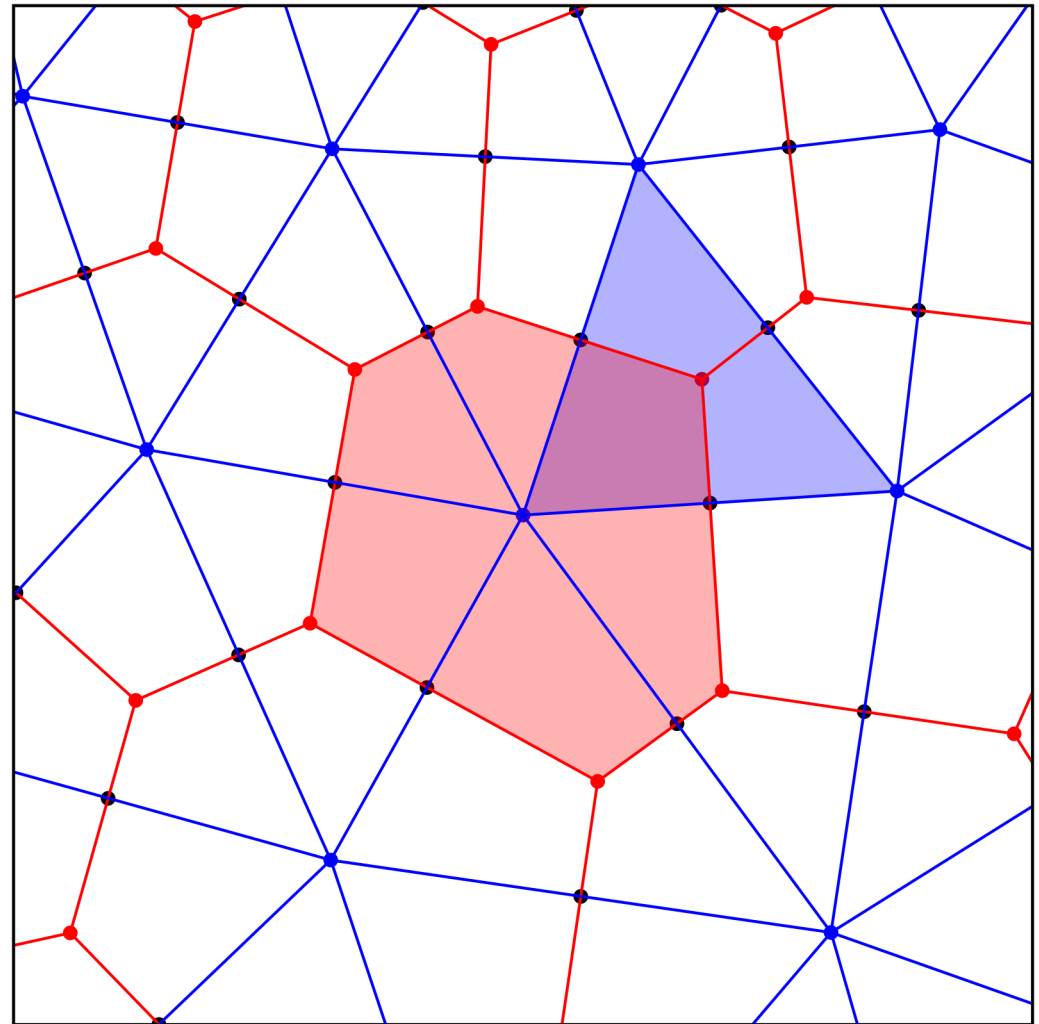
areaCell

areaTriangle

kiteAreasOnVertex

Angles

angleEdge



Fields that comprise a `grid.nc` netCDF file

Cell connectivity

`nEdgesOnCell`

`cellsOnCell`

`edgesOnCell`

`verticesOnCell`

Vertex connectivity

`cellsOnVertex`

`edgesOnVertex`

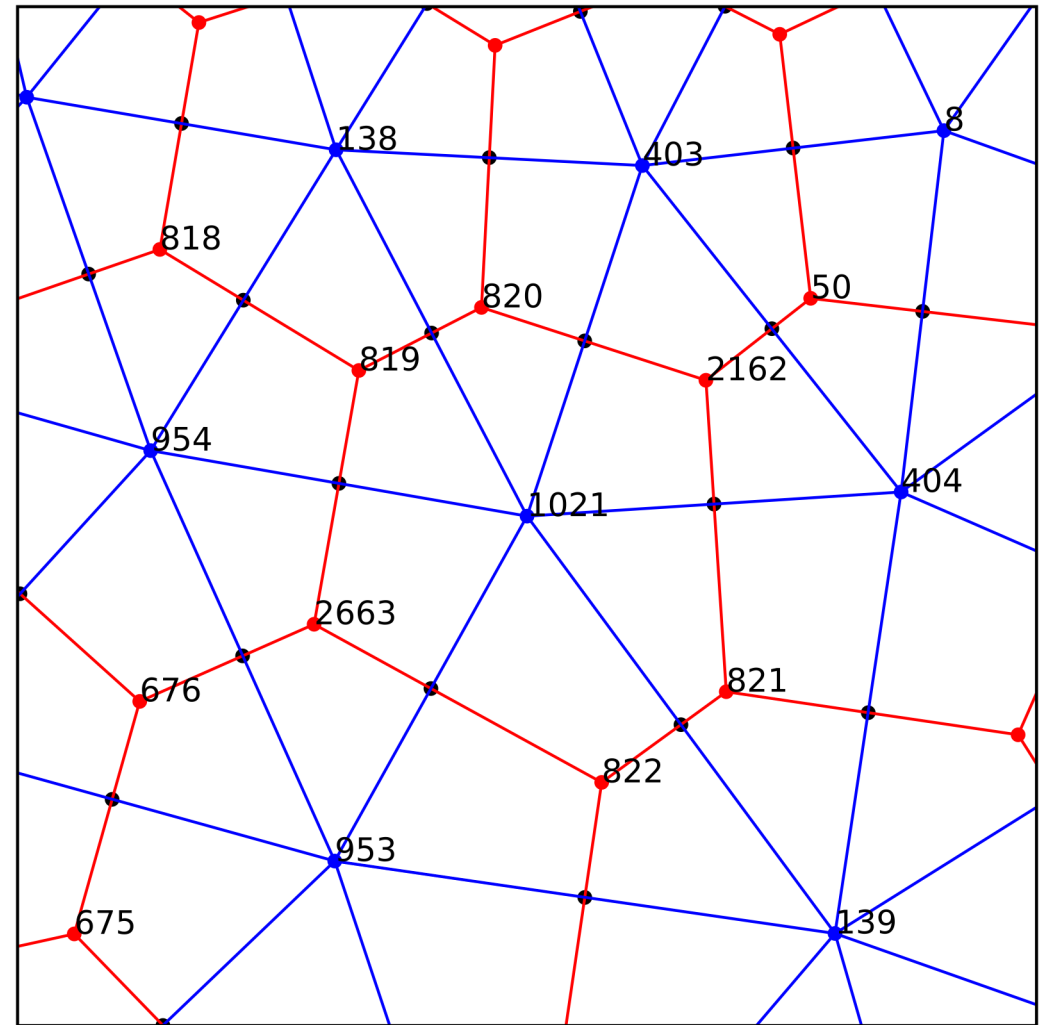
Edge connectivity

`nEdgesOnEdge`

`cellsOnEdge`

`verticesOnEdge`

`edgesOnEdge`



A subsection of a mesh with some cells and vertices labeled.

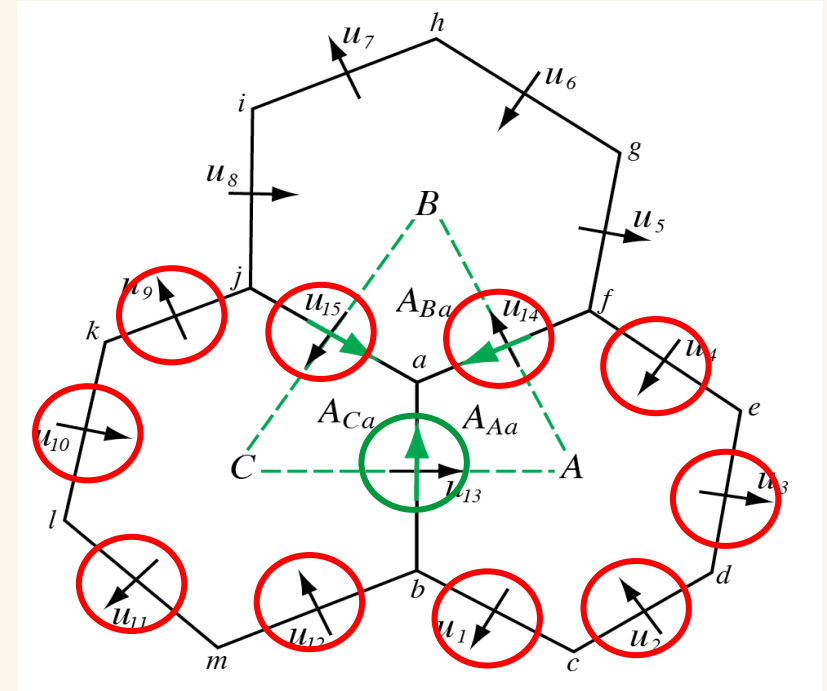
Fields that comprise a `grid.nc` netCDF file

Special fields

`weightsOnEdge`

The **`weightsOnEdge`** field provides, for each edge $\mathbf{e}_i$ in the mesh, the weights for neighboring edge normal velocities when reconstructing the tangential horizontal velocity at $\mathbf{e}_i$.

- The **`nEdgesOnEdge`** field provides the number of edges that participate in the reconstruction for $\mathbf{e}_i$
- The **`edgesOnEdge`** field gives the indices of participating edges



Above: The neighboring edges (red circles) that contribute to the reconstruction of the tangential velocity at an edge (green circle).

Fields that comprise a `grid.nc` netCDF file

Special fields

weightsOnEdge

meshDensity

nominalMinDc

$$\rho(\mathbf{x}) = \left(\frac{h_{fine}}{h(\mathbf{x})} \right)^4 \iff \frac{h_i}{h_j} \approx \left(\frac{\rho(\mathbf{x}_j)}{\rho(\mathbf{x}_i)} \right)^{\frac{1}{4}}$$

The **meshDensity** and **nominalMinDc** fields provide a means of determining the nominal grid length in any region of a mesh

- $\rho(\mathbf{x}_{fine}) = 1.0$
- **meshDensity** sampled at cell generating point locations using the underlying (continuous) mesh distance function

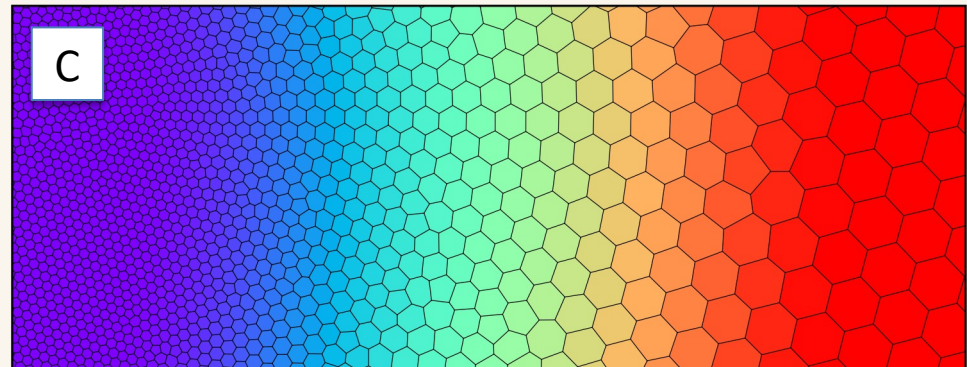
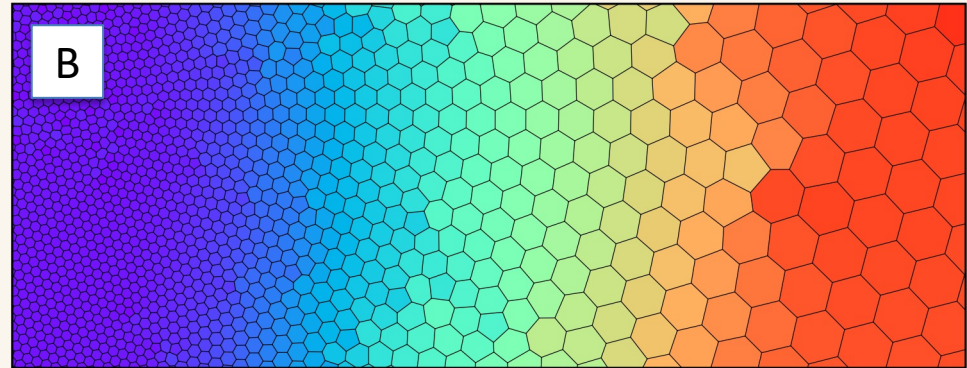
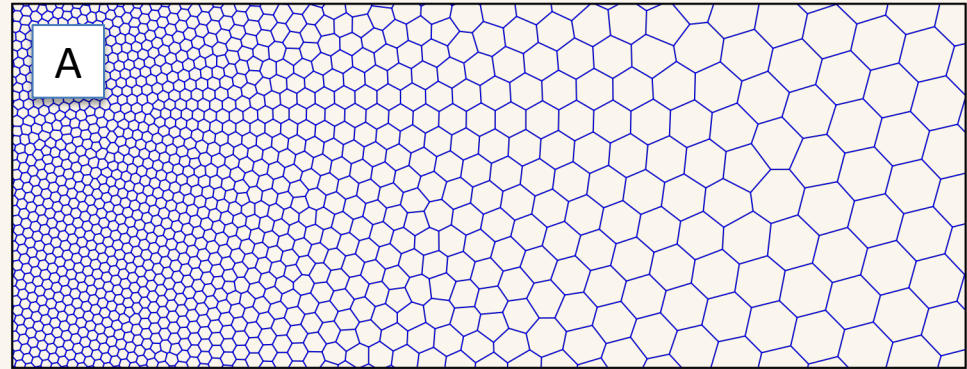
*MPAS's **meshDensity** definition follows from the conjectured relationship between mesh size and density function in centroidal Voronoi tessellations in Ju et al.**

* “Voronoi Diagrams and their Application in Climate and Global Modeling”, Chapter in *Numerical Techniques for Global Atmospheric Models*. Springer-Verlag Berlin Heidelberg, 2011.

Fields that comprise a `grid.nc` netCDF file

The **meshDensity** field is used in the MPAS-A solver when we need a smooth field of nominal cell sizes.

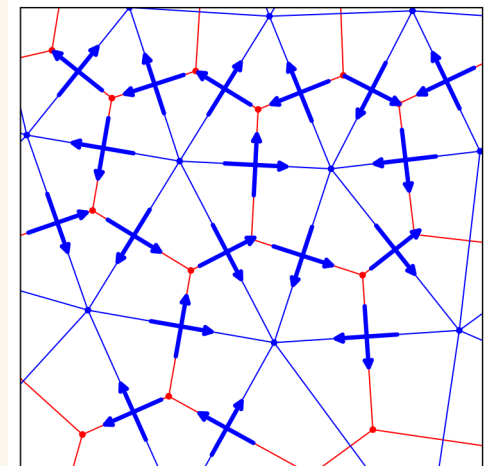
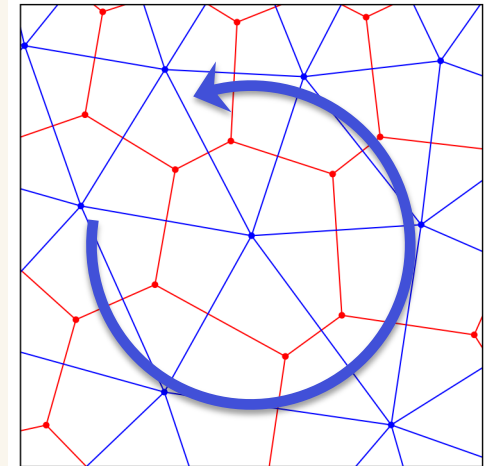
- Consider a cutout from a variable-resolution mesh (**A**). How might we obtain a local mesh size?
- Taking the mean distance between a cell and its neighbors is possible, but leads to a "noisy" cell size field (**B**).
- Employing **nominalMinDc** and **meshDensity** provides a smooth field of *nominal* cell sizes (**C**).



Final notes on `grid.nc` netCDF files

When generating meshes using an alternative workflow, or when working with MPAS-A meshes, there are a few other points worth remembering:

- Neighbors of cells, vertices listed in CCW order
- Connectivity fields are 1-based
`cellsOnCell` *et al.* lie in the range `[1..n]`
- Cross product of normal vector and tangential vector at edges points up
- Meshes are defined on unit sphere



Outline

1. A quick refresher on Voronoi tessellations

2. The goals of mesh generation

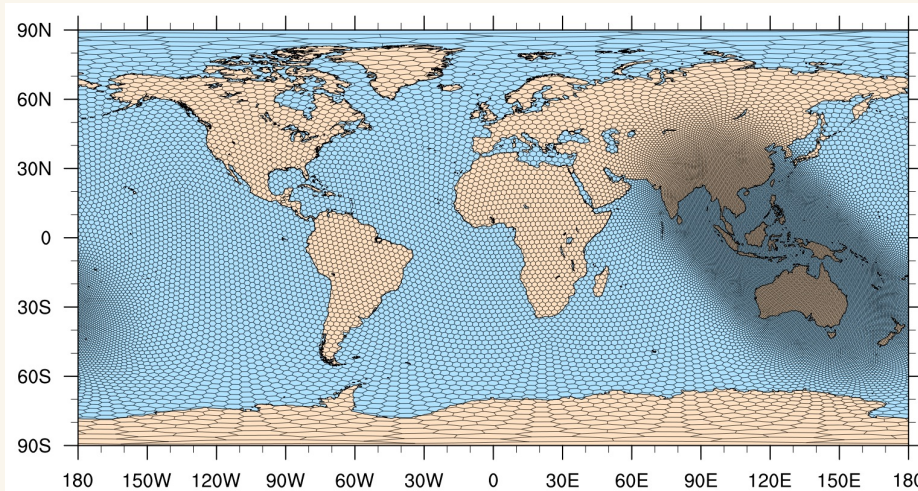
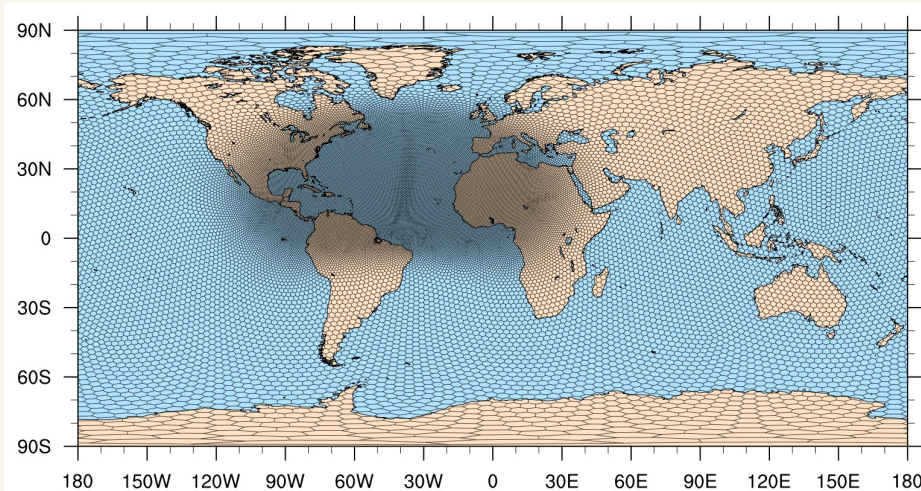
3. Existing mesh tools

Existing tools: `grid_rotate`

The `grid_rotate` tool implements arbitrary rotations of existing meshes (global or regional) about the sphere

Applies two solid-body rotations for spherical meshes:

1. Move center of refined region from one location to another
2. Rotate the relocated refinement about its center to change orientation



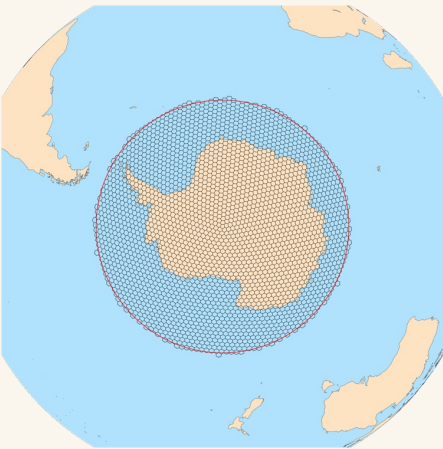
Above: A refinement region originally centered at 25N, 40W has been shifted to 7S, 125E and rotated by -45 degrees.

Clone the <https://github.com/MPAS-Dev/MPAS-Tools.git> repository and look in the `mesh_tools/grid_rotate` directory.

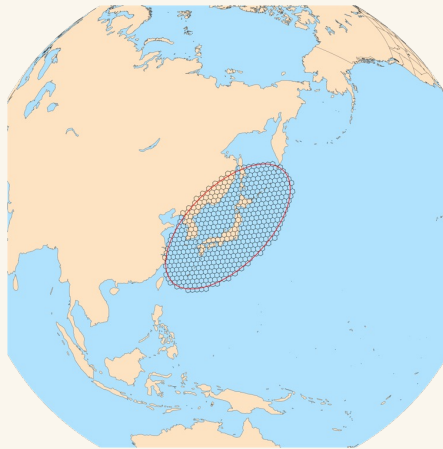
Existing tools: `create_region`

The `create_region` tool subsets any global mesh (uniform or variable-resolution) to create a limited-area mesh

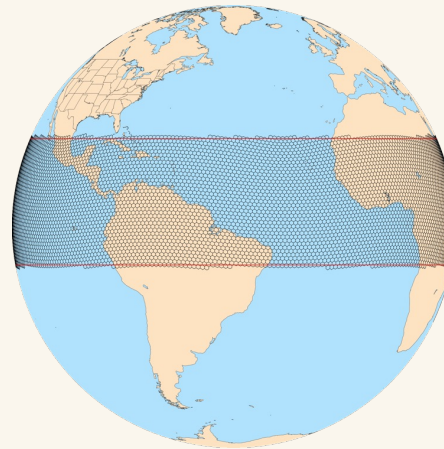
Regions may be described as **circles**, **ellipses**, **latitude bands**, or **polygons**.



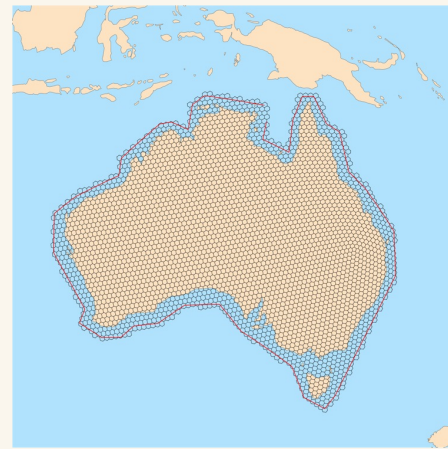
Circular region



Elliptical region



*Latitude band
("channel") region*

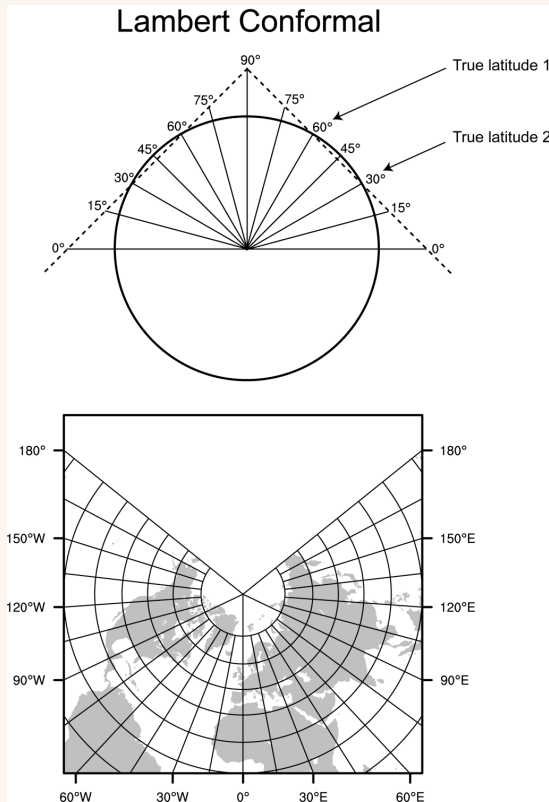


Polygonal region

Clone the <https://github.com/MPAS-Dev/MPAS-Limited-Area.git> repository; example region definitions are in `docs/points-examples`.

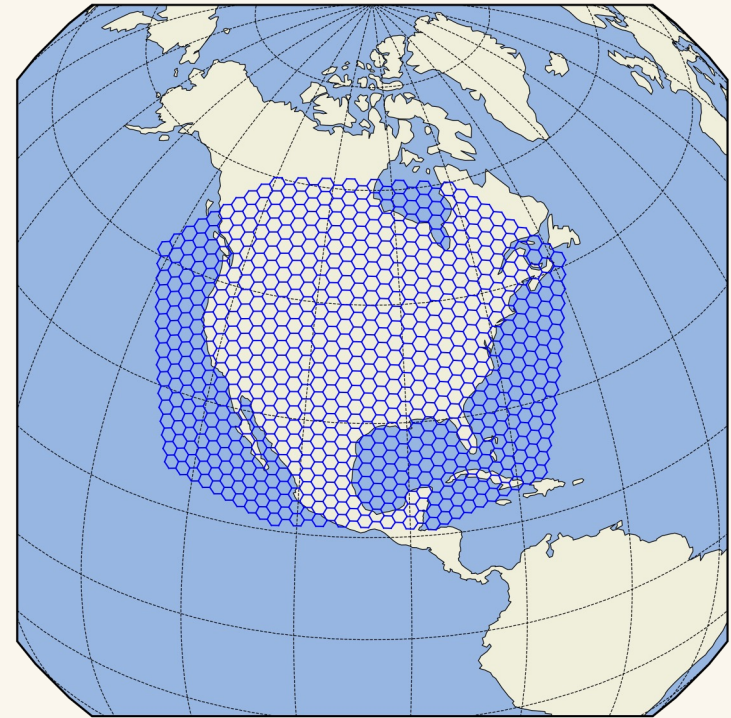
Existing tools: `project_hexes`

The `project_hexes` tool lays out a region of perfect hexagons on a **Lambert conformal conic** map projection plane



After hexagons are arranged on the projection plane, they are transferred back to the surface of the sphere

- Keep regions to mid-latitudes
- Mesh can be subsequently rotated to any other part of the sphere



Clone the <https://github.com/MPAS-Dev/MPAS-Tools.git> repository and look in the `mesh_tools/hex_projection` directory.

Outline

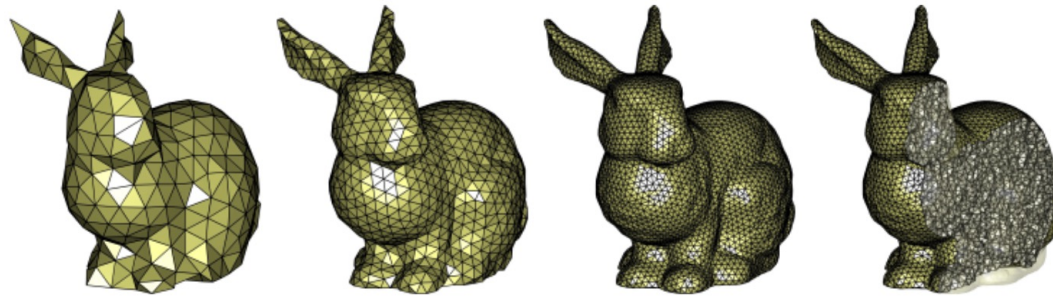
1. A quick refresher on Voronoi tessellations
2. The goals of mesh generation
3. Existing mesh tools
4. A simple JIGSAW-based workflow

Enter JIGSAW

At the core of our workflow is a tool called **JIGSAW**

<https://github.com/dengwirda/jigsaw>

JIGSAW: An unstructured mesh generator



JIGSAW is an unstructured mesh generator and tessellation library; designed to generate high-quality triangulations and polyhedral decompositions of general planar, surface and volumetric domains.

JIGSAW includes refinement-based algorithms for constructing new meshes, optimisation-driven techniques for improving existing grids, as well as routines to assemble (restricted) Delaunay tessellations, Voronoi complexes and Power diagrams.

Note that we're using the "core" JIGSAW software, not JIGSAW(GEO) nor JIGSAW-Python

Prerequisites for our rudimentary workflow

- Install JIGSAW

```
git clone https://github.com/dengwirda/jigsaw.git
cd jigsaw && mkdir build && cd build
cmake .. -DCMAKE_BUILD_TYPE=Release -DCMAKE_INSTALL_PREFIX=...
make -j 4 install
```

- Set up a Python virtual environment

```
python -m venv mpas_mesh
source mpas_mesh/bin/activate
pip install --upgrade pip
pip install numpy netCDF4 matplotlib cartopy
```

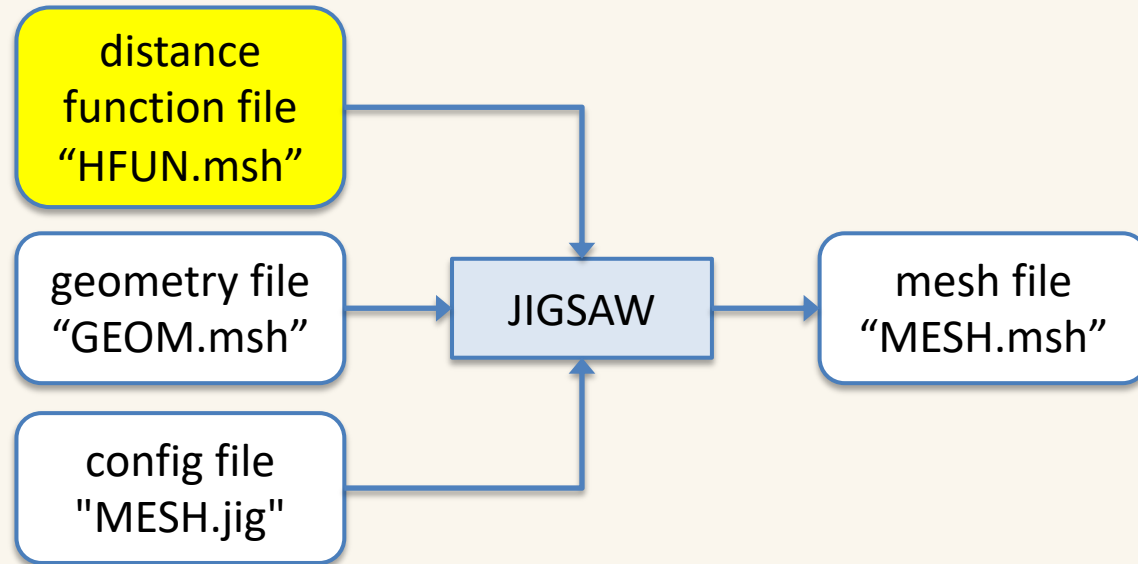
- Install MPI and Parallel-netCDF

(You may already have these installed for MPAS-Atmosphere...)

- Finally, obtain the workflow scripts

```
git clone https://github.com/mgduda/mpas_jigsaw_tutorial.git
```

A rudimentary JIGSAW workflow for MPAS-A



At its simplest, we'll supply just three input files to JIGSAW:

MESH.jig

```
VERBOSITY=1
GEOM_FILE=GEOM.msh
HFUN_FILE=HFUN.msh
HFUN_SCAL=absolute
HFUN_HMAX=inf
HFUN_HMIN=0.0
MESH_FILE=MESH.msh
MESH_DIMS=2
OPTM_QLIM=0.9375
```

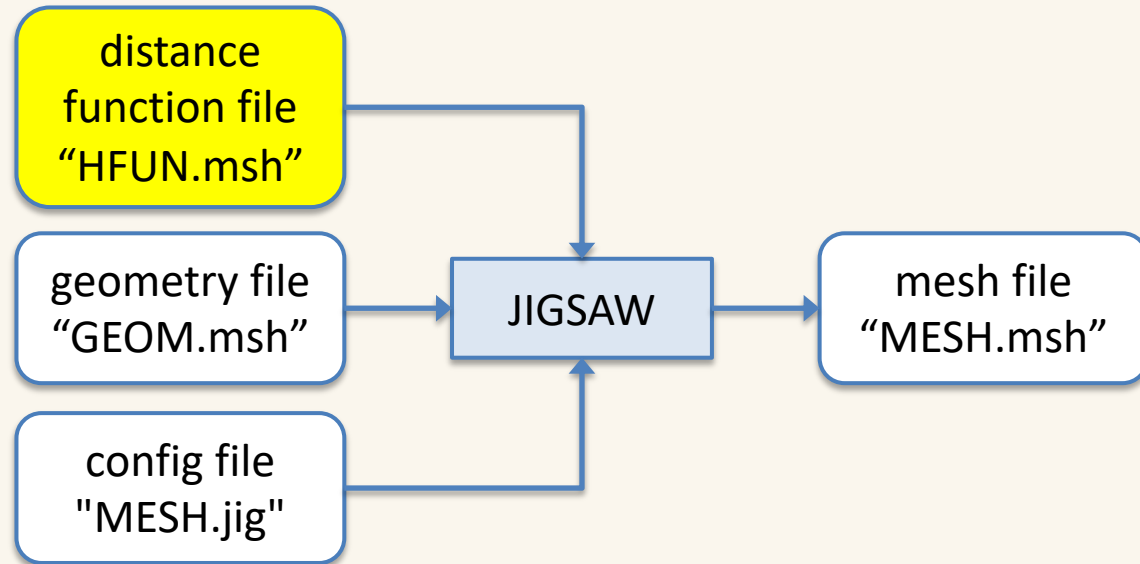
GEOM.msh

```
MSHID=3;ellipsoid-mesh
RADII=6371.229;6371.229;6371.229
```

HFUN.msh

A 2-d grid of mesh distance function values in a text-based format. More details to follow...

A rudimentary JIGSAW workflow for MPAS-A



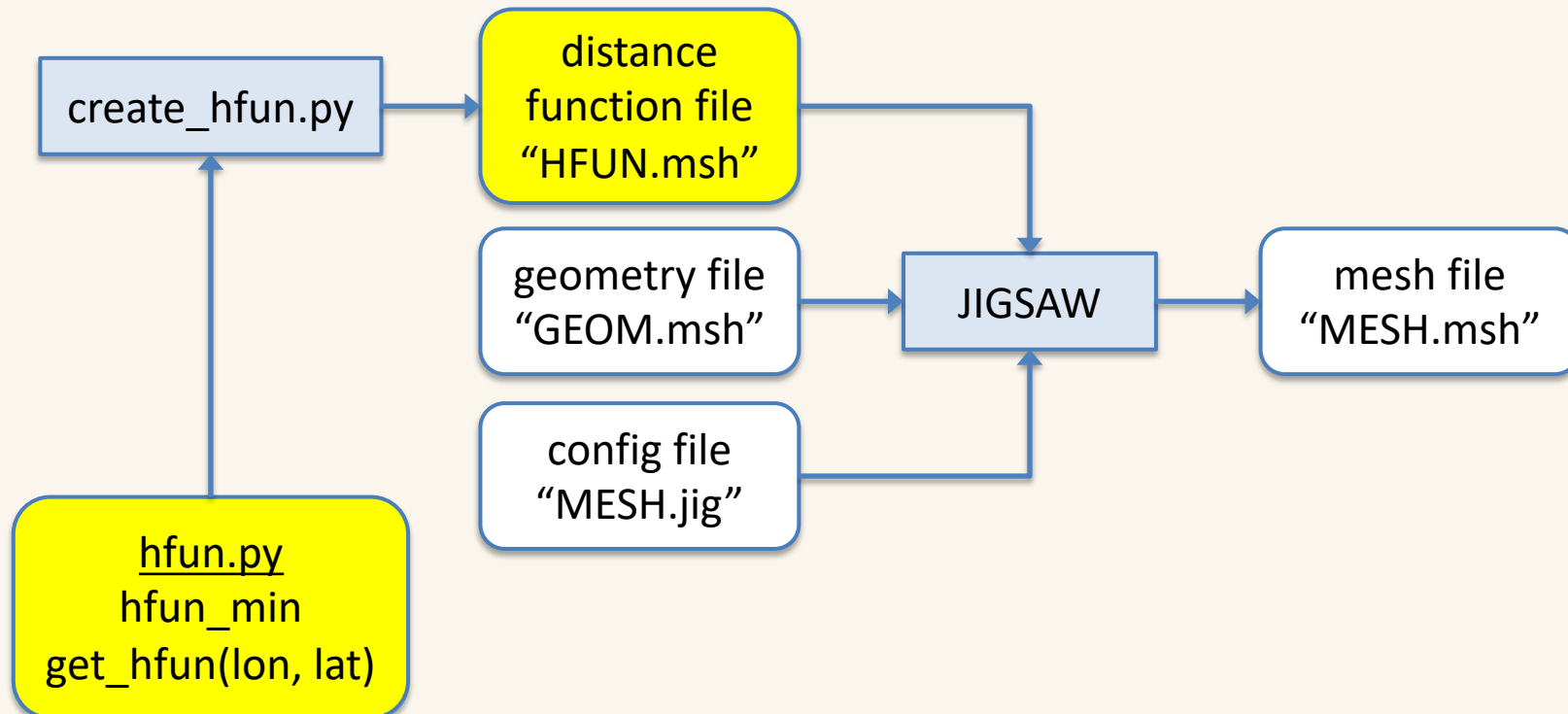
The output file `MESH.msh` is a text-based file:

```
MSHID=3;EUCLIDEAN-MESH
NDIMS=3
POINT=361533
1260.4254956155548;-5726.4101862545276;2492.411065455231;0
1196.596776851369;-5724.5940429348921;2527.7931417867767;0
...
TRIA3=723062
106;21;0;0
115;106;0;0
...
```

Generating point
coordinates

Indices of
triangle vertices

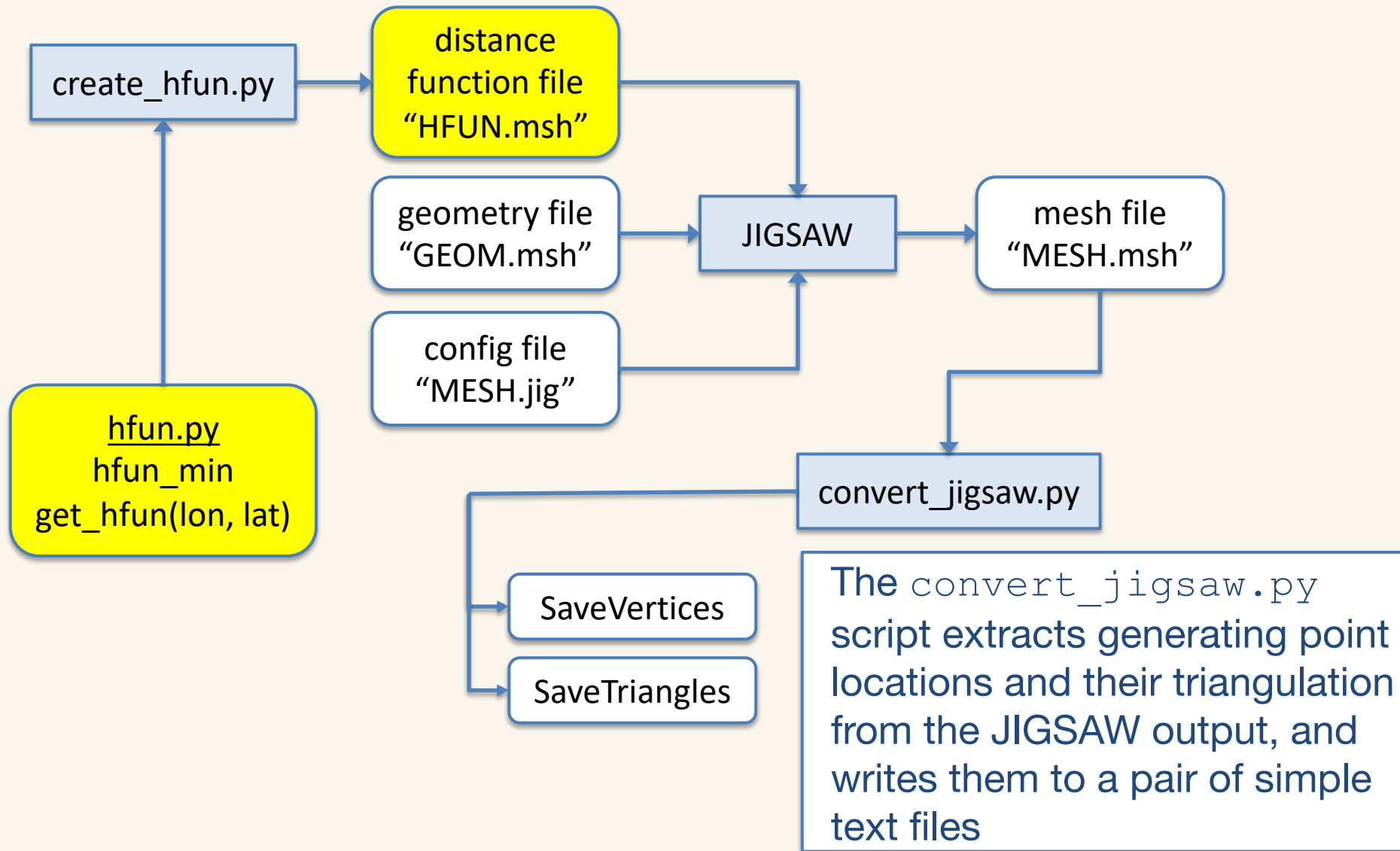
A rudimentary JIGSAW workflow for MPAS-A



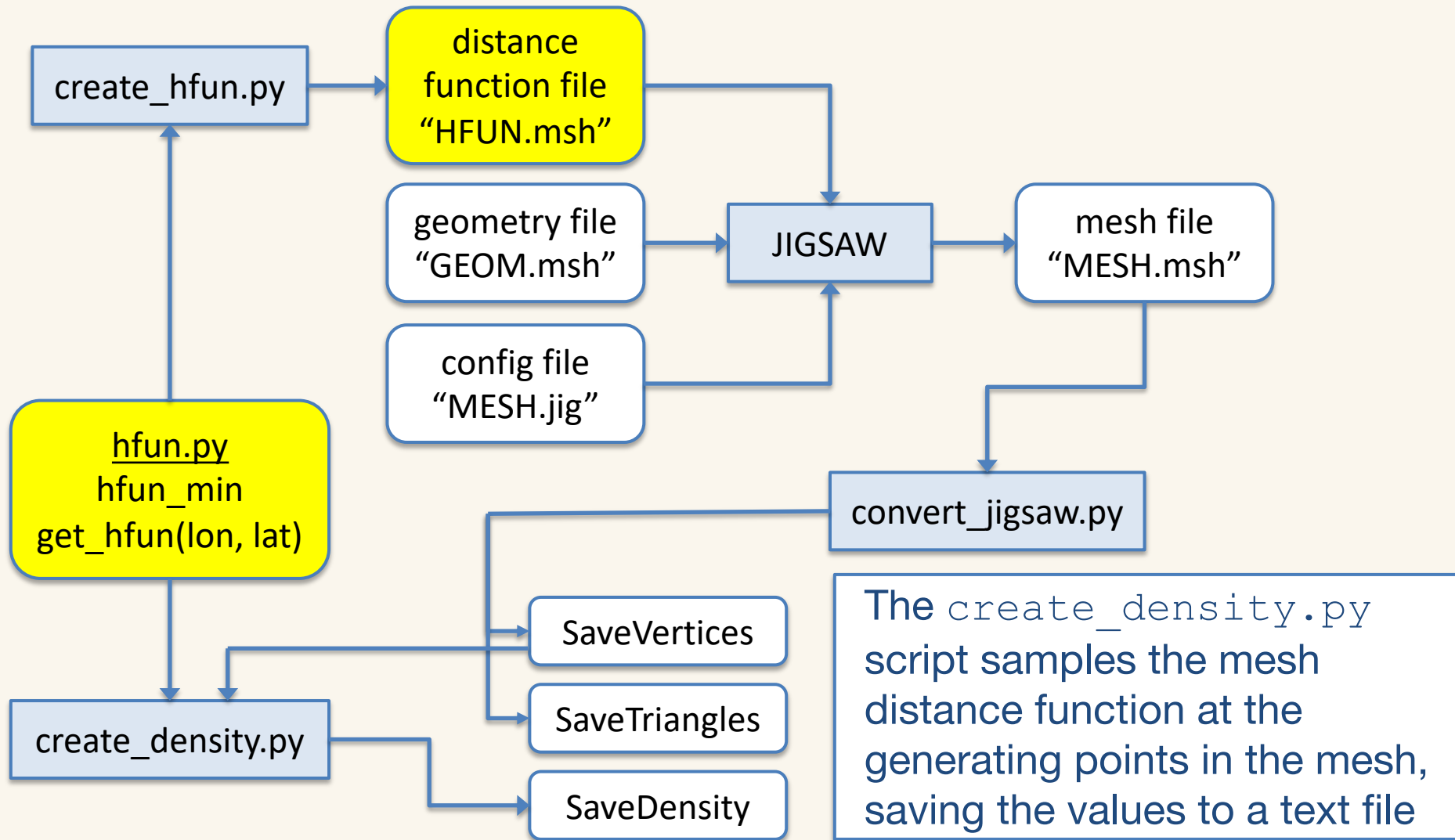
The `hfun.py` Python file defines the mesh distance at any given geographic location

- Writing an `HFUN.msh` file by hand is impractical, so we would create this file with code anyway
- We will also need to sample the distance function at mesh generating points later...

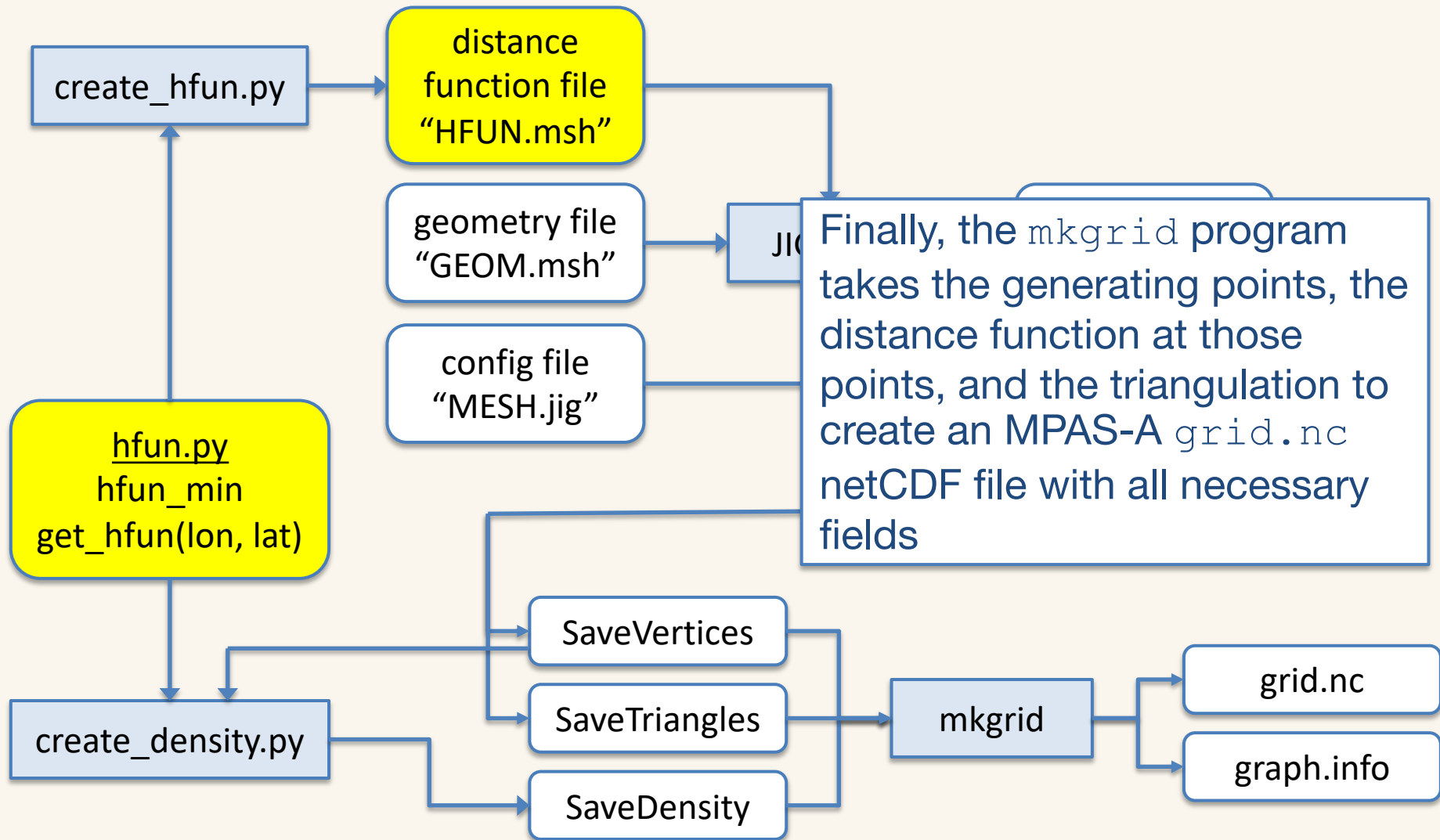
A rudimentary JIGSAW workflow for MPAS-A



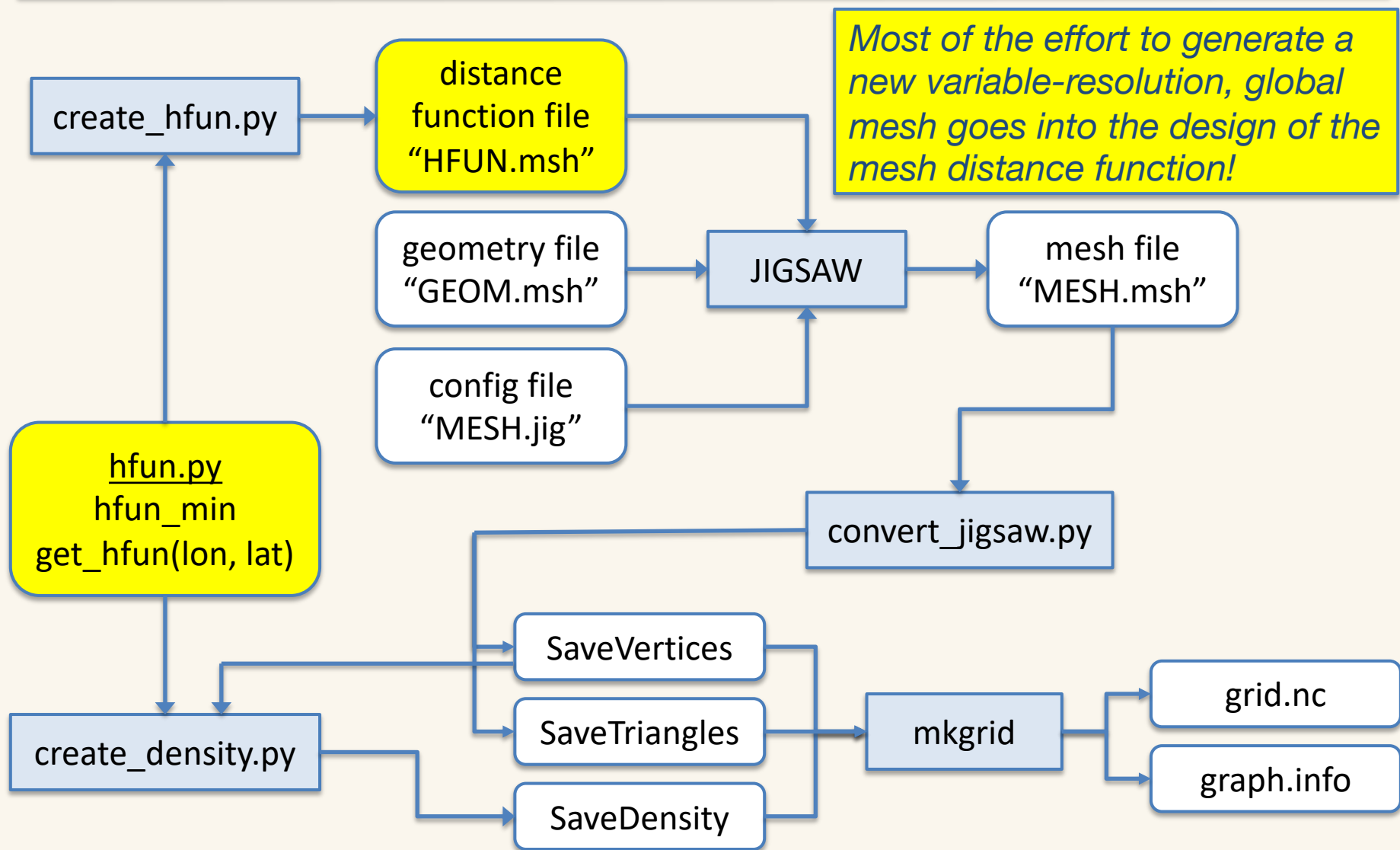
A rudimentary JIGSAW workflow for MPAS-A



A rudimentary JIGSAW workflow for MPAS-A



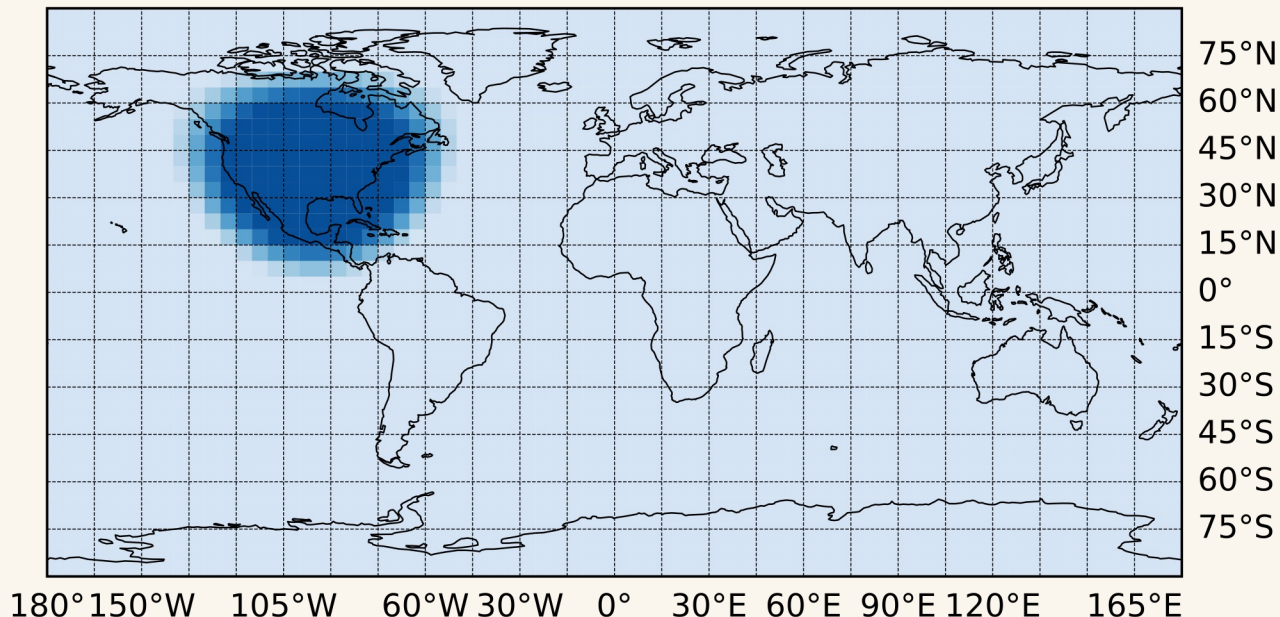
A rudimentary JIGSAW workflow for MPAS-A



JIGSAW “HFUN” files

The `HFUN.msh` distance function file describes the mesh size over the surface of the sphere

- Text format
- List of longitude coordinates (radians)
- List of latitude coordinates (radians)
- 2-d array of mesh sizes at (lat, lon) coordinates



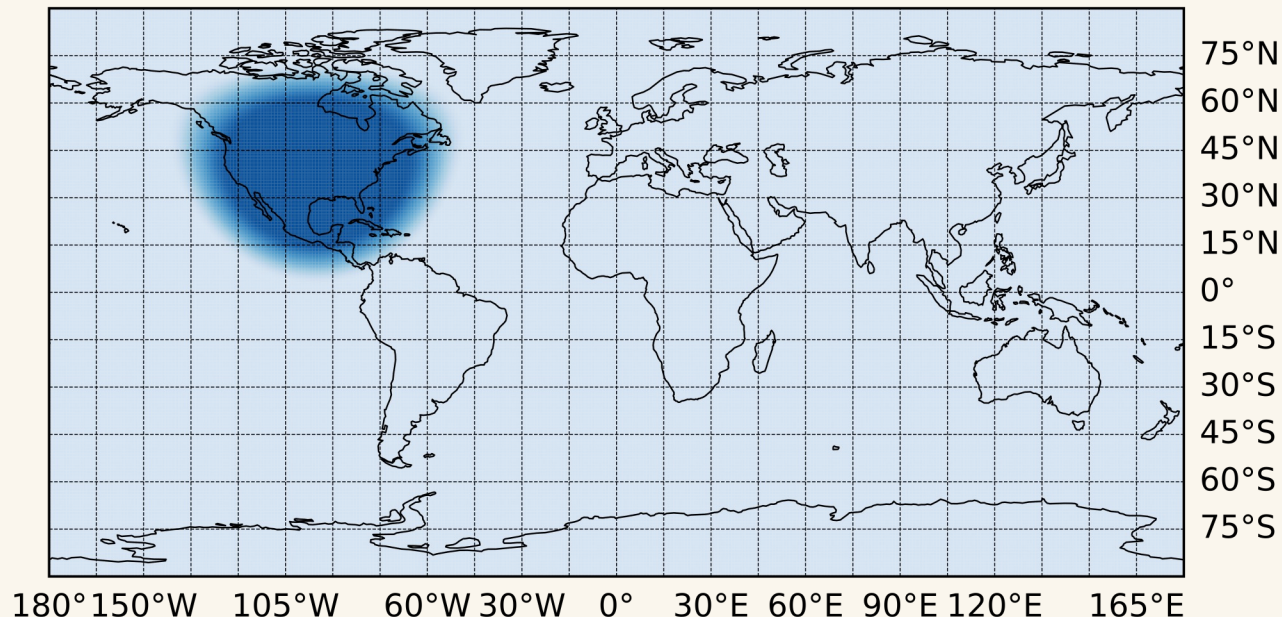
Left: An HFUN file describing refinement over part of North America on a 5°x5° grid

Important: Ensure your distance function is adequately resolved by the lat-lon grid!

JIGSAW “HFUN” files

The `HFUN.msh` distance function file describes the mesh size over the surface of the sphere

- Text format
- List of longitude coordinates (radians)
- List of latitude coordinates (radians)
- 2-d array of mesh sizes at (lat, lon) coordinates



Left: An HFUN file describing refinement over part of North America on a $0.5^\circ \times 0.5^\circ$ grid

The resolution of the lat-lon grid is set in the `create_hfun.py` script.

The `hfun.py` Python file

The `hfun.py` file must define the following (in addition to whatever additional functions are needed to implement these):

- `hfun_min` – a global variable specifying the minimum grid distance
- `get_hfun(lon, lat)` – a function returning an array of mesh distances at locations given by the longitude and latitude input arrays (in radians)

The `get_hfun` is only called once with arrays of longitudes and latitudes, and so it can include expensive setup code

```
hfun_min = 15.0

def get_hfun(lon, lat):
    import numpy as np
    return np.full_like(lon, hfun_min)
```

Above: The contents of a very simple `hfun.py` file describing a uniform 15-km mesh.

Outline

1. A quick refresher on Voronoi tessellations
2. The goals of mesh generation
3. Existing mesh tools
4. A simple JIGSAW-based workflow
5. Generating variable-resolution meshes

A simple example: circular refinement



- Refined region with $\Delta x = 12$ km centered at 96.0 W, 38.5 N, and 2600 km radius
- Transition to $\Delta x = 60$ km over a distance of 960 km
- What we ultimately must provide is a function
`get_hfun(lon, lat)`

A simple example: circular refinement



- Refined region with $dx = 12$ km centered at 96.0 W, 38.5 N, and 2600 km radius
- Transition to $dx = 60$ km over a distance of 960 km
- What we ultimately must provide is a function
`get_hfun(lon, lat)`

In this case, we can take advantage of symmetry and focus on a **1-d function of radial distance from center of refinement**

A simple example: circular refinement

We aim to provide a definition for the function `get_hfun( $\lambda, \phi$ )`

It may be easier to work in Cartesian coordinates on the surface of the unit sphere,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \cos(\lambda) \cdot \cos(\phi) \\ \sin(\lambda) \cdot \cos(\phi) \\ \sin(\phi) \end{bmatrix}$$

in which case, the great-circle distance between any two points is easily computed,

$$r = R_{earth} \cdot \arccos(\hat{\mathbf{p}}_1 \cdot \hat{\mathbf{p}}_2)$$

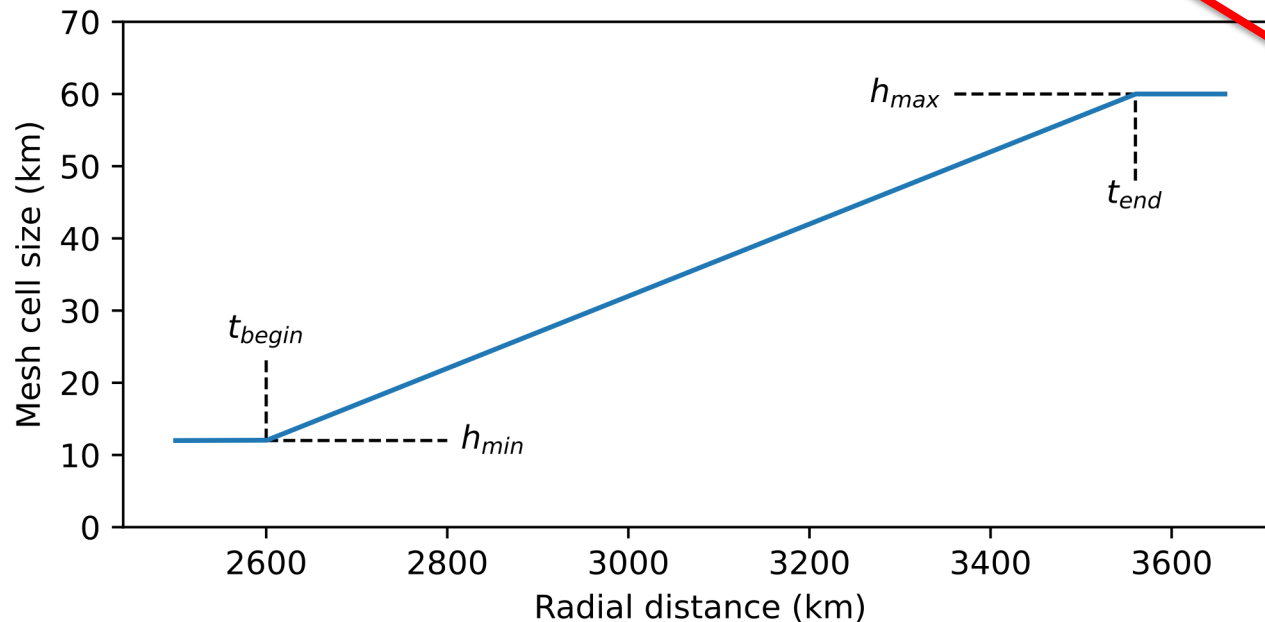
and our `get_hfun` evaluates a scalar function of the form

$$h(r) = \begin{cases} h_{min}, & \text{for } t_{begin} \geq r \\ h_{min} + (r - t_{begin}) \cdot \frac{h_{max} - h_{min}}{t_{end} - t_{begin}}, & \text{for } t_{begin} < r < t_{end} \\ h_{max}, & \text{for } r \geq t_{end} \end{cases}$$

A simple example: circular refinement

Four parameters define the 1-d mesh distance function:

$$h(r) = \begin{cases} h_{min}, & \text{for } t_{begin} \geq r \\ h_{min} + (r - t_{begin}) \cdot \frac{h_{max} - h_{min}}{t_{end} - t_{begin}}, & \text{for } t_{begin} < r < t_{end} \\ h_{max}, & \text{for } r \geq t_{end} \end{cases}$$



Limit the slope of the transition to a few percent! Experience suggests a value of 0.03 is generally safe...

A simple example: circular refinement

Within a file named `hfun.py`, we can add the following code:

```
import numpy as np

# Beginning and ending radii (km) of transition region
t_begin = 2600.0
t_end = 2600.0 + 960.0

# Grid size (km) in the fine and coarse regions of the mesh
hfun_min = 12.0
hfun_max = 60.0

def geo_to_cartesian(lon, lat):
    """ Convert geographic coordinates (radians) to
    Cartesian coordinates (m) on the unit sphere
    """
    x = np.cos(lon) * np.cos(lat)
    y = np.sin(lon) * np.cos(lat)
    z = np.sin(lat)
    return (x, y, z)
```

A simple example: circular refinement

Within a file named `hfun.py`, we can add the following code:

```
def h(r):  
    """ Returns mesh distance (km) as a function of radial  
    distance (km) from refinement center  
    """  
    hires_mask = r < t_begin  
    transition_mask = np.logical_and(r >= t_begin, r < t_end)  
    lowres_mask = r >= t_end  
  
    ret = np.zeros_like(r)  
    ret[hires_mask] = hfun_min  
    ret[transition_mask] = (hfun_min  
                           + (r[transition_mask] - t_begin)  
                           * (hfun_max - hfun_min)  
                           / (t_end - t_begin))  
    ret[lowres_mask] = hfun_max  
  
    return ret
```

A simple example: circular refinement

Within a file named `hfun.py`, we can add the following code:

```
def get_hfun(lon, lat):  
    """ Returns array of mesh distances at locations  
    given by coordinate arrays lon and lat (radians)  
    """  
    lat_c, lon_c = 38.5, -96.0  # Center of refinement  
    r_earth = 6371.229  
  
    x_c, y_c, z_c = geo_to_cartesian(np.radians(lon_c),  
                                     np.radians(lat_c))  
    p_center = np.array([x_c, y_c, z_c])  
  
    x, y, z = geo_to_cartesian(lon.flatten(),  
                               lat.flatten())  
    p = np.column_stack([x, y, z])  
  
    r = r_earth * np.arccos(p @ p_center)  
    return h(r)
```

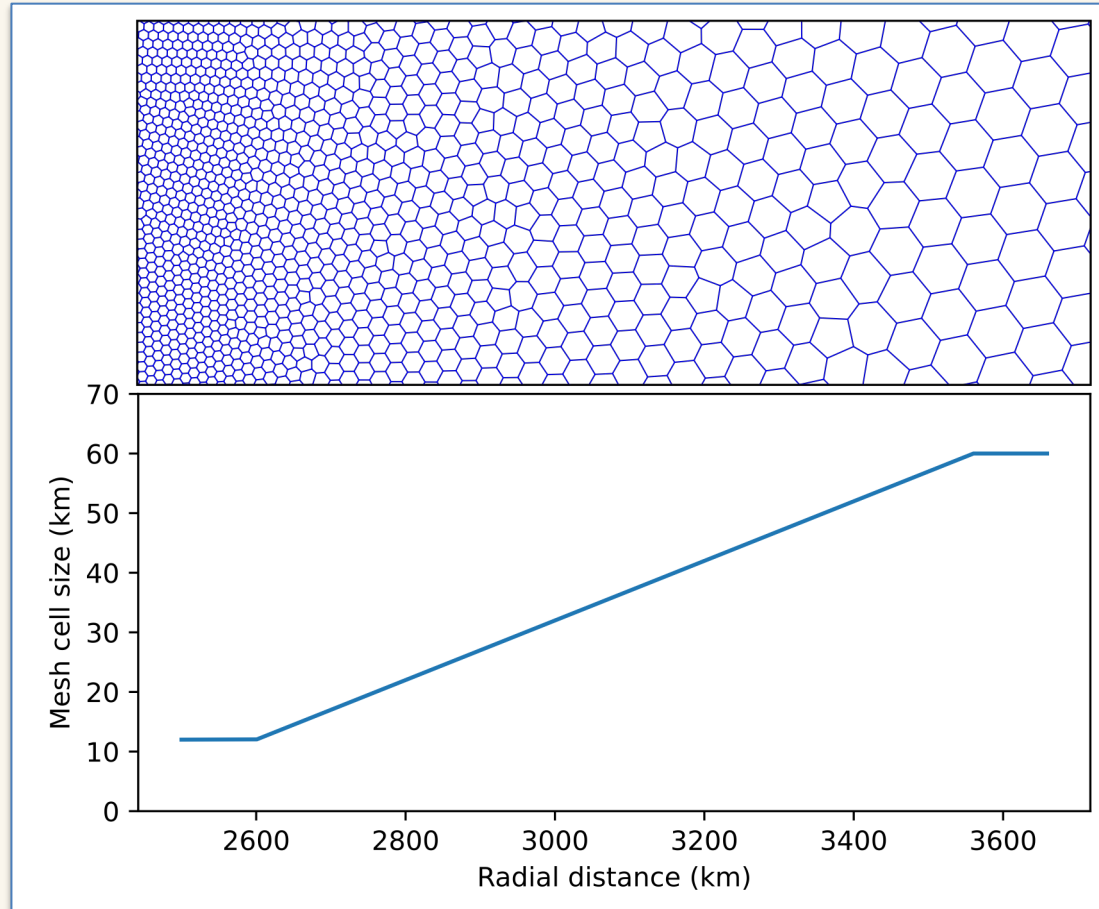
A simple example: circular refinement

Having written the `hfun.py` file, running through the workflow **takes < 5 minutes**

- Resulting mesh has ~346,000 cells

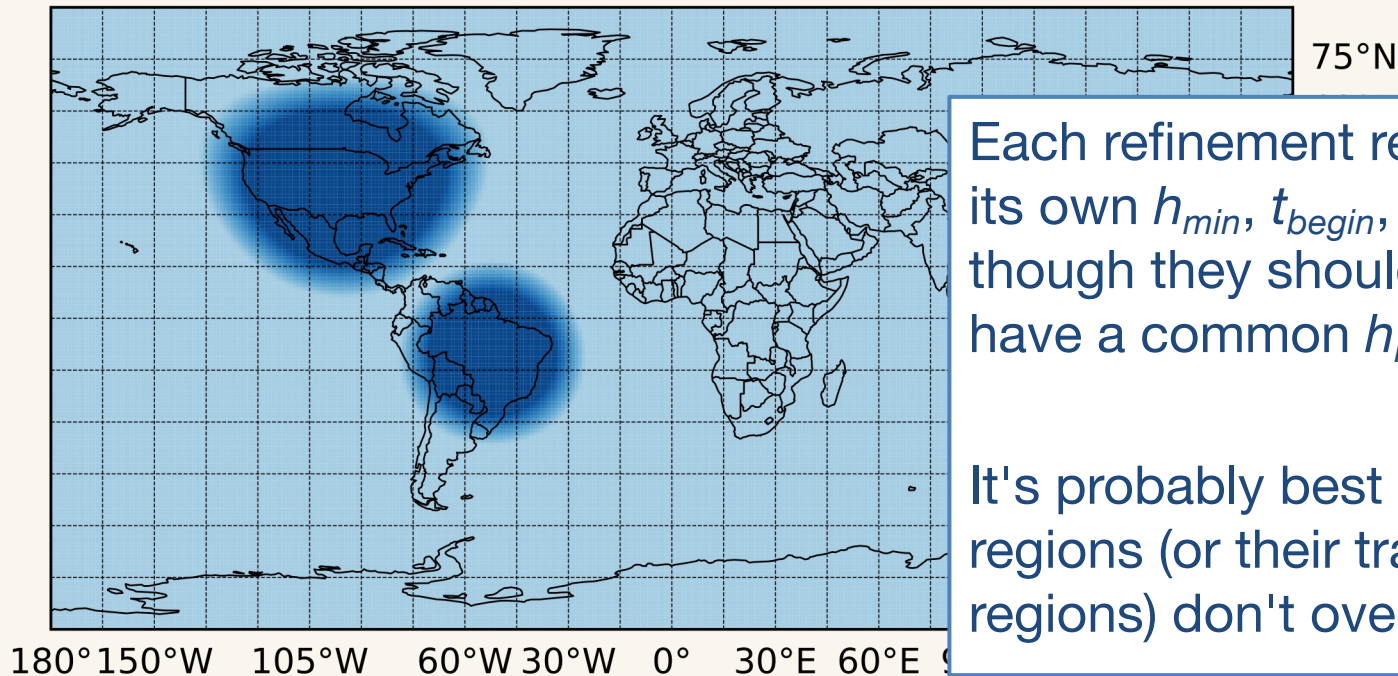


Above: Contours of nominal mesh size at 13 km and 59 km



Can we have multiple refinement regions?

Defining multiple refinement regions is simple...



Each refinement region can have its own h_{min} , t_{begin} , and t_{end} value, though they should obviously have a common h_{max} value!

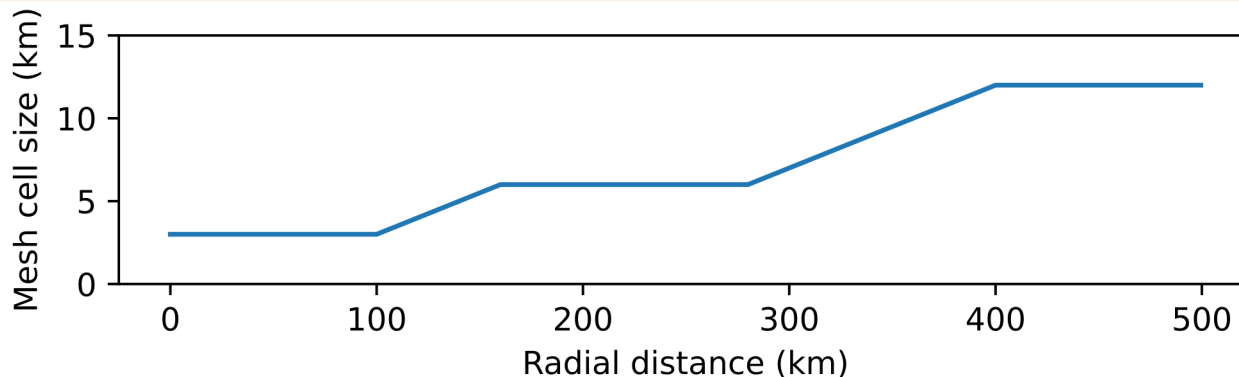
It's probably best if refinement regions (or their transition regions) don't overlap!

Just compute mesh distance fields over the sphere for each refinement region individually, then take the pointwise minimum of the mesh distance fields.

Can we have "nested" refinement regions?

One immediate idea is to simply define $h(r)$ with two transitions: $h_{min} \rightarrow h_{mid}$, and $h_{mid} \rightarrow h_{max}$

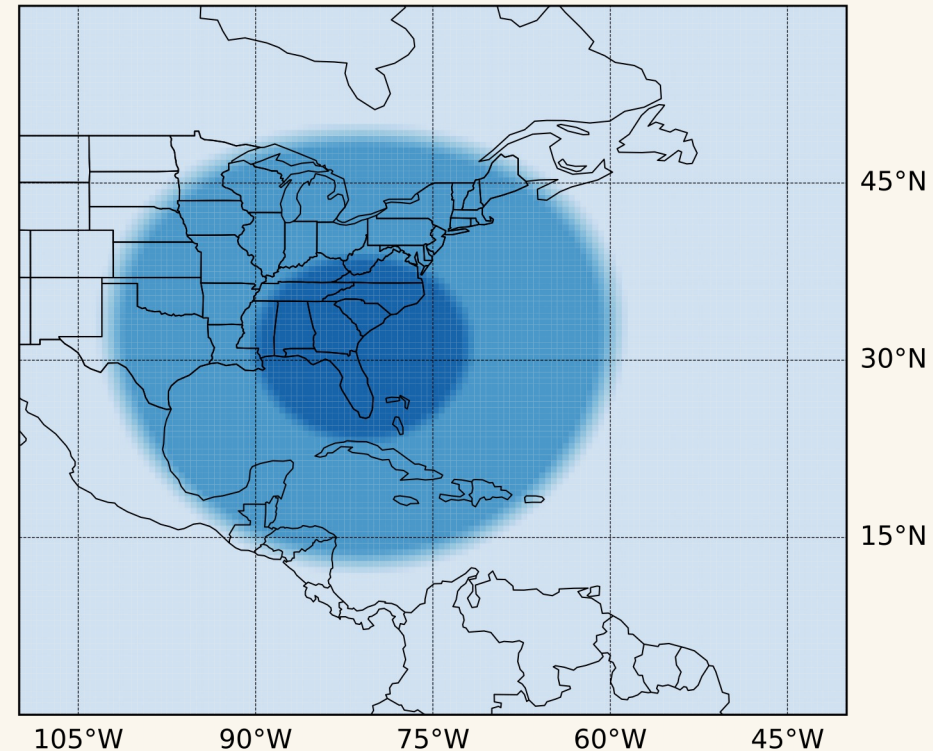
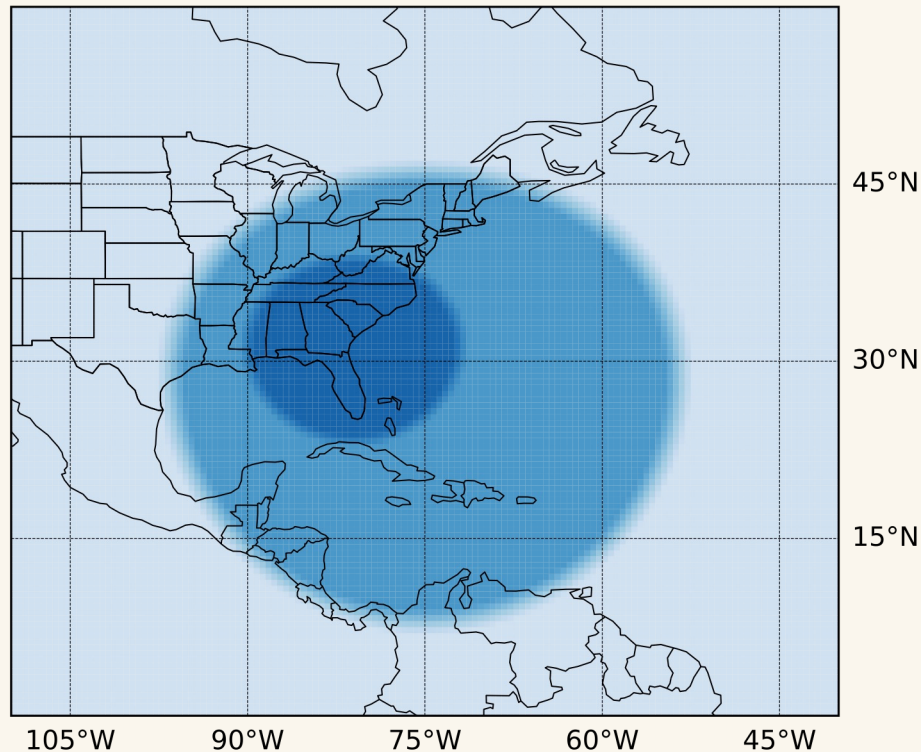
$$h(r) = \begin{cases} h_{min}, & \text{for } r \leq t_{begin,1} \\ h_{min} + (r - t_{begin,1}) \cdot \frac{h_{mid} - h_{min}}{t_{end,1} - t_{begin,1}}, & \text{for } t_{begin,1} < r \leq t_{end,1} \\ h_{mid}, & \text{for } t_{end,1} < r \leq t_{begin,2} \\ h_{mid} + (r - t_{begin,2}) \cdot \frac{h_{max} - h_{mid}}{t_{end,2} - t_{begin,2}}, & \text{for } t_{begin,2} < r \leq t_{end,2} \\ h_{max}, & \text{for } r > t_{end,2} \end{cases}$$



Left: Note that the transition regions have the same slope and therefore span different distances.

Can we have "nested" refinement regions?

The preceding idea works fine for concentric refinement regions (right)



But it won't allow us to have non-concentric refinement regions (left)

Can we have "nested" refinement regions?

For more general "nested" refinement regions, borrow the same idea from the case of multiple refinement regions

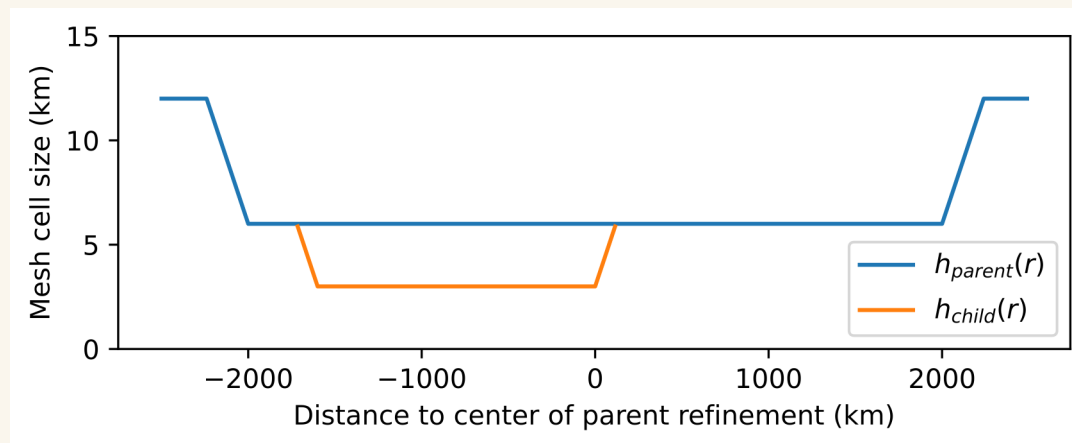
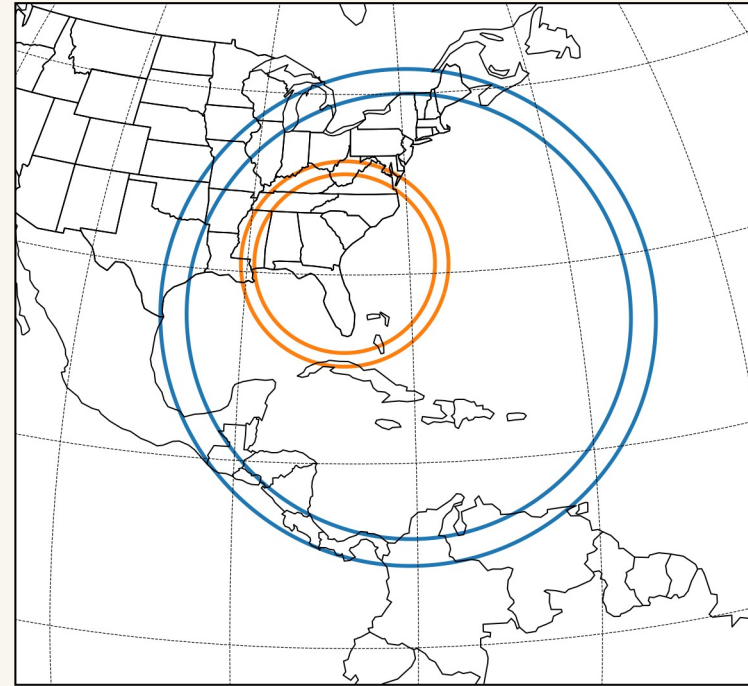
$$h_{parent}(r) = \begin{cases} h_{mid}, & \text{for } r \leq t_{begin,p} \\ h_{mid} + (r - t_{begin,p}) \cdot \frac{h_{max} - h_{mid}}{t_{end,p} - t_{begin,p}}, & \text{for } t_{begin,p} < r < t_{end,p} \\ h_{max}, & \text{for } r \geq t_{end,p} \end{cases}$$

$$h_{child}(r) = \begin{cases} h_{min}, & \text{for } r \leq t_{begin,c} \\ h_{min} + (r - t_{begin,c}) \cdot \frac{h_{mid} - h_{min}}{t_{end,c} - t_{begin,c}}, & \text{for } t_{begin,c} < r < t_{end,c} \\ \infty, & \text{for } r \geq t_{end,c} \end{cases}$$

$$h(r) = \min\{h_{parent}(r_{parent}), h_{child}(r_{child})\}$$

Notes on nested refinement regions

- When evaluating $h_{child}(r)$ ($h_{parent}(r)$), the argument should be the distance to the center of the child (parent) refinement region
- Ensure that "child" refinement region *and its transition region* are complete contained within the "parent" refinement region
- Ensure that h_{mid} parameter in $h_{child}(r)$ *and* $h_{parent}(r)$ are identical

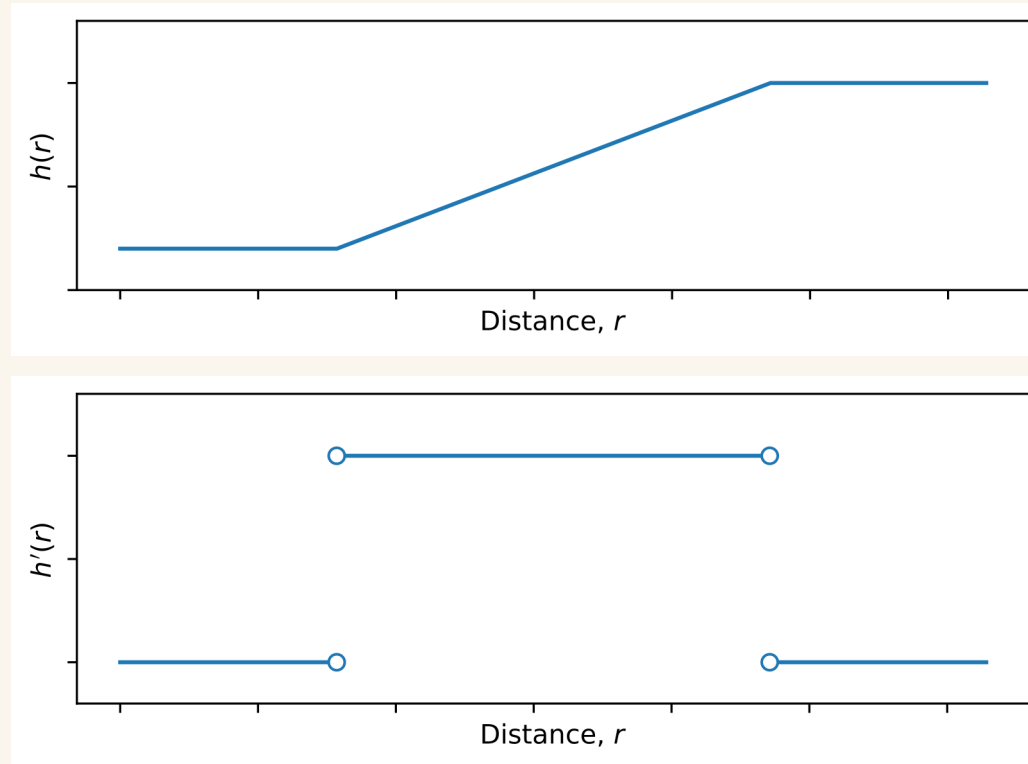


Another idea for a resolution transition function

What if we would like our mesh distance function to have a continuous first (and second?) derivative?

Our initial piecewise linear mesh distance function will clearly not suffice

We can start by considering a different form for $h'(r)$ or $h''(r)$



Above: A piecewise linear mesh distance function, $h(r)$, and its derivative.

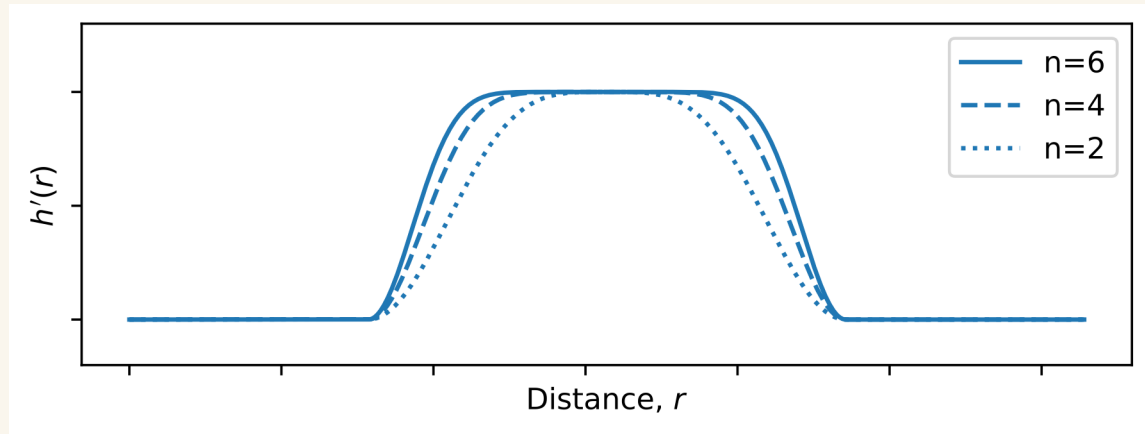
Another idea for a resolution transition function

The functions $1 - \sin^{2n}(x)$ have aesthetic appeal...

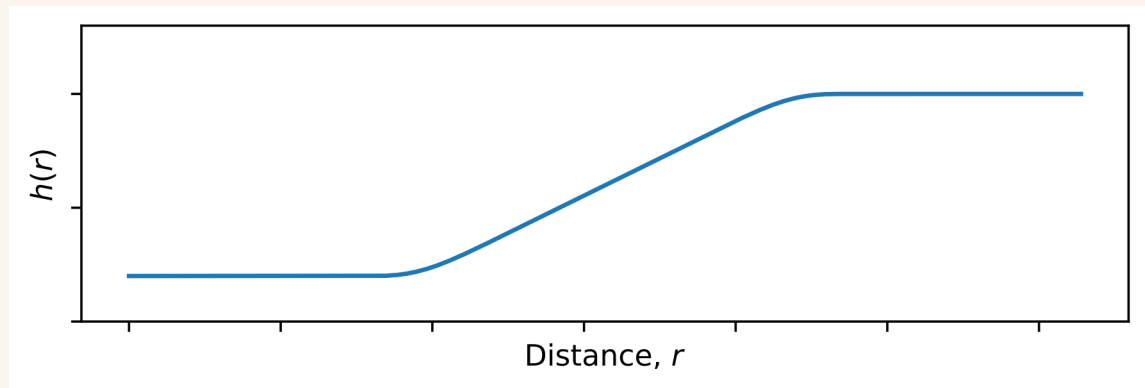
$$h'(x) = 1 - \sin^{2n}(x),$$

for $n \in \mathbb{N}^+$,

$$x \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right].$$



as do their integrals.



Another idea for a resolution transition function

A direct code implementation of the integrals is usable with simple normalization

Let

$$\hat{\tau}(x) := \frac{\tau(\pi x - \frac{\pi}{2}) - \tau(-\frac{\pi}{2})}{\tau(\frac{\pi}{2}) - \tau(-\frac{\pi}{2})}$$

where

$$\begin{aligned}\tau(x) := \int 1 - \sin^{12}(x) dx = & \frac{793}{1024}x \\ & + \frac{99}{512} \sin(2x) - \frac{495}{8192} \sin(4x) \\ & + \frac{55}{3072} \sin(6x) - \frac{33}{8192} \sin(8x) \\ & + \frac{3}{5120} \sin(10x) - \frac{1}{24576} \sin(12x) + C\end{aligned}$$

Note: the least common multiple of the denominators is 122880

Another idea for a resolution transition function

Our piecewise $h(r)$ function now has the form:

$$h(r) = \begin{cases} h_{min}, & \text{for } t_{begin} \geq r \\ h_{min} + (h_{max} - h_{min}) \cdot \hat{\tau} \left(\frac{r - t_{begin}}{t_{end} - t_{begin}} \right), & \text{for } t_{begin} < r < t_{end} \\ h_{max}, & \text{for } r \geq t_{end} \end{cases}$$

For comparison, our $h(r)$ function with a linear transition:

$$h(r) = \begin{cases} h_{min}, & \text{for } t_{begin} \geq r \\ h_{min} + (h_{max} - h_{min}) \cdot \frac{r - t_{begin}}{t_{end} - t_{begin}}, & \text{for } t_{begin} < r < t_{end} \\ h_{max}, & \text{for } r \geq t_{end} \end{cases}$$

What about other shapes of refinement regions?

Re-using the same distance functions $h(r)$ that we used for circular refinement, we could, for example, let

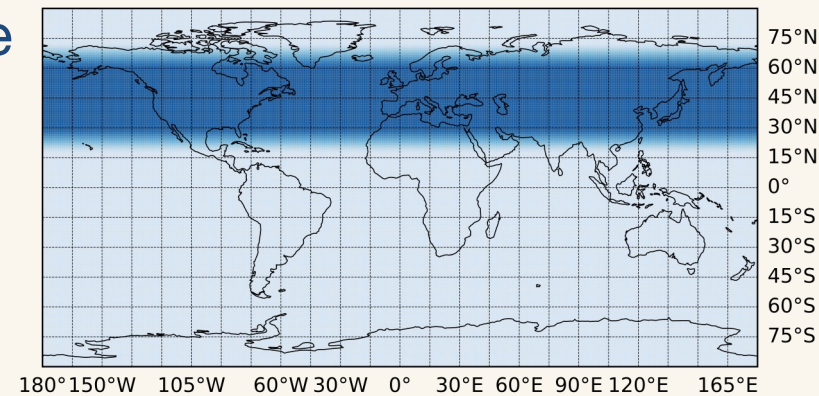
$$r = R_{earth} \cdot |\phi_1 - \phi_2|$$

- ϕ is latitude, in radians

When $\phi_1 = \pi/4$ and $\phi_2 =$ any other point on the globe, we get refinement in the northern mid-latitudes

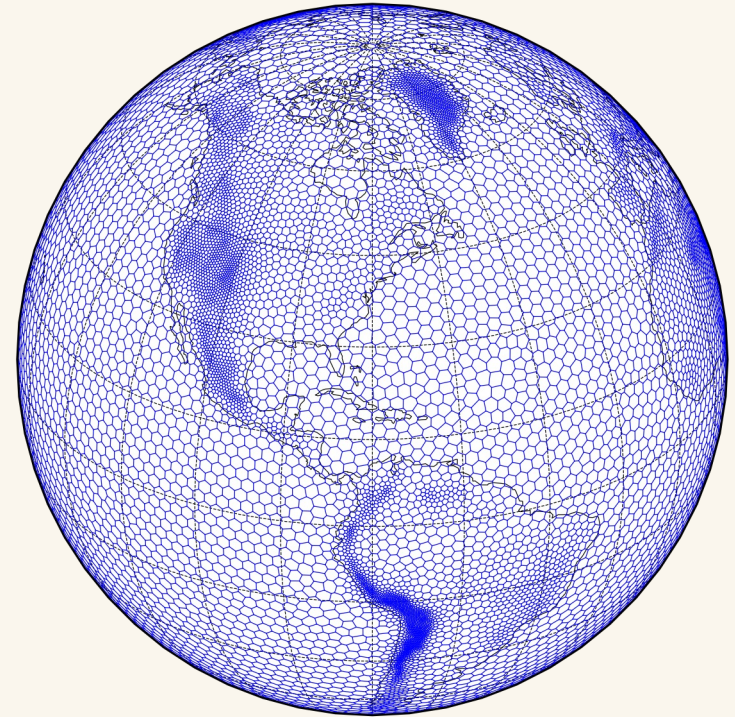
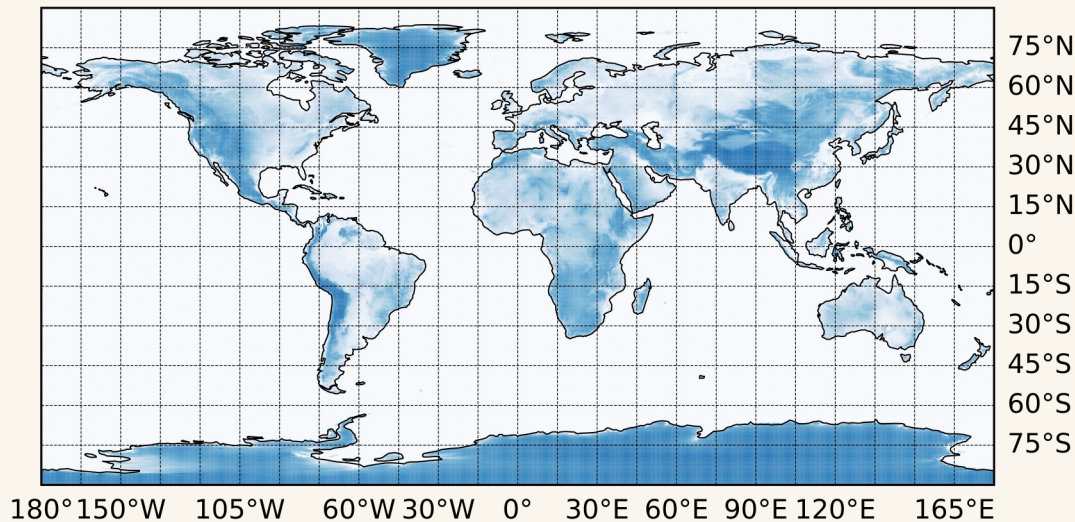
Other ideas:

- r = distance to a line segment
- r = minimum distance to a polygon



Non-analytic distance functions

Of course, we can use something besides an analytic function. We could use a raster dataset and interpolate within that in our `get_hfun` function, for example

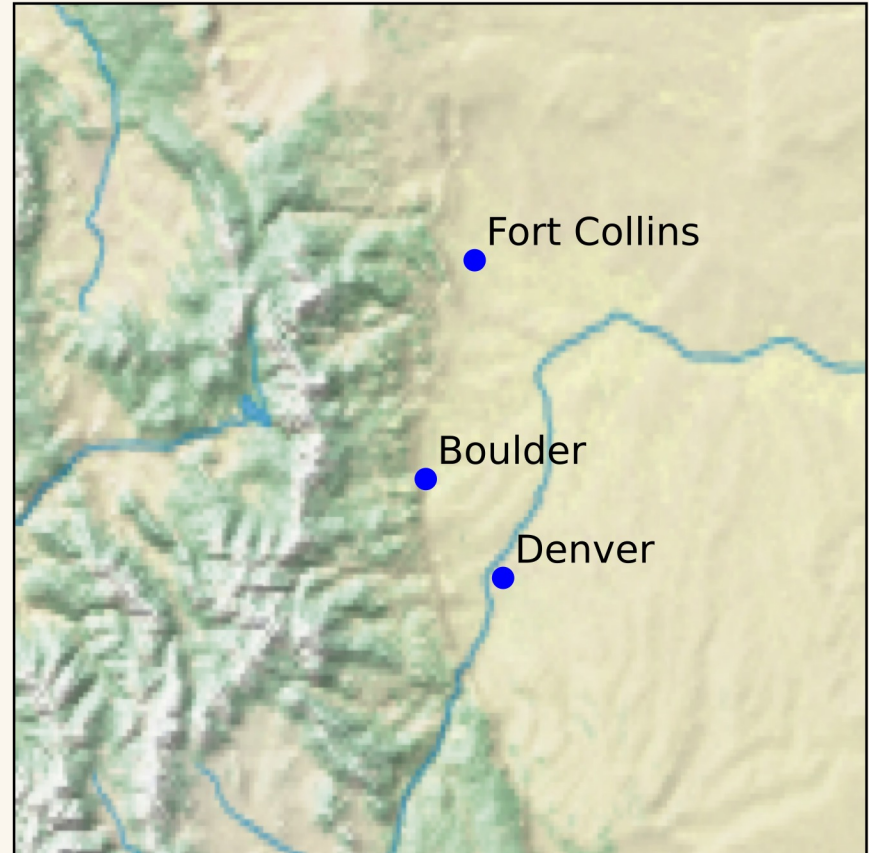


For a raster datasets,
`scipy.interpolate.RegularGridInterpolator`
may be especially helpful!

With great power comes great responsibility

Exercise good judgement in designing variable-resolution meshes

- Don't change mesh size too quickly. Limit gradients (km of cell size) / (km of distance) to a few percent
- Refinement regions should be sufficiently large (in grid cell space), ideally several hundred cells in diameter or more
- Better to have one larger refinement region than several smaller, closely spaced regions of refinement

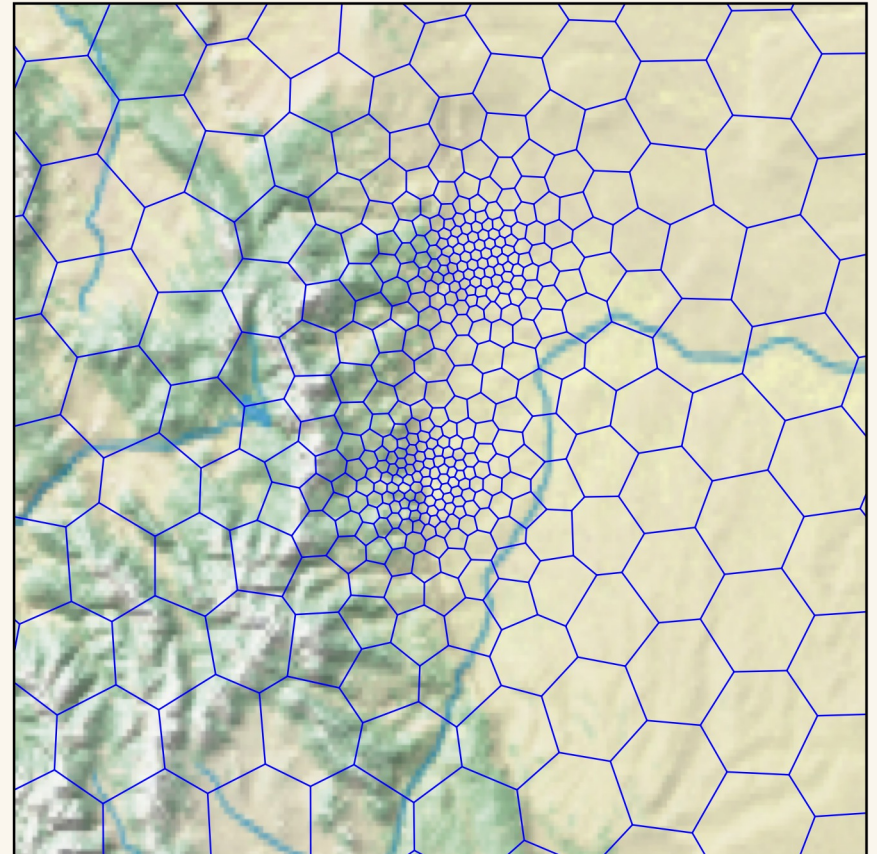


Suppose we would like a mesh that refines from 30 km down to 3 km grid distance over Boulder and Fort Collins

With great power comes great responsibility

A not-so-good mesh covering Boulder and Fort Collins

- ✗ Some cells are $>35\%$ larger in diameter than their neighbors (an order of magnitude too quickly!)
- ✗ Refinement regions are only $O(10)$ cells in diameter (an order of magnitude too small!)
- ✗ Refinement regions are so close that their transition regions begin to overlap



The mesh above violates all of the guidelines from the previous slide!

Assessing mesh quality

The [mpas_jigsaw tutorial](#) repository contains a simple script to help assess mesh quality: `mesh_quality.py`

- Run with the `grid.nc` netCDF mesh file as input
- Still a work in progress – expect better checks to be added over time!

Current checks include:

Among the current tests,
these may be the most
important to focus on



```
Min/max dcEdge (m) : 8955.998 83924.640
Min/max dvEdge (m) : 2745.289 58330.347
```

```
sum(areaCell) : 12.56637061436
sum(areaTriangle) : 12.56637061436
sum(kiteAreasOnVertex) : 12.56637061436
4 * Pi : 12.56637061436
```

No obtuse triangles!

```
Min nominal cell size gradient: 0.0
Max nominal cell size gradient: 0.0299
```

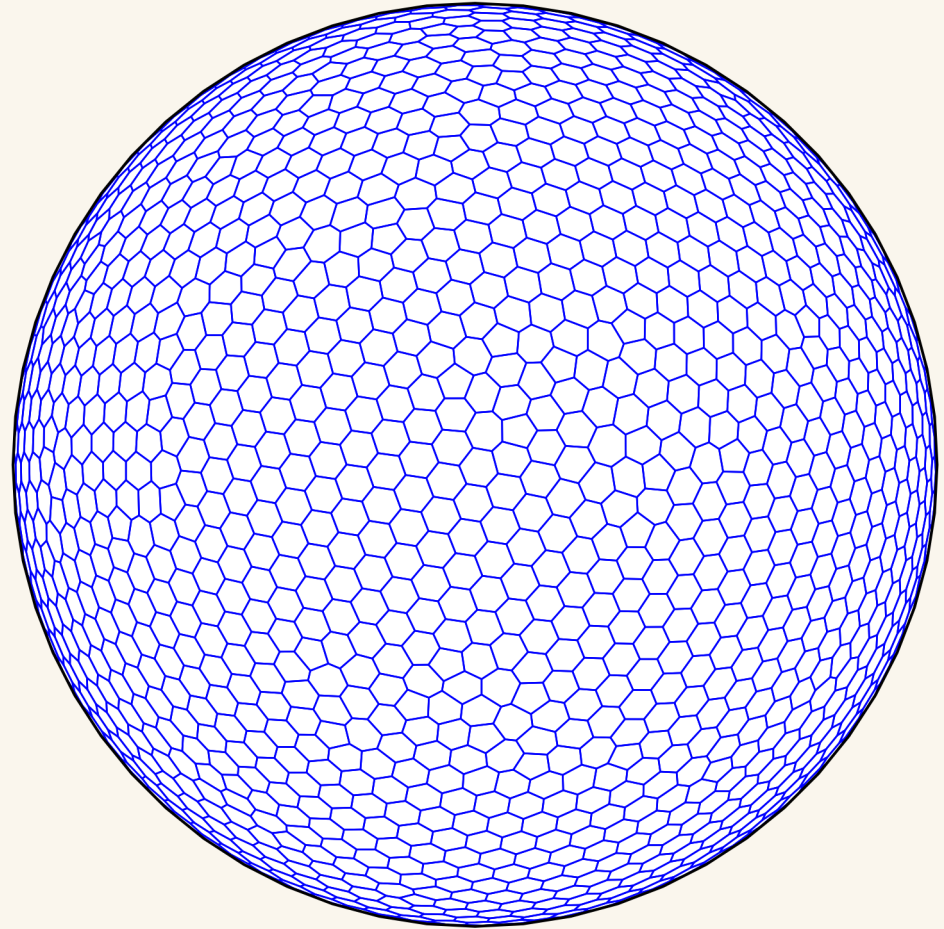
Outline

1. A quick refresher on Voronoi tessellations
2. The goals of mesh generation
3. Existing mesh tools
4. A simple JIGSAW-based workflow
5. Generating variable-resolution meshes
6. Generating quasi-uniform meshes

Generating quasi-uniform meshes

What happens if we simply set our HFUN array to, say, 480 km everywhere?

- The resulting mesh will likely have 7-sided cells and less-desirable cell shapes
- Is there a way to impose some icosahedral structure in the mesh produced by JIGSAW?

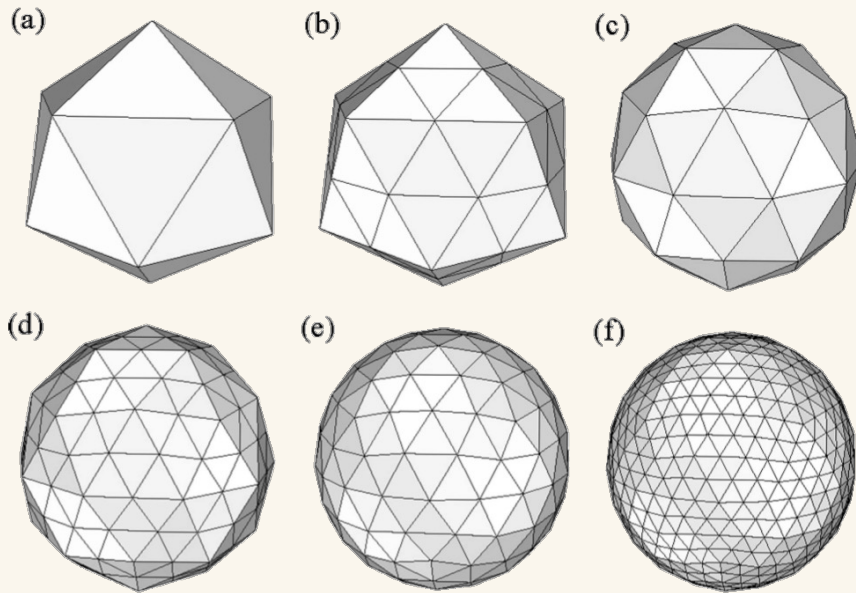


The solution is to provide an initial set of points to JIGSAW

- Filename specified by `INIT_FILE` option in JIGSAW's config file

Generating quasi-uniform meshes

If we seek icosahedral structure, what better thing to do than to start with an icosahedron!



Bisecting an icosahedral mesh and projecting vertices to the sphere. Lipscomb and Ringler (2005)

With $R_{\text{earth}} = 6371.229$ km, the base icosahedron has a grid length of $\Delta x = 7053.898$ km

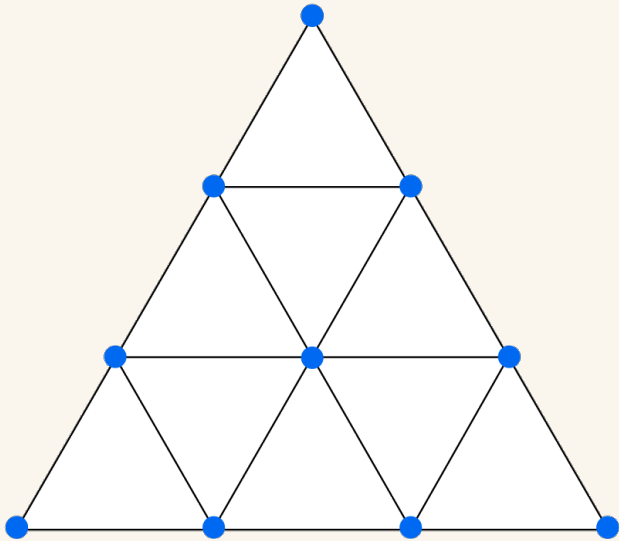
n	$\Delta x / 2^n$
0	7053.898
1	3526.949
2	1763.475
3	881.737
4	440.869
5	220.434
6	110.217

Generating quasi-uniform meshes

What can we do if we need, e.g., a quasi-uniform 10 km mesh?

- This resolution is intermediate to what we get with division factors of 2^9 and 2^{10}

Generalizing the idea of bisection, we can divide triangles in the icosahedron by any positive integer factor



The **sphtri_subdiv** tool from Anthony Islas's **mesh_tools** repository does just this!

https://github.com/islas/mesh_tools

Generating quasi-uniform meshes

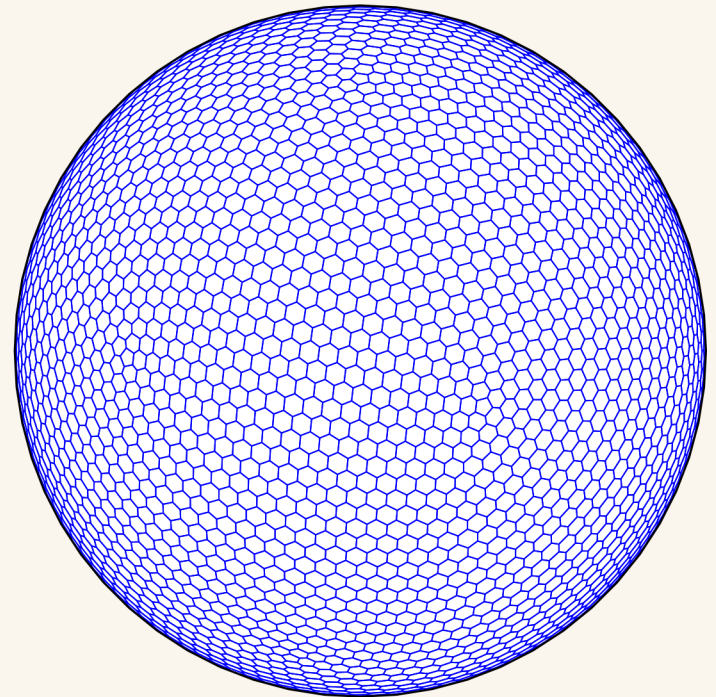
Whatever the (integral) division factor, the **sphtri_subdiv** tool can produce a JIGSAW-format mesh file that can be supplied to JIGSAW through the `INIT_FILE` option

- The [mpas_jigsaw tutorial](#) repository includes a JIGSAW mesh file describing an icosahedron: `ICOS.msh`
- Try, e.g., the command

```
sphtri_subdiv ICOS.msh \
120KM.msh 63
```
- Then add the line

```
INIT_FILE=120KM.msh
```

to the `MESH.jig` file
- Finally, let `hfun.py` describe a uniform resolution of 120 km



Above: A mesh using initial points from a division of the icosahedron by a factor of 21 (nominal $dx \sim 335$ km).