An Overview of the Structure of MPAS Meshes

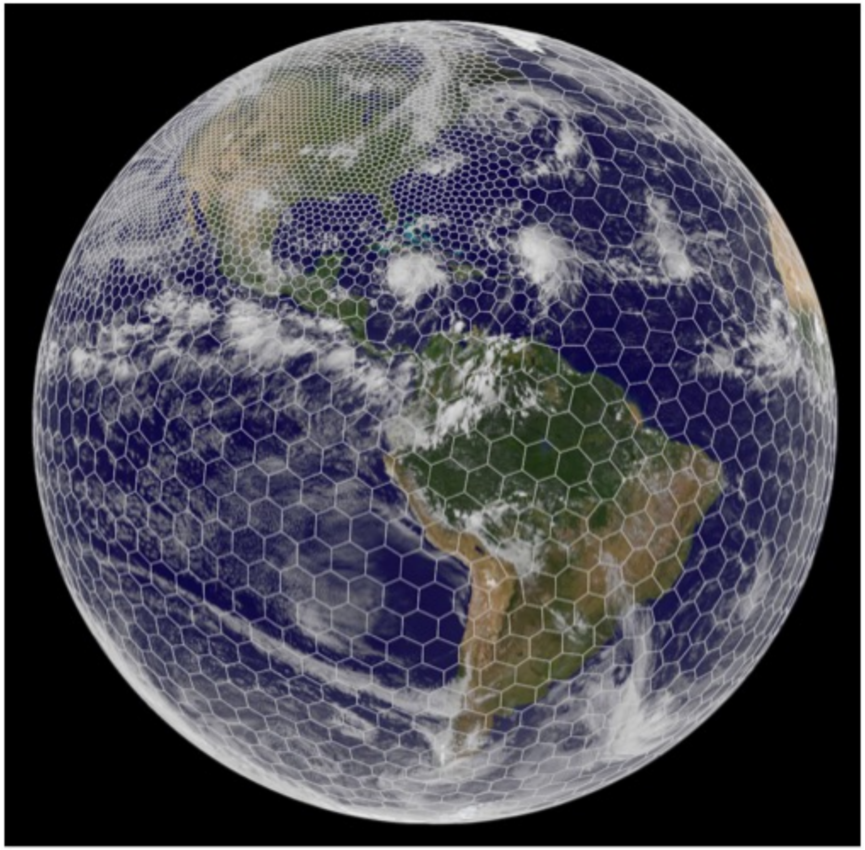
Michael G. Duda
NCAR/MMM

This material is based upon work supported by the National Center for Atmospheric Research, which is a major facility sponsored by the National Science Foundation under Cooperative Agreement No. 1852977.

Background

A defining feature of MPAS models is their use of **centroidal Voronoi tessellations** (CVTs) with a C-grid staggering

- When constrained to lie on the surface of a sphere, we often call them spherical centroidal Voronoi tessellations (SCVTs)



Background

A defining feature of MPAS models is their use of **centroidal Voronoi tessellations** (CVTs) with a C-grid staggering

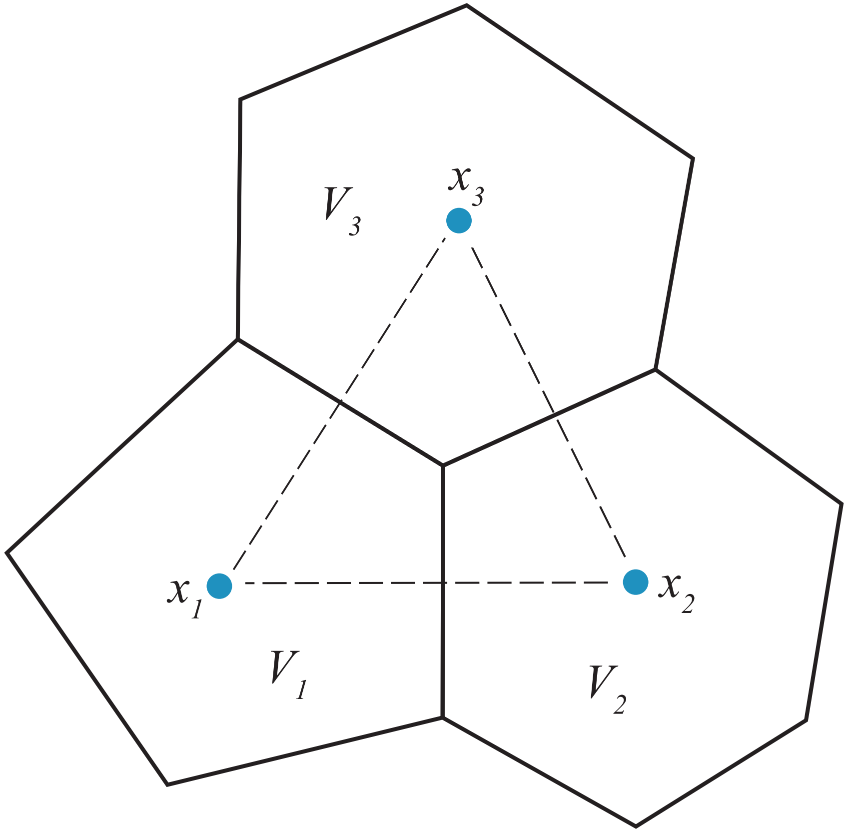
– When constrained to lie on the surface of a sphere, we often call them spherical centroidal Voronoi tessellations (SCVTs)

Voronoi = each grid volume (cell) V_i is uniquely associated with a *generating point* $\boldsymbol{x}_i$ such that all points within V_i are closer to $\boldsymbol{x}_i$ than to any other $\boldsymbol{x}_j$

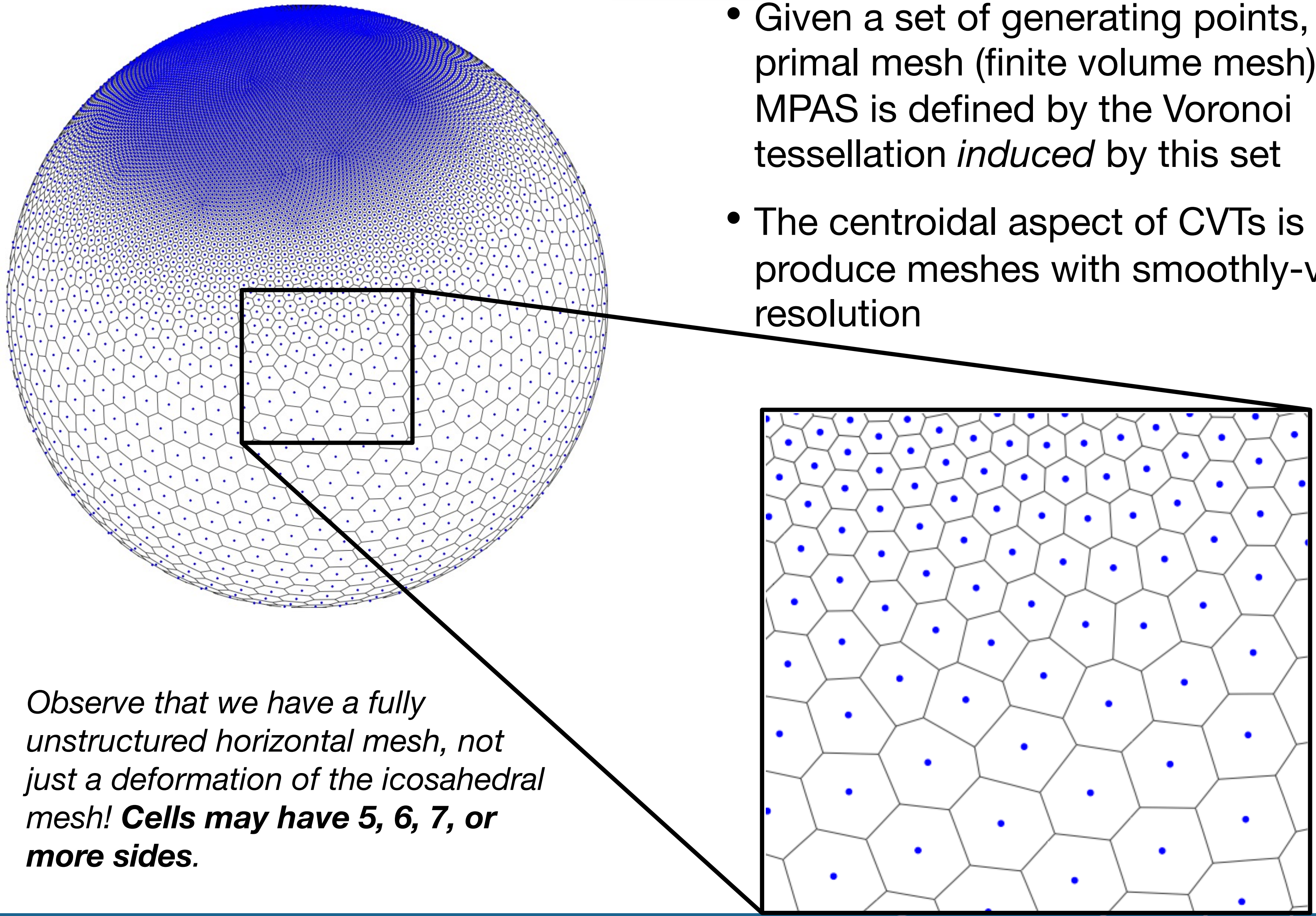
- Lines joining generating points of adjacent cells are

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

Centroidal = the generating point for each Voronoi cell is also the mass centroid of that cell (**w.r.t. some density function**)



Background

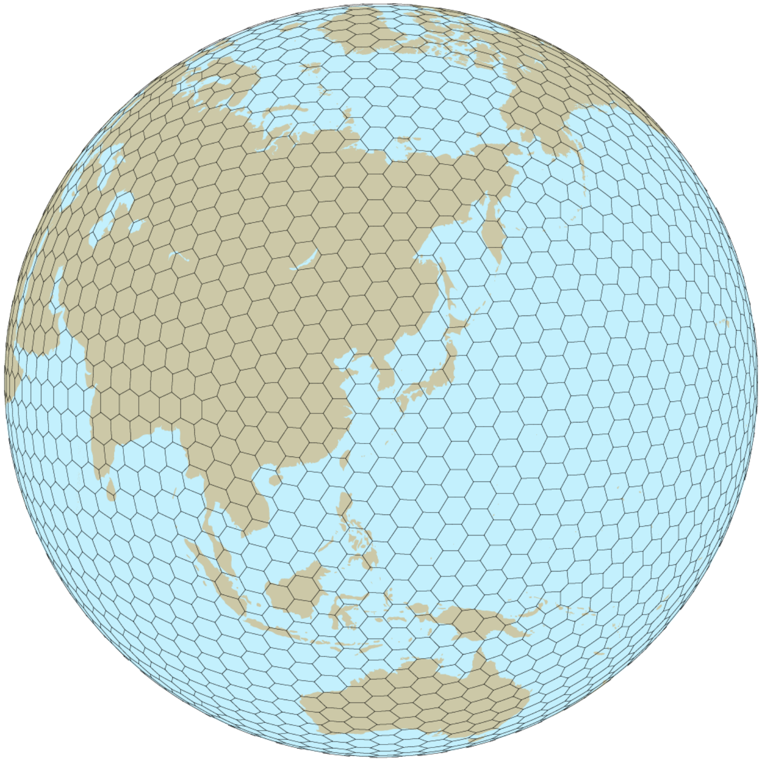


- Given a set of generating points, the primal mesh (finite volume mesh) in MPAS is defined by the Voronoi tessellation *induced* by this set
- The centroidal aspect of CVTs is used to produce meshes with smoothly-varying resolution

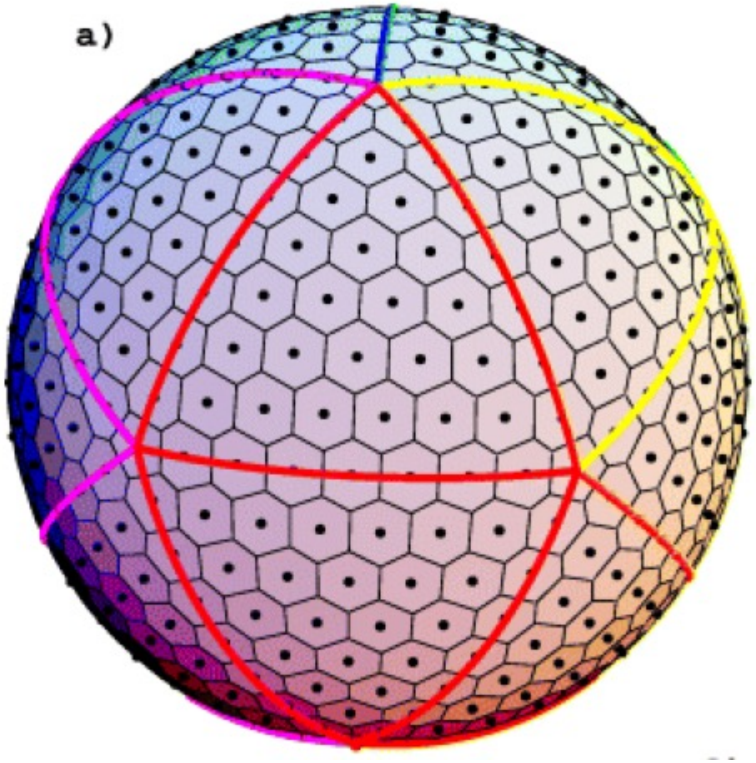
Observe that we have a fully unstructured horizontal mesh, not just a deformation of the icosahedral mesh! ***Cells may have 5, 6, 7, or more sides****.*

How does MPAS keep track of this unstructured mesh?

Quasi-uniform MPAS meshes look just like icosahedral meshes...



A quasi-uniform MPAS mesh



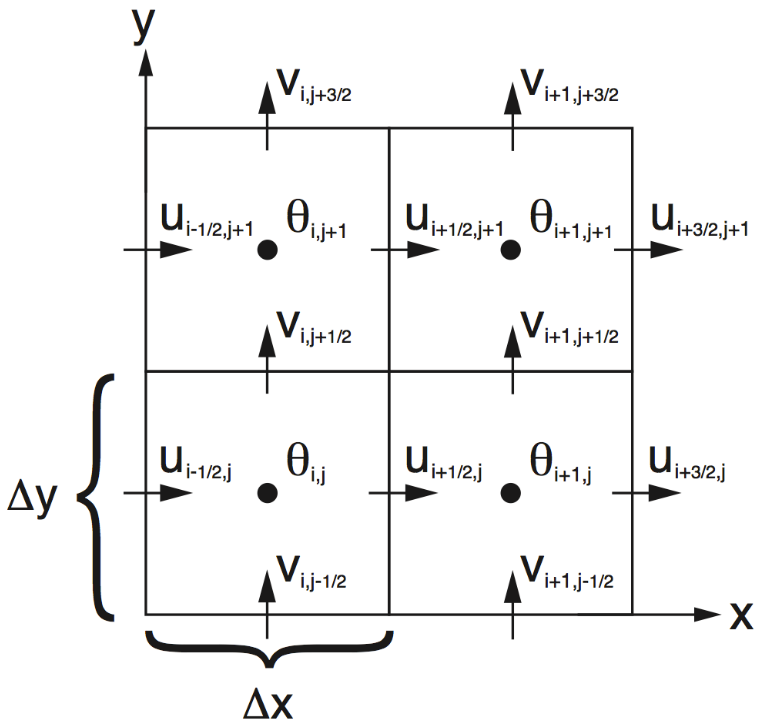
An icosahedral mesh

... but the MPAS solver considers every mesh as a completely general, unstructured mesh. There are no special cases!

How does MPAS keep track of this unstructured mesh?

One can start to imagine ways to identify neighboring cells implicitly based on the index or location of each cell

- In a rectangular mesh, our neighbors are at (i+1, j), (i-1, j), (i, j+1), (i, j-1)

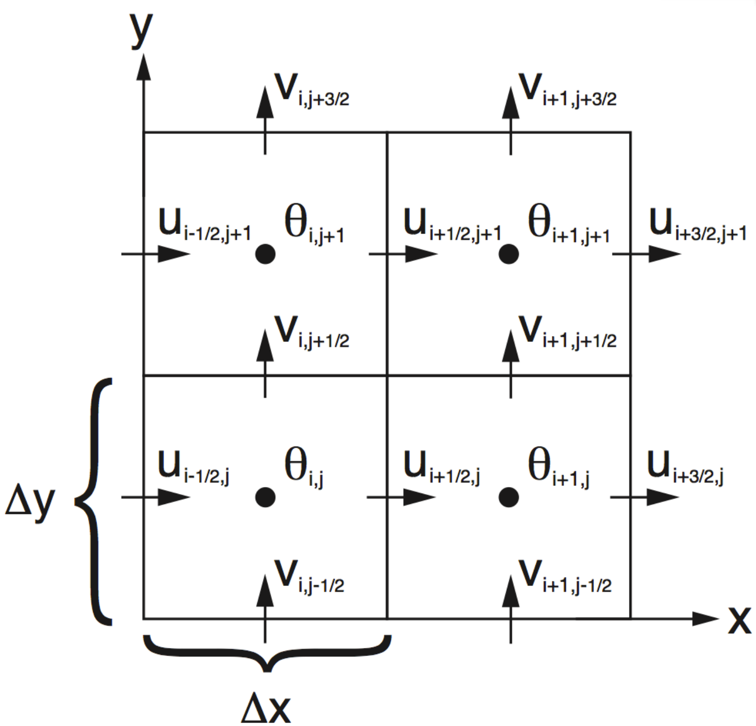


Above: A region from the ARW C-staggered grid, stored in a 2-d array.

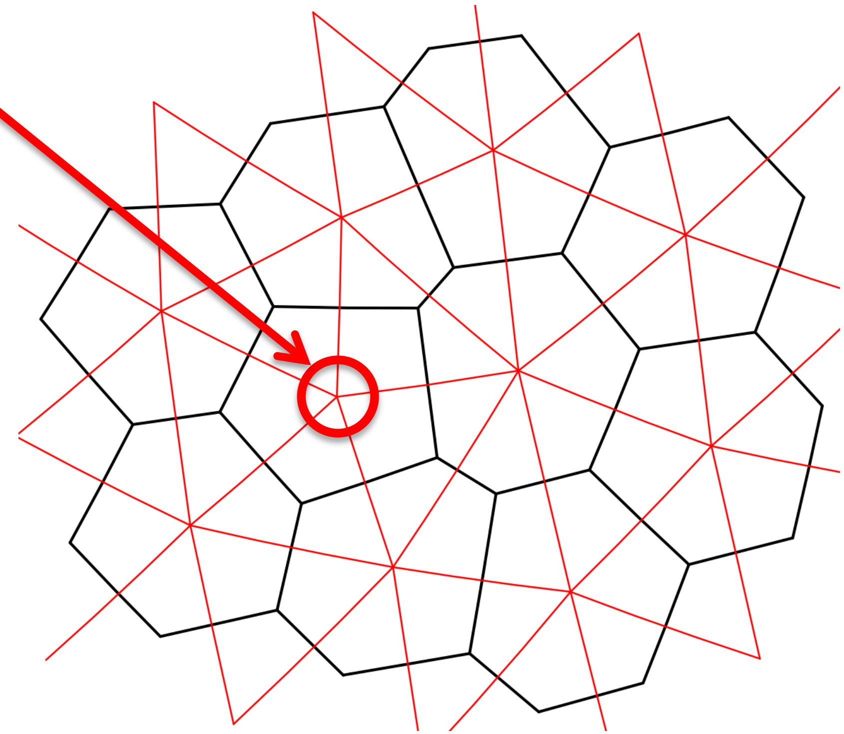
How does MPAS keep track of this unstructured mesh?

One can start to imagine ways to identify neighboring cells implicitly based on the index or location of each cell

- In a rectangular mesh, our neighbors are at (i+1, j), (i-1, j), (i, j+1), (i, j-1)
- Who is the "next" cell after this one in any direction?



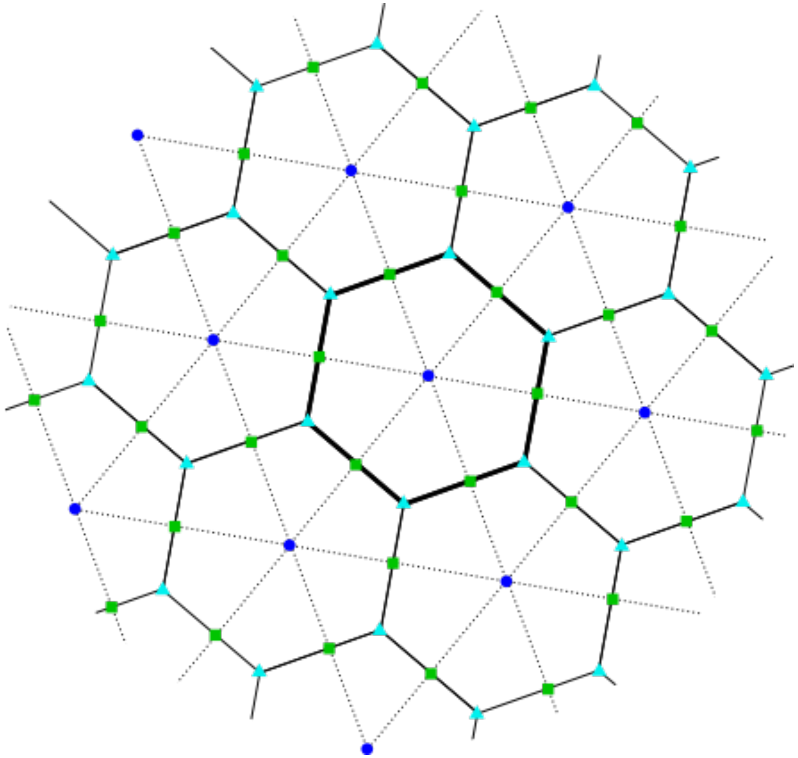
Above: A region from the ARW C-staggered grid, stored in a 2-d array.



Above: A region from an MPAS mesh showing Voronoi regions (black) and Delaunay triangles (red).

How does MPAS keep track of this unstructured mesh?

Schemes for implicitly finding the indices/identities (the "IDs") of neighboring mesh elements (i.e., cells, edges, vertices) are bound to fail...
... so we must find them explicitly through connectivity fields that are the foundation of the MPAS mesh representation.



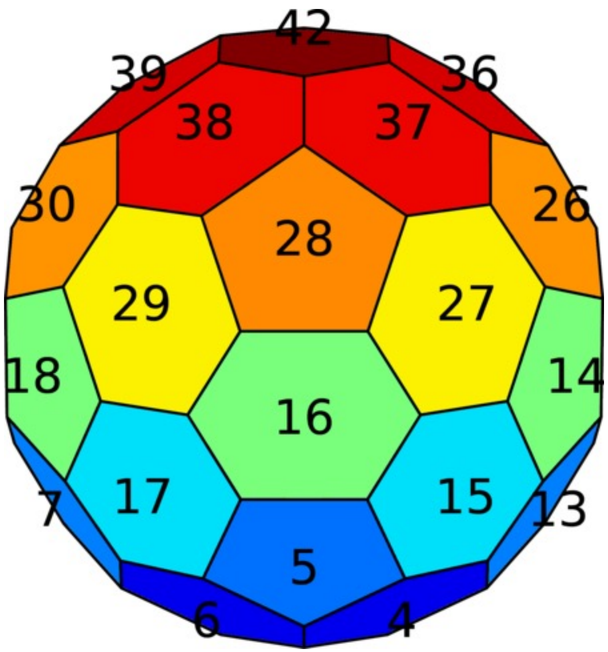
Three types of mesh elements are tracked in the mesh representation:

- **Cell** locations (blue circles) - the generating points of the Voronoi mesh
- **Vertex** locations (cyan triangles) - the corners of primal mesh cells
- **Edge** locations (green squares) - the points where the dual mesh edges intersect the primal mesh edges

How does MPAS keep track of this unstructured mesh?

For the unstructured, horizontal dimension there is nothing to be gained from using 2-d arrays...

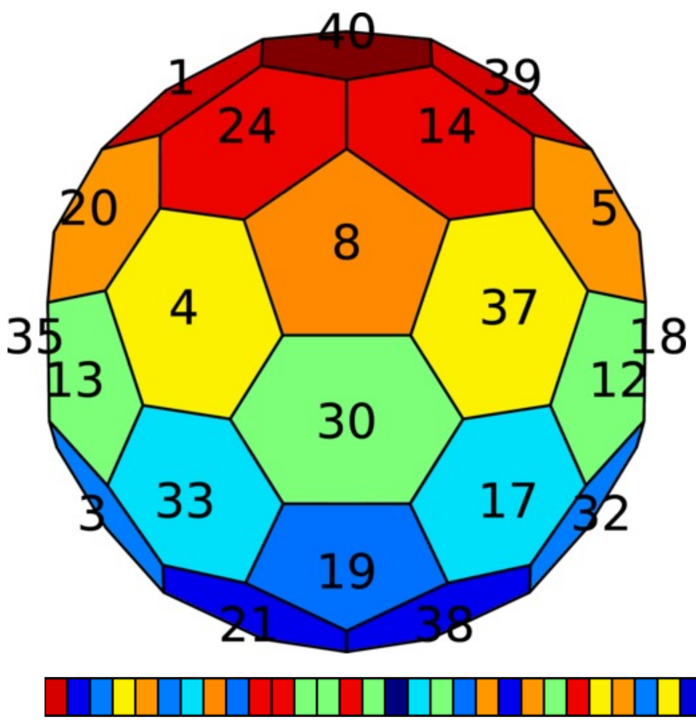
...hence, the horizontal dimension is collapsed into a single array dimension: we then have a simple list of elements



Example: For some 2-d field (shown in color) defined on mesh cells, that field is stored in a 1-d array (bottom) that is indexed by cell number (labeled in black).

How does MPAS keep track of this unstructured mesh?

For the unstructured, horizontal dimension there is nothing to be gained from using 2-d arrays...
...hence, the horizontal dimension is collapsed into a single array dimension: we then have a simple list of elements



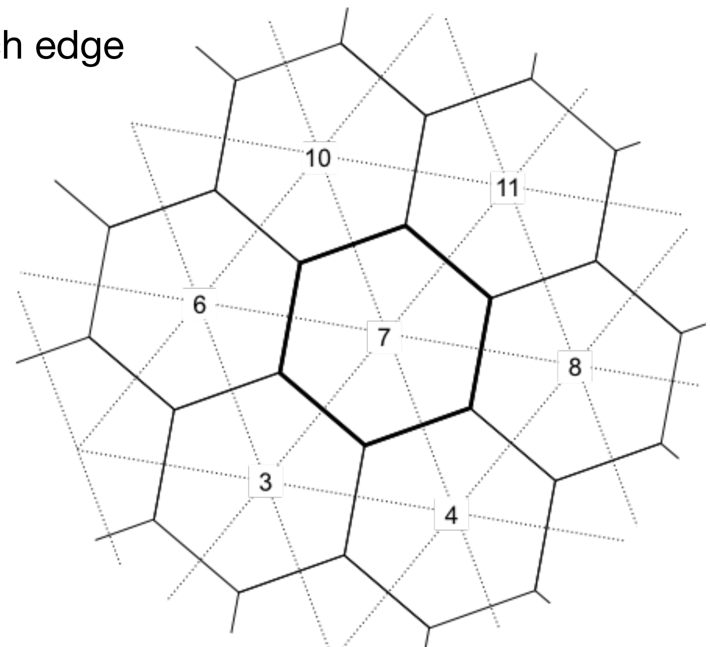
From the perspective of the MPAS solver, any ordering of cells in the mesh is as good as any other[1], as long as the mesh representation is consistent with this ordering.

[1]*Though some orderings may give better performance, e.g., due to better cache reuse.*

Explicit connectivity fields

- **nEdgesOnCell(nCells)** – the number of neighbors for each cell
- **cellsOnCell(maxEdges, nCells)** – the indices of neighboring cells for each cell
- **edgesOnCell(maxEdges, nCells)** – the indices of bounding edges for each cell
- **verticesOnCell(maxEdges, nCells)** – the indices of corner vertices for each cell
- **edgesOnVertex(vertexDegree,nVertices)** – the indices of edges incident with each vertex
- **verticesOnEdge(2,nEdges)** – the indices of endpoint vertices for each edge
- **cellsOnVertex(vertexDegree,nVertices)** – the indices of cells meeting at each vertex
- **cellsOnEdge(2,nEdges)** – the indices of cells separated by each edge

```
nEdgesOnCell(7)=6   cellsOnCell(1,7)=8
                    cellsOnCell(2,7)=11
                    cellsOnCell(3,7)=10
                    cellsOnCell(4,7)=6
                    cellsOnCell(5,7)=3
                    cellsOnCell(6,7)=4
```

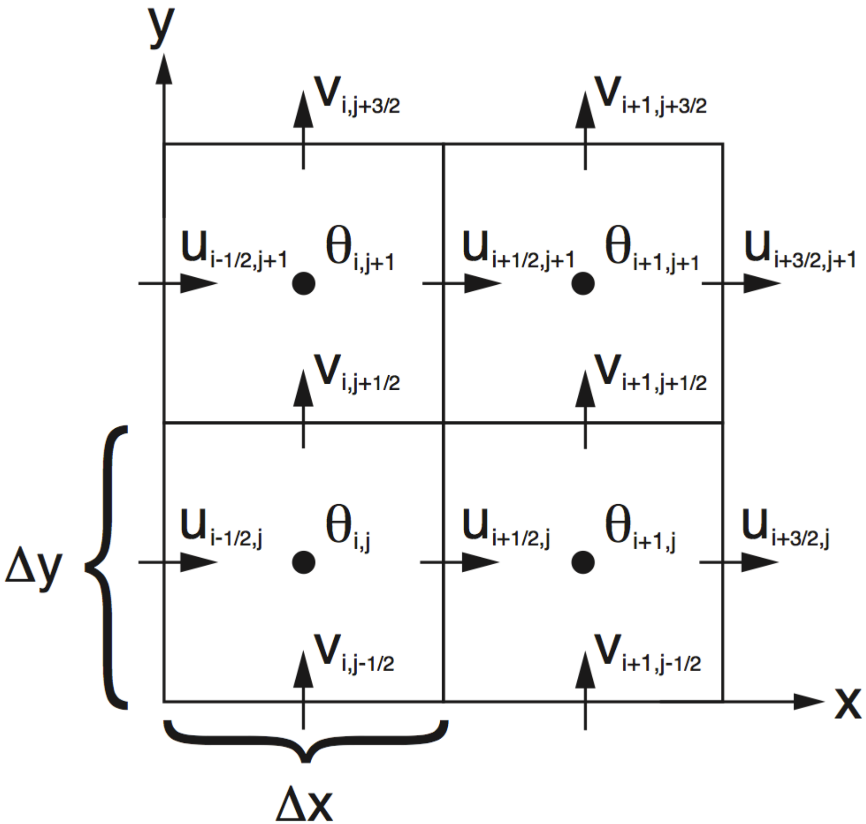


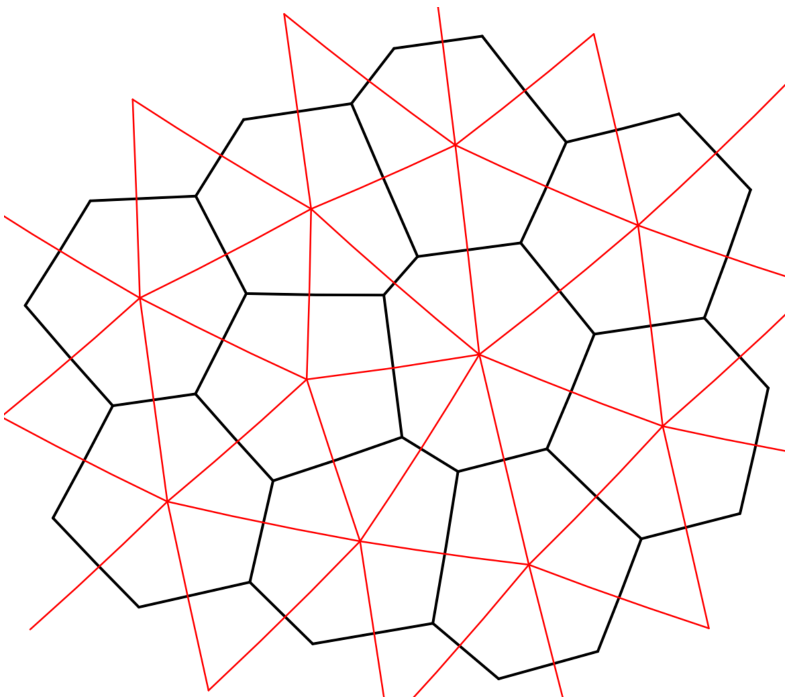
At model start-up, all indices in these arrays are re-numbered to a local indexing scheme.

MPAS
Model for Prediction Across Scales

Horizontal wind vectors: how are these defined?

In a rectangular C-grid, which directions represent positive U and positive V?

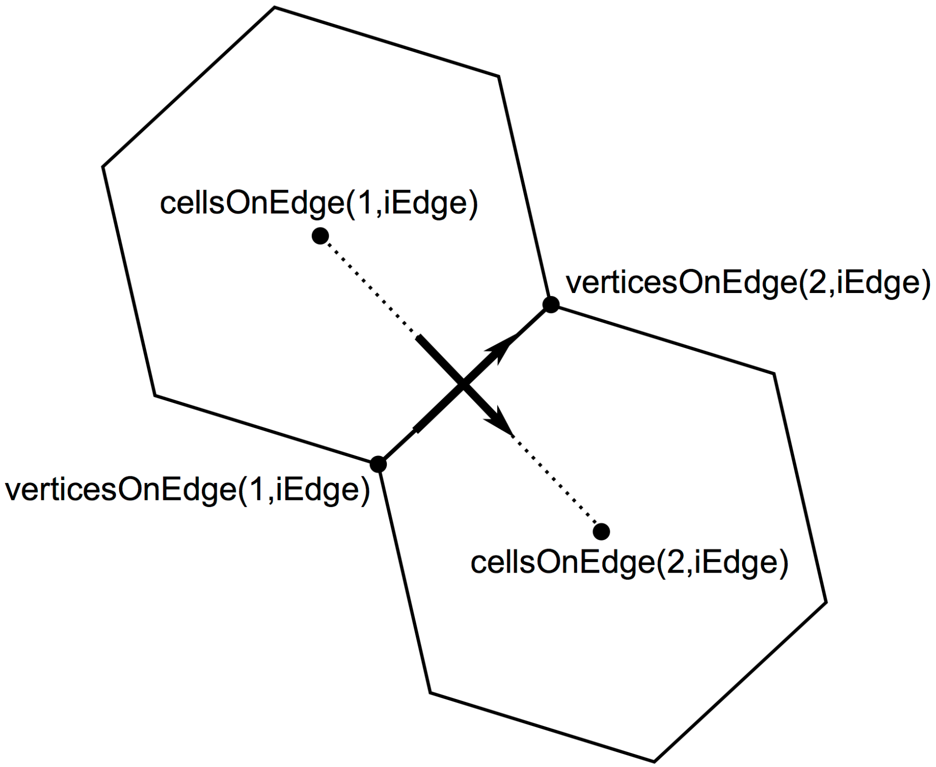




On a rectangular grid, one might say that positive U flows from left to right, and positive V flows from bottom to top when looking down on the xy-plane.

On a CVT mesh, one could introduce a similar definition, but we have only U, not V, so such a definition becomes more complicated...

Horizontal wind vectors: how are these defined?



Positive u (normal) velocity is always defined as flow from cellsOnEdge(1,iEdge) to cellsOnEdge(2,iEdge) for edge iEdge

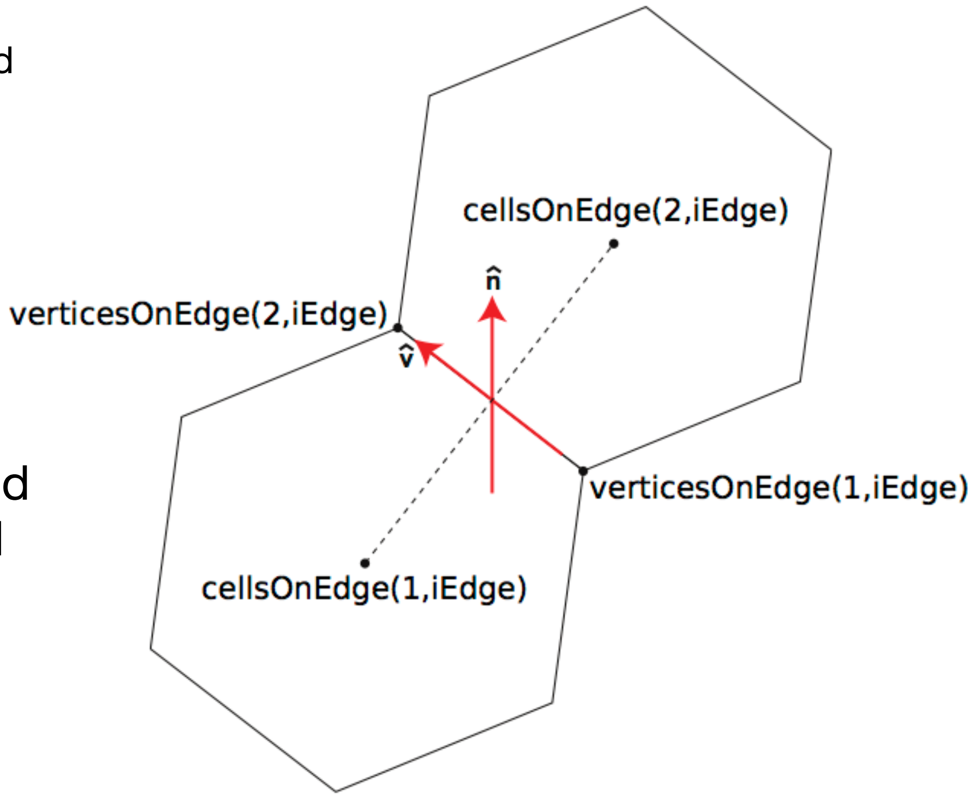
Positive v (tangential) velocity is always defined as flow from verticesOnEdge(1,iEdge) to verticesOnEdge(2,iEdge) for edge iEdge

The cross product of the positive *u* and *v* vectors always points upward (out of the plane)

Horizontal wind vectors: how are these defined?

angleEdge(nEdges) – angle between east and positive u

$$\text{angleEdge} = \arcsin \left\| \hat{\mathbf{n}} \times \hat{\mathbf{v}} \right\|$$

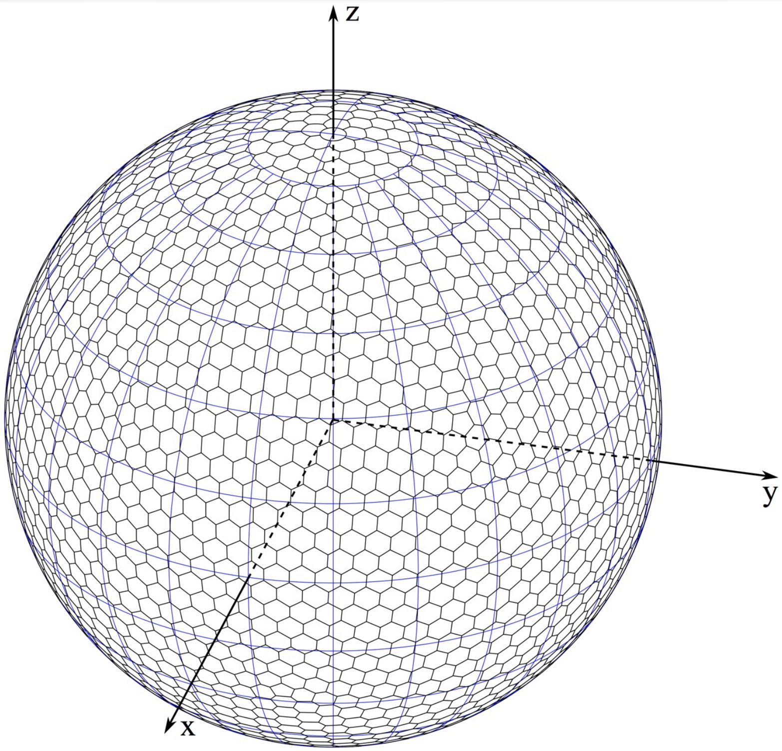


Earth-relative horizontal winds, u_{zonal} and $u_{meridional}$, can be calculated using u and v:

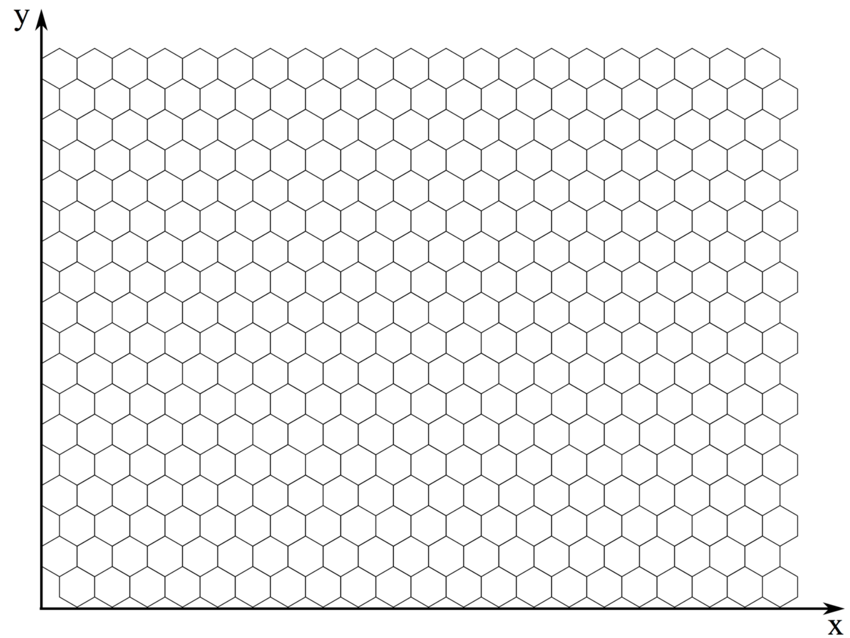
$$\begin{bmatrix} u_\lambda \\ u_\phi \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix}$$

where α is angleEdge.

Which mesh geometries does MPAS support?



On the surface of the sphere: all distances and areas are computed in spherical geometry.

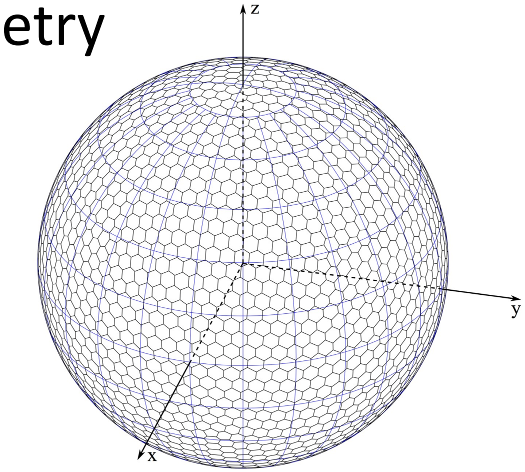


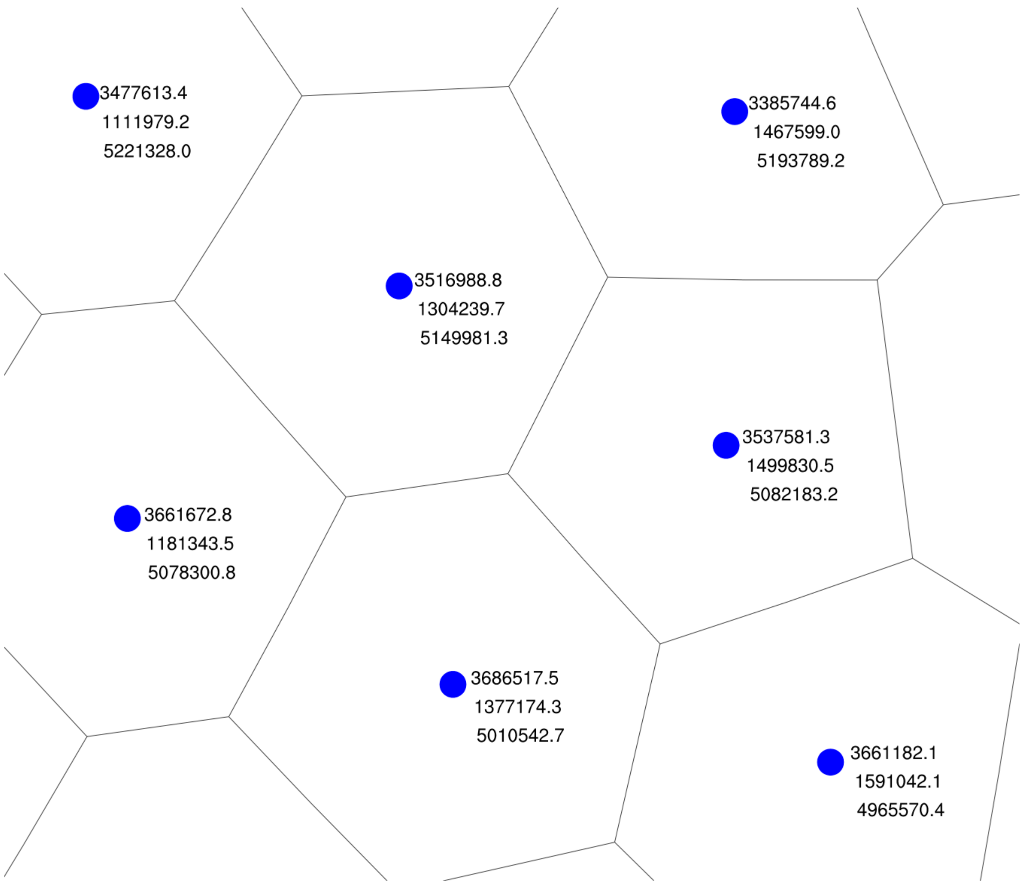
In the Cartesian plane: all distances and areas are computed in Euclidean geometry.

$$x = r\cos(\lambda)\cos(\phi)$$
$$y = r\sin(\lambda)\cos(\phi)$$
$$z = r\sin(\phi)$$

$$\phi = \arcsin\left(\frac{z}{r}\right)$$
$$\lambda = \arctan\left(\left|\frac{y}{x}\right|\right)$$

Notes about MPAS mesh geometry





Above: Cartesian coordinates for cell locations near (52.9°N lat, 20.8°E lon) in a variable-resolution spherical mesh with radius 6371229 m.

Global Cartesian coordinates are computed for each element

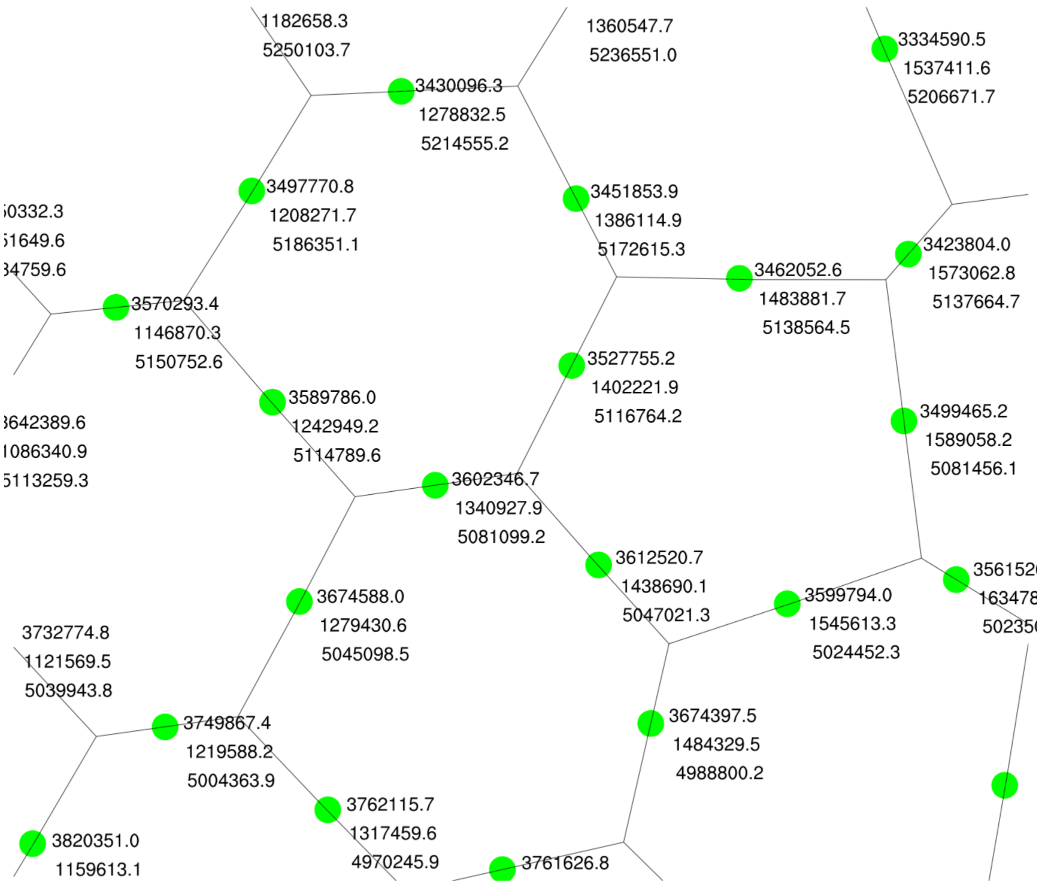
- For planar meshes, coordinates lie in the plane z=0
- For spherical meshes, coordinates lie on the surface of the sphere

For cells: **xCell**, **yCell**, **zCell**

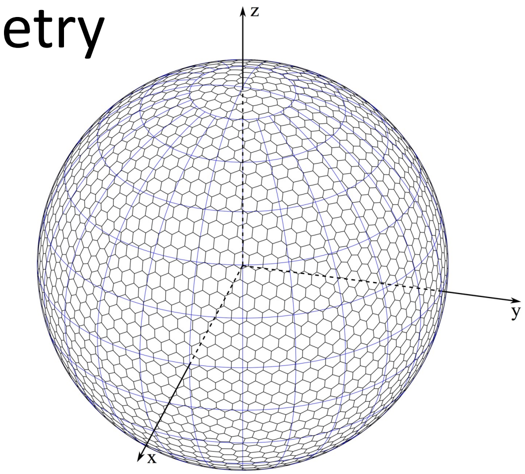
Latitudes and longitudes are computed from Cartesian coordinates as described earlier

- positive x-axis through 0° longitude
- positive y-axis through 90° longitude
- positive z-asix through 90° latitude

Notes about MPAS mesh geometry



Above: Cartesian coordinates for cell locations near (52.9°N lat, 20.8°E lon) in a variable-resolution spherical mesh with radius 6371229 m.



Global Cartesian coordinates are computed for each element

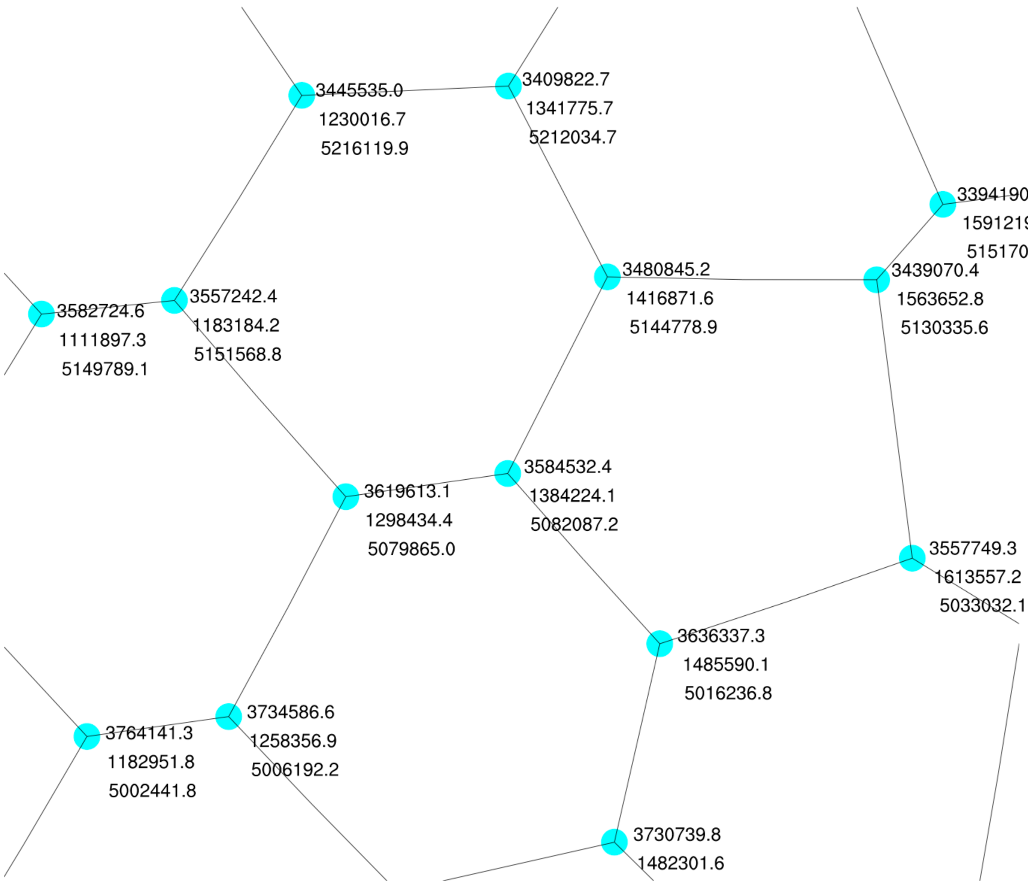
- For planar meshes, coordinates lie in the plane z=0
- For spherical meshes, coordinates lie on the surface of the sphere

For cells: **xEdge**, **yEdge**, **zEdge**

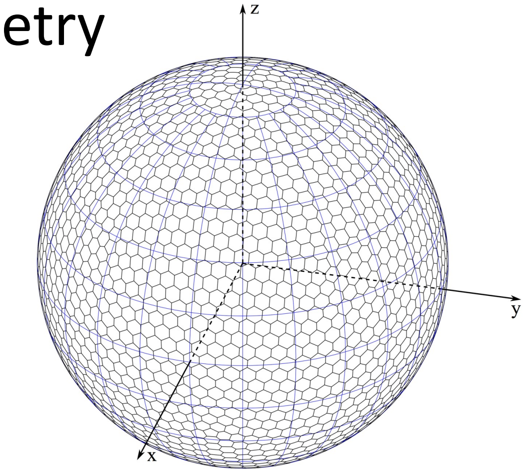
Latitudes and longitudes are computed from Cartesian coordinates as described earlier

- positive x-axis through 0° longitude
- positive y-axis through 90° longitude
- positive z-asix through 90° latitude

Notes about MPAS mesh geometry



Above: Cartesian coordinates for cell locations near (52.9°N lat, 20.8°E lon) in a variable-resolution spherical mesh with radius 6371229 m.



Global Cartesian coordinates are computed for each element

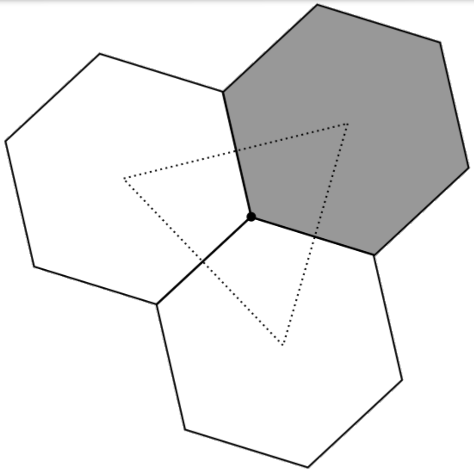
- For planar meshes, coordinates lie in the plane z=0
- For spherical meshes, coordinates lie on the surface of the sphere

For cells: **xVertex**, **yVertex**, **zVertex**

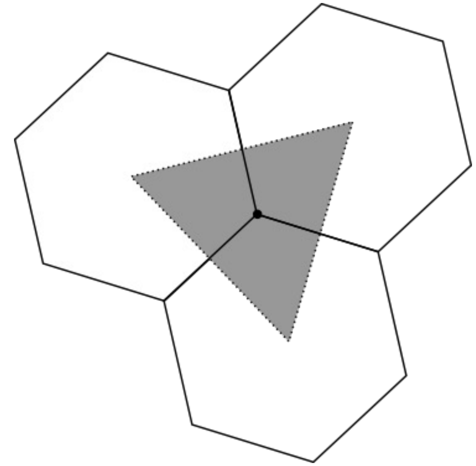
Latitudes and longitudes are computed from Cartesian coordinates as described earlier

- positive x-axis through 0° longitude
- positive y-axis through 90° longitude
- positive z-asix through 90° latitude

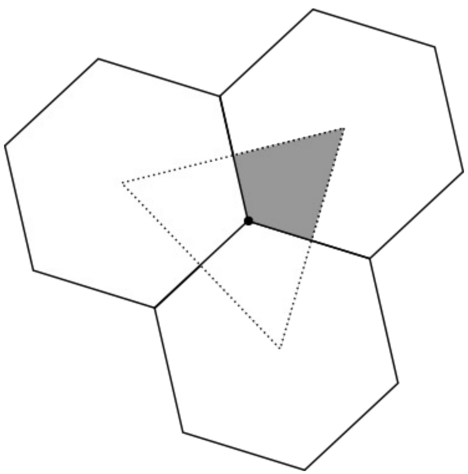
Mesh geometry fields in MPAS



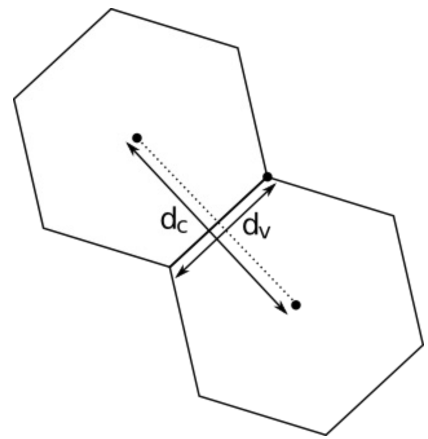
areaCell(nCells) – area of each cell



areaTriangle(nVertices) – area of each dual-grid cell



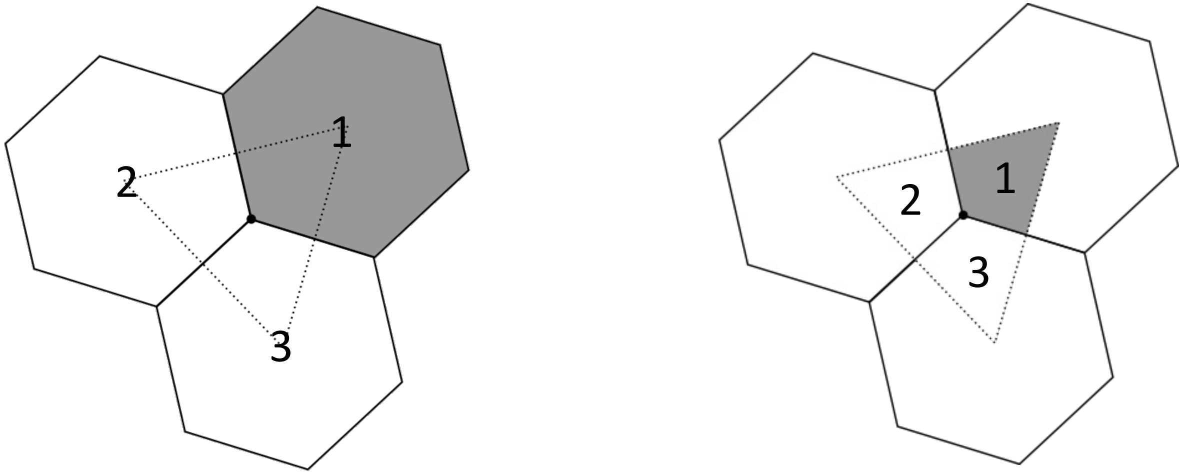
kiteAreasOnVertex(vertexDegree,nVertices) – area of intersection between dual- and primal-mesh cells



dcEdge(nEdges) – distances between cell centers
dvEdge(nEdges) – length of each edge

Indexing of mesh fields, both connectivity and geometry

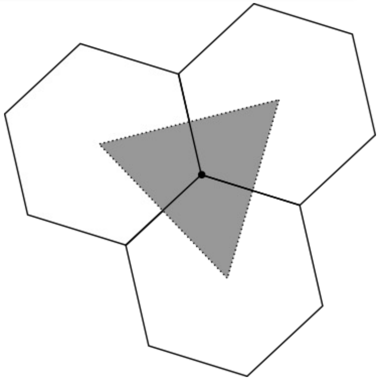
- For *On* arrays (e.g., cellsOnCell), elements are listed in anti-clockwise order
  - Whenever possible, starting points are consistent between indexing arrays (e.g., cellsOnVertex and kiteAreasOnVertex)
  - E.g., the first edgeOnCell separates a given cell from the first cellOnCell
- All indices are 1-based
  - (MPAS is written in Fortran, after all...)

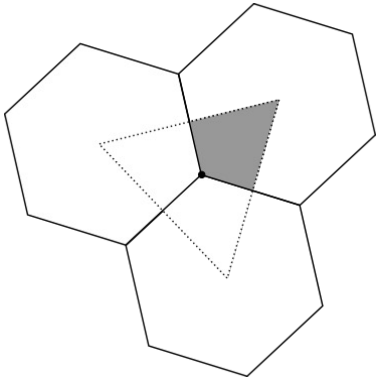


cellsOnVertex(:,V) kiteAreasOnVertex(:,V)

A practical example of mesh fields in use

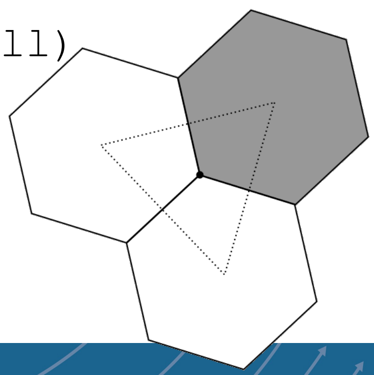
An example of using mesh fields: Averaging a vertex-based field, `vorticity(nVertLevels,nVertices)`, to cells as `vortcell(nVertLevels,nCells)`:





```
vortcell(:,:) = 0.0
do iVtx = 1, nVertices
do j = 1, vertexDegree
    iCell = cellsOnVertex(j,iVtx)
    vortcell(:,iCell) = vortcell(:,iCell) +
          kiteAreasOnVertex(j,iVtx) * vorticity(:,iVtx)
end do
end do

do iCell = 1, nCells
    vortcell(:,iCell) = vortcell(:,iCell) / areaCell(iCell)
end do
```



What do MPAS meshes look like in netCDF files?

When stored in netCDF files ("*grid.nc*"), MPAS meshes have at least the following dimensions:

```
dimensions:
    nCells = 40962 ;
    nEdges = 122880 ;
    nVertices = 81920 ;
    maxEdges = 10 ;
    maxEdges2 = 20 ;
    TWO = 2 ;
    vertexDegree = 3 ;
```

The number of cells, edges, and vertices in the mesh.

For global, spherical meshes:
nVertices = 2 * (nCells - 2)
nEdges = 3 * (nCells - 2)

For *doubly-periodic* planar meshes:
nEdges = nCells + nVertices

For *limited-area* meshes:
nEdges + 1 = nCells + nVertices

What do MPAS meshes look like in netCDF files?

When stored in netCDF files (“*grid.nc*”), MPAS meshes have at least the following dimensions:

```
dimensions:
    nCells = 40962 ;
    nEdges = 122880 ;
    nVertices = 81920 ;
    maxEdges = 10 ;
    maxEdges2 = 20 ;
    TWO = 2 ;
    vertexDegree = 3 ;
```

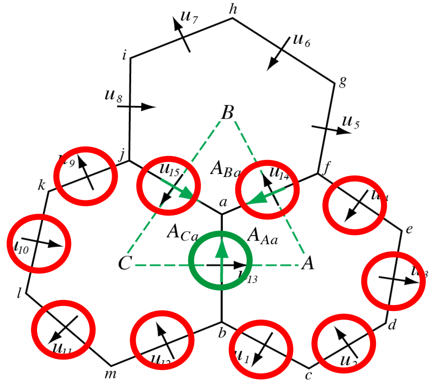
The maximum number of faces (edges) any cell can have; equivalent to the maximum number of cell neighbors or vertices that a cell can have.

What do MPAS meshes look like in netCDF files?

When stored in netCDF files (“*grid.nc*”), MPAS meshes have at least the following dimensions:

```
dimensions:
   nCells = 40962 ;
   nEdges = 122880 ;
   nVertices = 81920 ;
   maxEdges = 10 ;
   maxEdges2 = 20 ;
   TWO = 2 ;
   vertexDegree = 3 ;
```

The maximum number of edges that participate in the reconstruction of tangential velocities at cell faces (edges).



What do MPAS meshes look like in netCDF files?

When stored in netCDF files ("*grid.nc*"), MPAS meshes have at least the following dimensions:

```
dimensions:
   nCells = 40962 ;
   nEdges = 122880 ;
   nVertices = 81920 ;
   maxEdges = 10 ;
   maxEdges2 = 20 ;
   TWO = 2 ;
   vertexDegree = 3 ;
```

Always 2 (every dimension must have a name in netCDF). Used for, e.g., the number of vertices forming the endpoints of edges and the number of cells separated by an edge.

What do MPAS meshes look like in netCDF files?

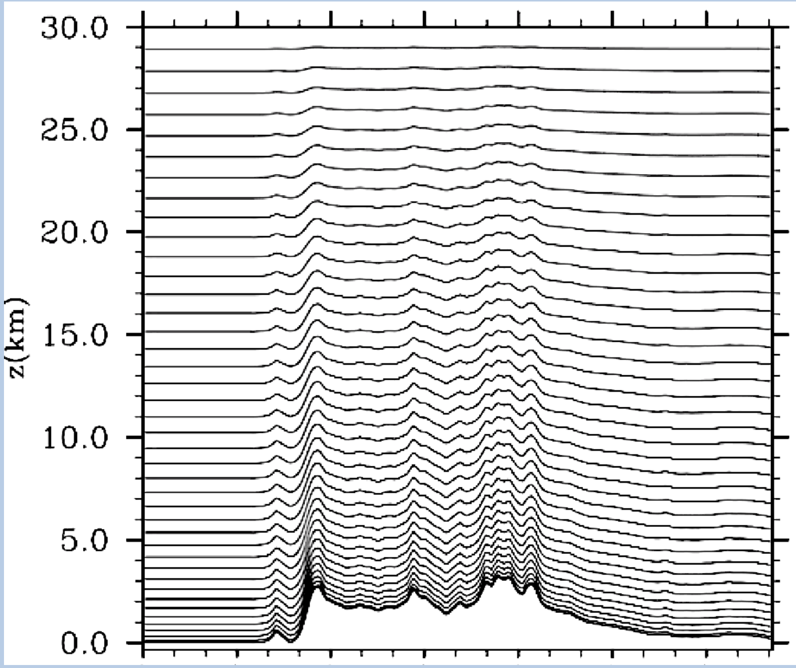
When stored in netCDF files ("*grid.nc*"), MPAS meshes have at least the following dimensions:

```
dimensions:
    nCells = 40962 ;
    nEdges = 122880 ;
    nVertices = 81920 ;
    maxEdges = 10 ;
    maxEdges2 = 20 ;
    TWO = 2 ;
    vertexDegree = 3 ;
```

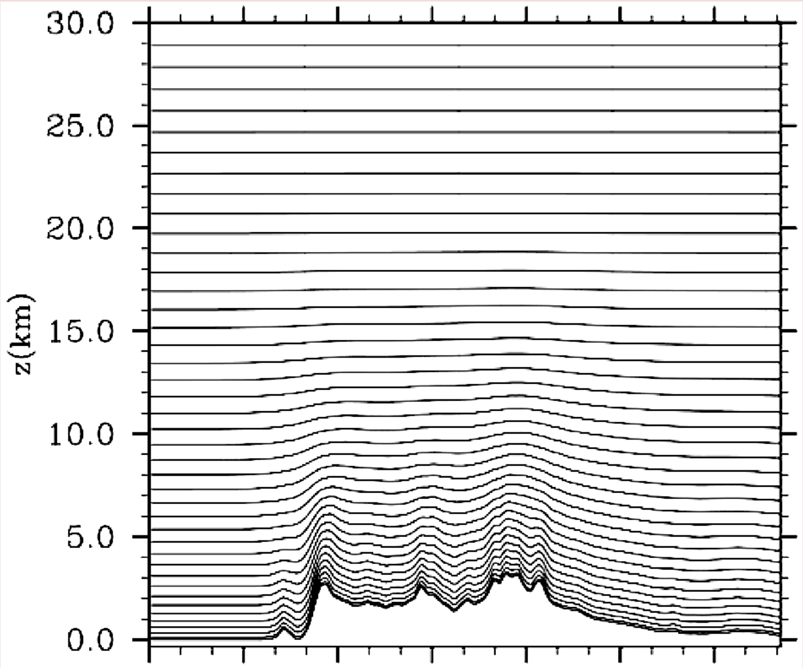
The number of cells/edges that meet at each vertex.

In principle, quadrilateral meshes could be represented by setting vertexDegree = 4

What about the vertical grid?



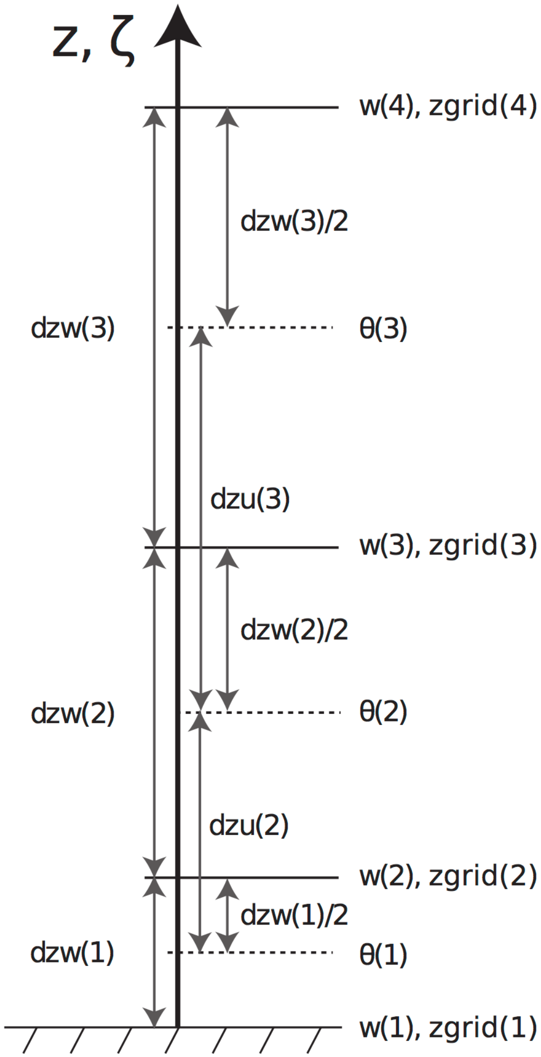
WRF
Pressure-based terrain-following sigma vertical coordinate



MPAS
Height-based hybrid smoothed terrain-following vertical coordinate

- Improved numerical accuracy

Vertical grid



The MPAS-Atmosphere vertical grid is also staggered:

- vertical velocities on *w* levels
- all other fields on Θ levels

`zgrid` gives geometric height at w levels

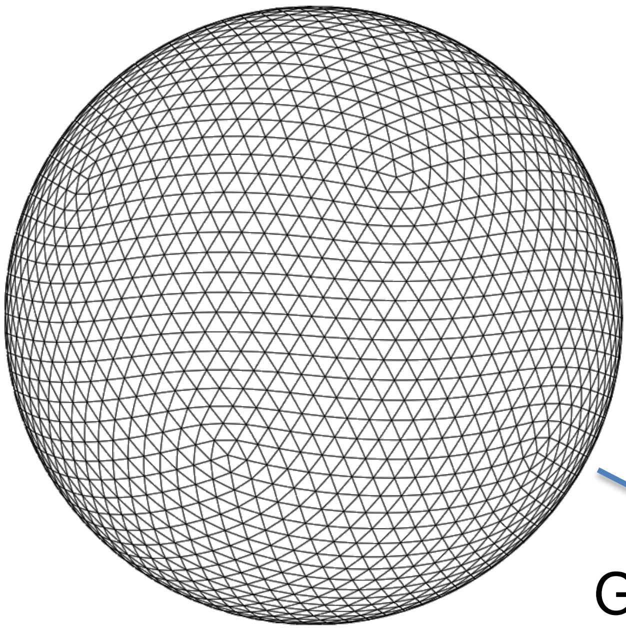
Θ levels lie at the midpoints of bracketing w levels

To vertically interpolate field *F* from Θ levels to w levels:

fzp(k) = 0.5 * dzw(k) / dzu(k)
fzm(k) = 0.5 * dzw(k-1) / dzu(k)

$F_w(k) = fzm(k) * F_\Theta(k) + fzp(k) * F_\Theta(k-1)$

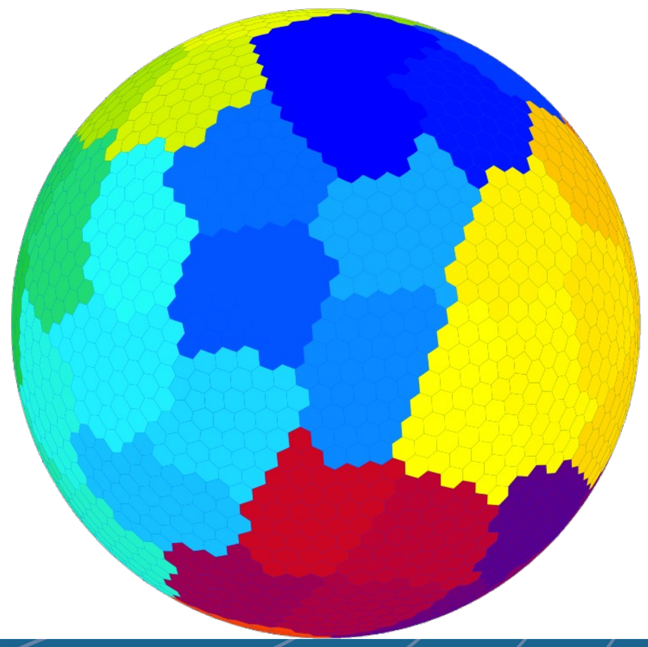
Meshes partitioning for parallelization



•The *dual* mesh of a Voronoi tessellation is a Delaunay triangulation – essentially the connectivity graph of the cells

•Parallel decomposition of an MPAS mesh then becomes a graph partitioning problem: ***equally distribute nodes among partitions (give each process equal work) while minimizing the edge cut (minimizing parallel communication)***

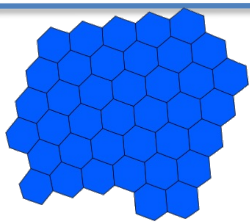
Graph partitioning



We use the Metis package for parallel graph decomposition

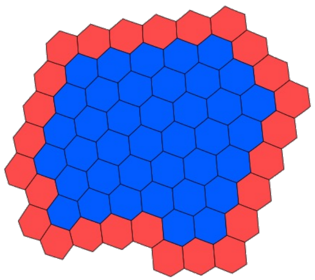
- Currently done as a pre-processing step, but could be done “on-line”
- Fortunately, Metis runs quickly, and a partitioning into *n* pieces only needs to be done once for a given mesh

Meshes partitioning for parallelization

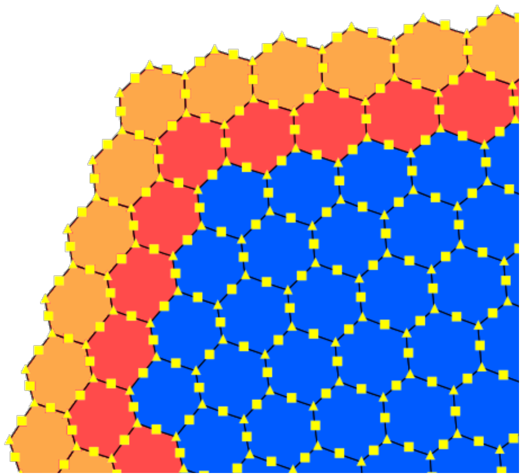


Block of cells owned by a process

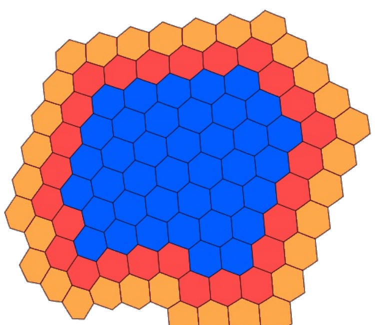
Given an assignment of cells to a process, any number of layers of halo (ghost) cells may be added



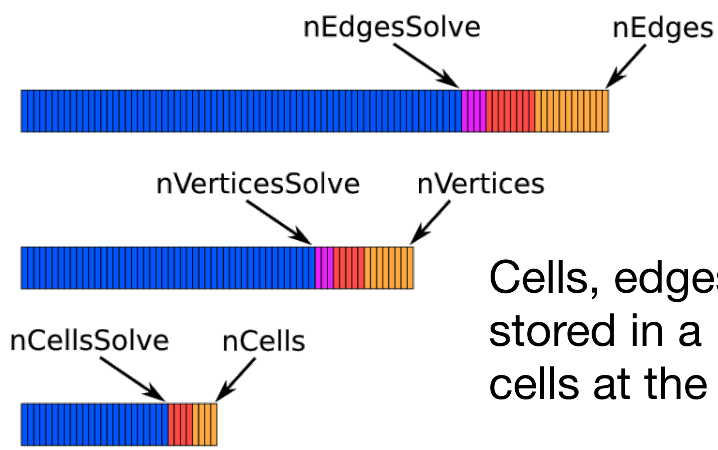
Block plus one layer of halo/ghost cells



With a complete list of cells stored in a block, adjacent edge and vertex locations can be found; we apply a simple rule to determine ownership of edges and vertices adjacent to real cells in different blocks



Block plus two layers of halo/ghost cells

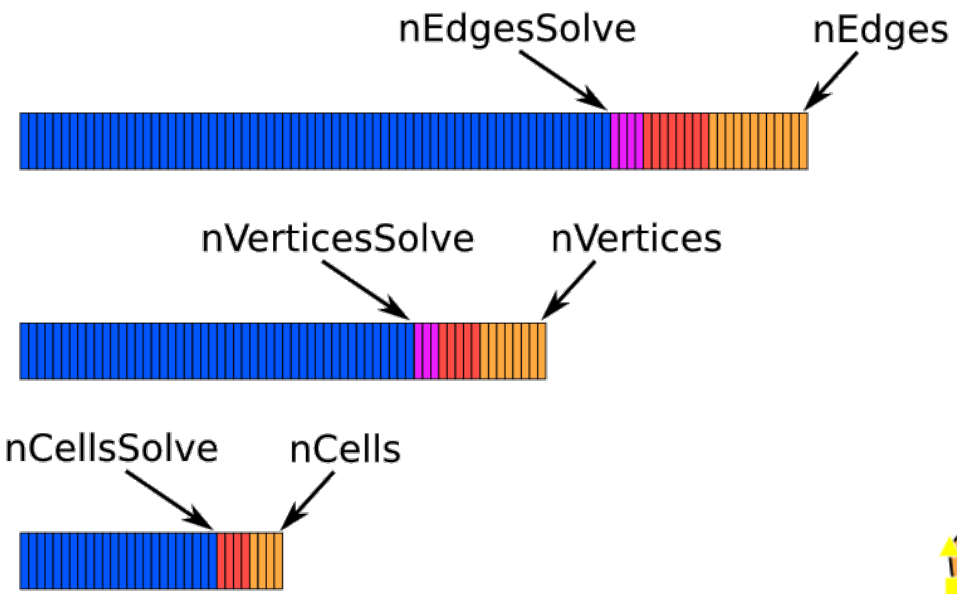


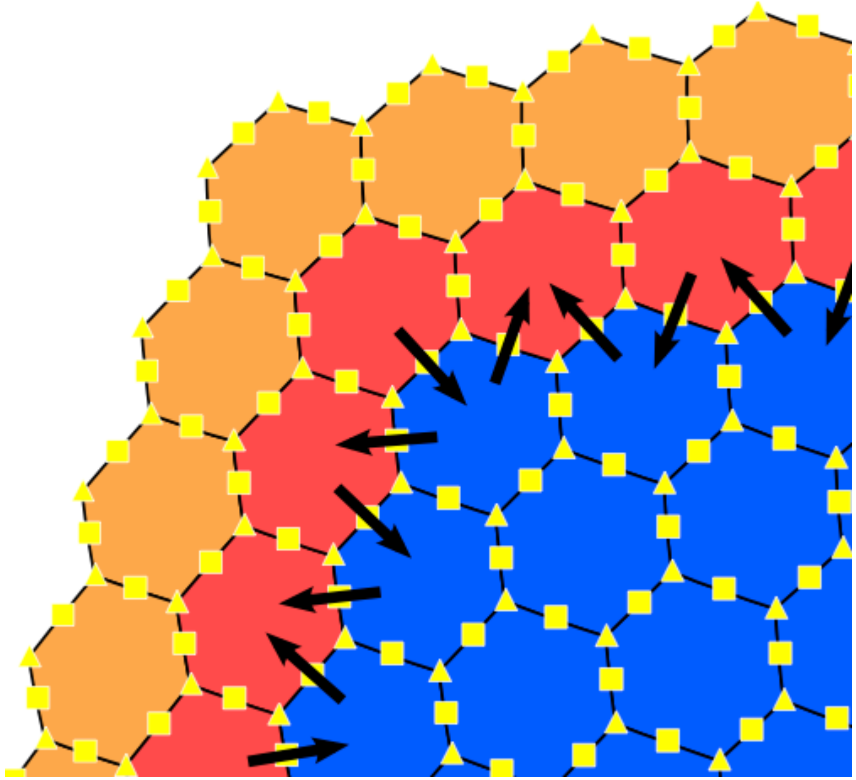
Cells, edges, and vertices are stored in a 1-d array, with halo cells at the end of the array

Meshes partitioning for parallelization

An edge *E* is an owned edge **iff** cellsOnEdge(1,*E*) is an owned cell

A vertex *V* is an owned vertex **iff** cellsOnVertex(1,*V*) is an owned cell





For *n* layers of ghost cells, we have *n+1* layers of ghost edges and ghost vertices.