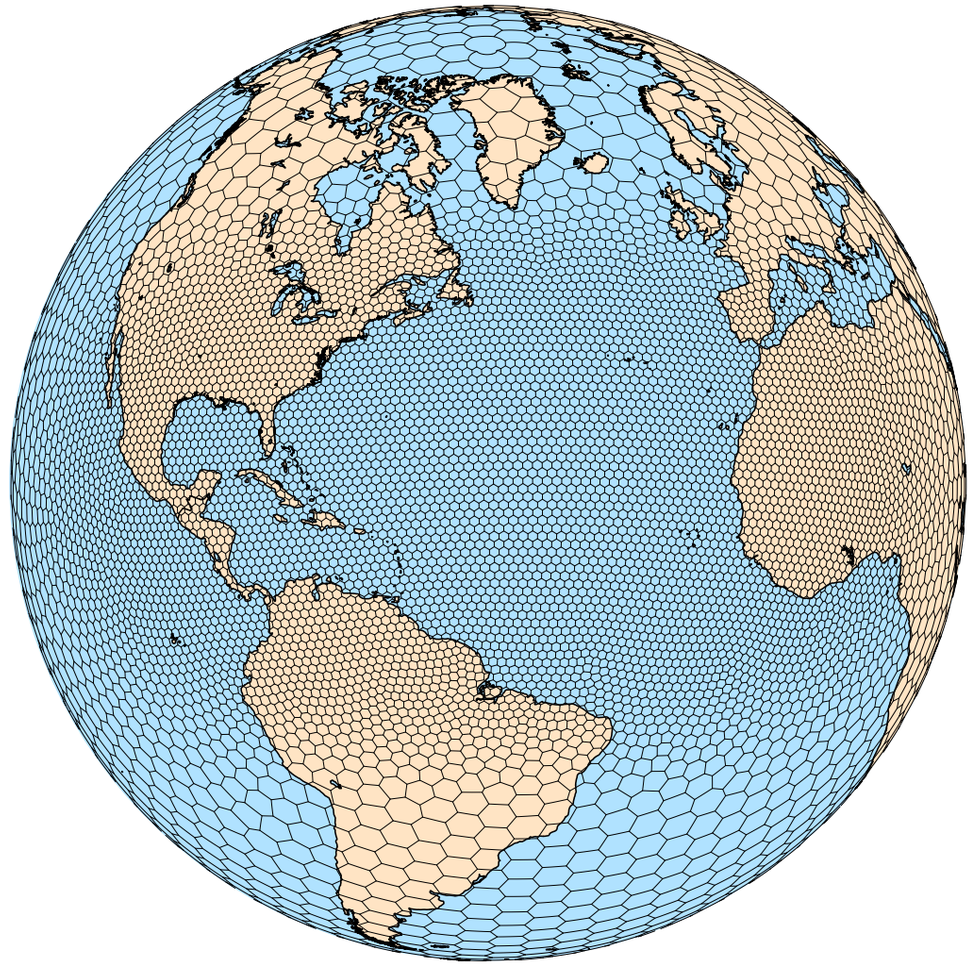
MPAS

Model for Prediction Across Scales



- Overview
- *Mesh description*
- Atmospheric solver, physics
- Registry, installation, running MPAS
- MPAS support, future evolution

Los Alamos NATIONAL LABORATORY EST. 1943

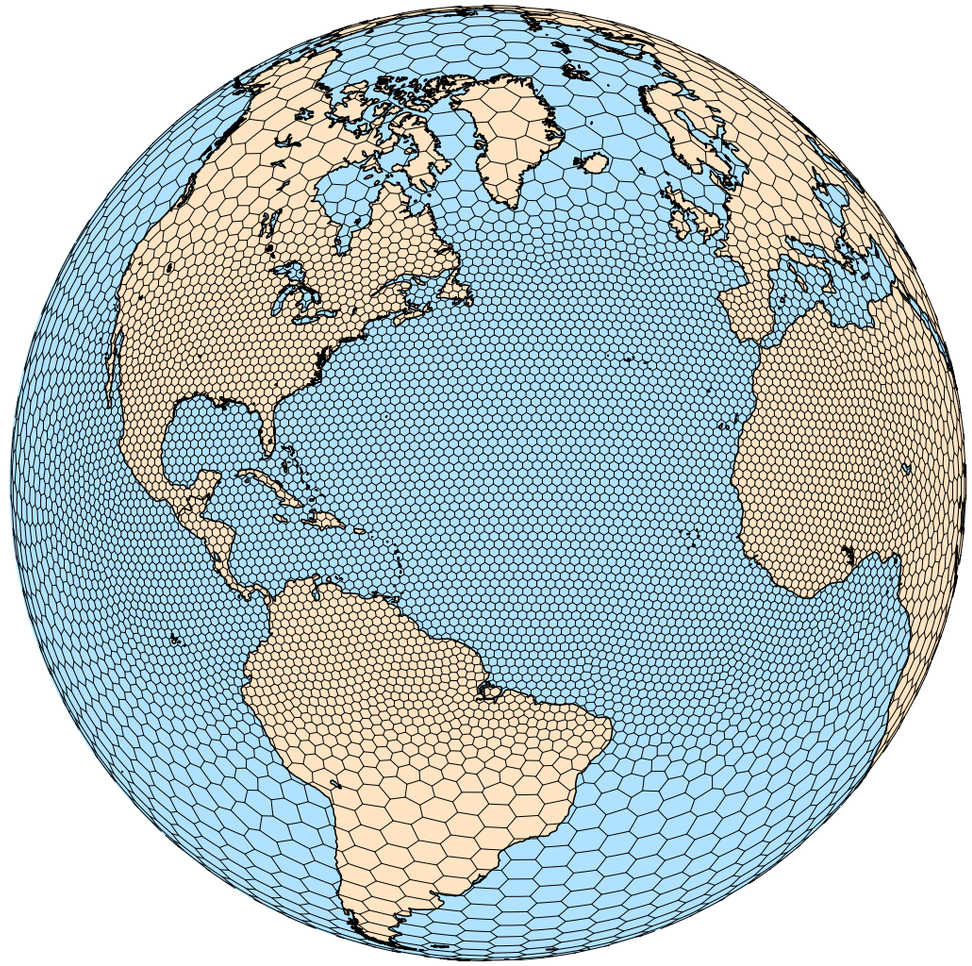
U.S. DEPARTMENT OF ENERGY | Office of Science

NSF

NESL

MPAS

Model for Prediction Across Scales



- Overview
- *Mesh description*
  - How do we generate meshes for MPAS?
  - Which mesh geometries are supported in MPAS-Atmosphere?
  - How are cells indexed; and how does one “navigate” through a mesh?
  - How is the mesh decomposed for parallel processing?

Los Alamos NATIONAL LABORATORY EST. 1943

U.S. DEPARTMENT OF ENERGY | Office of Science

NSF

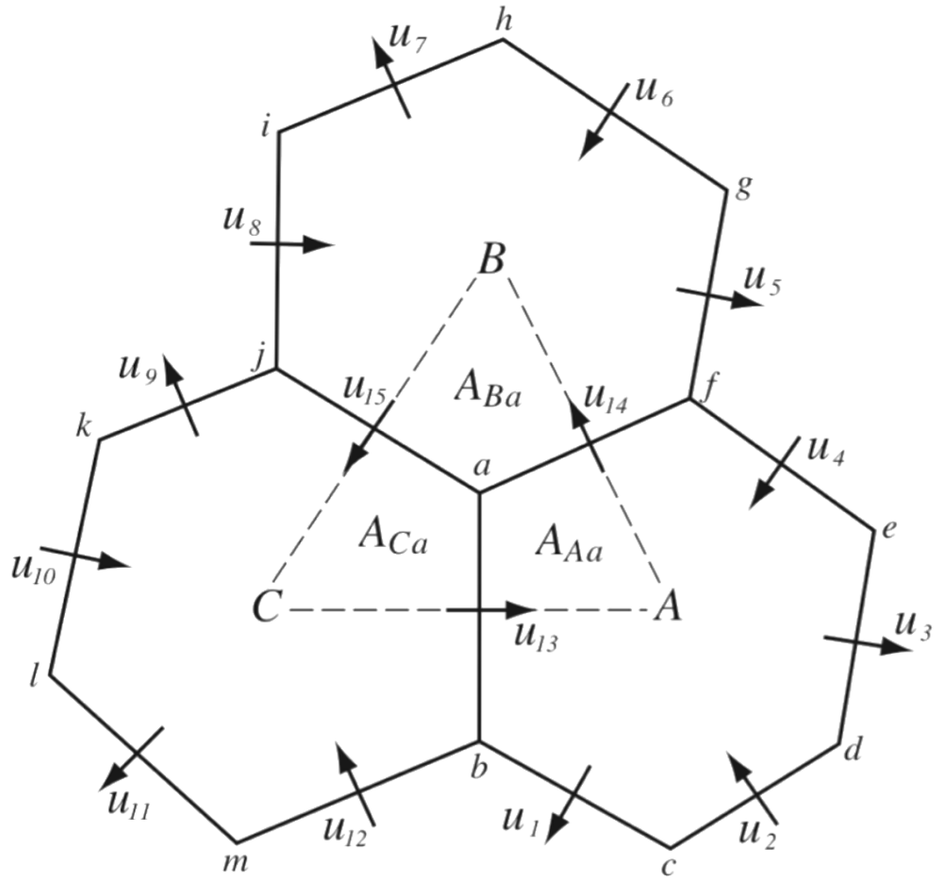
NESL

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 $\boldsymbol{x}_j$

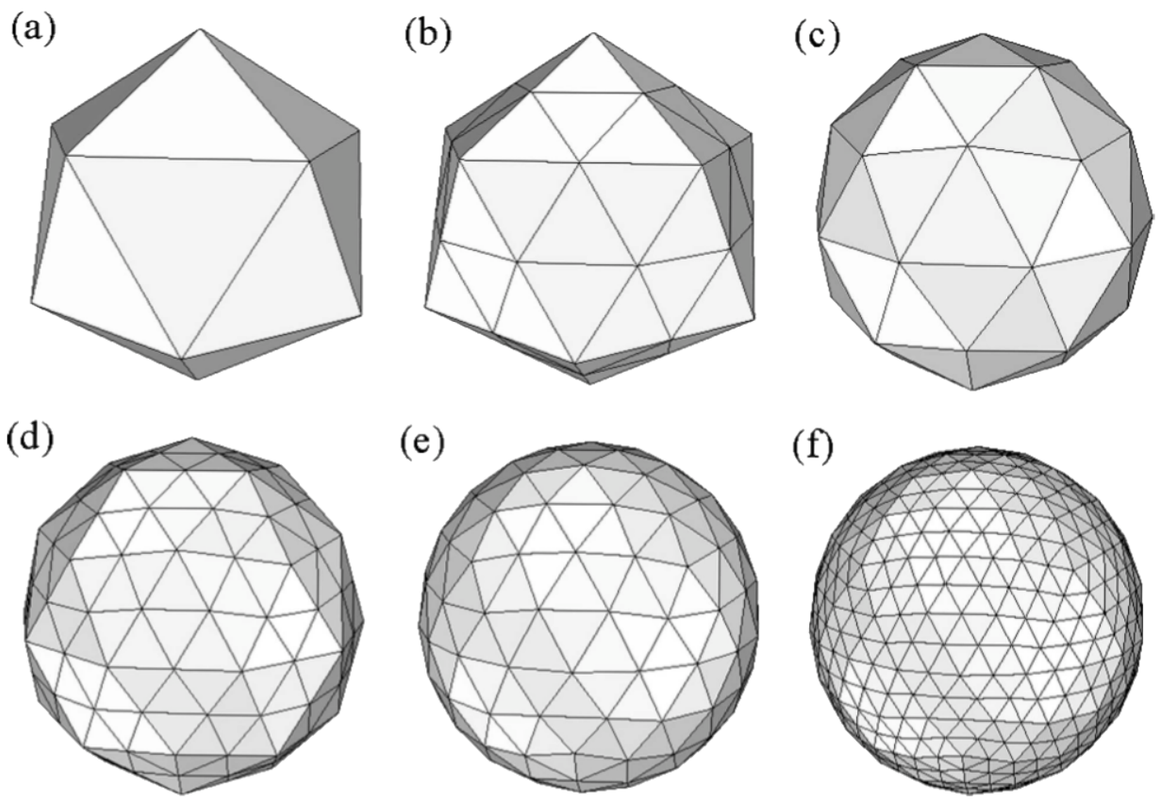
Centroidal := the generating point for each Voronoi cell is also the mass centroid of that cell (**w.r.t. some density function**)



Prognostic velocities are velocities normal to cell faces ("edges") at the point where the edge intersects the arc joining cells on either side

For quasi-uniform Voronoi tessellations, we can employ successive subdivision of the icosahedron

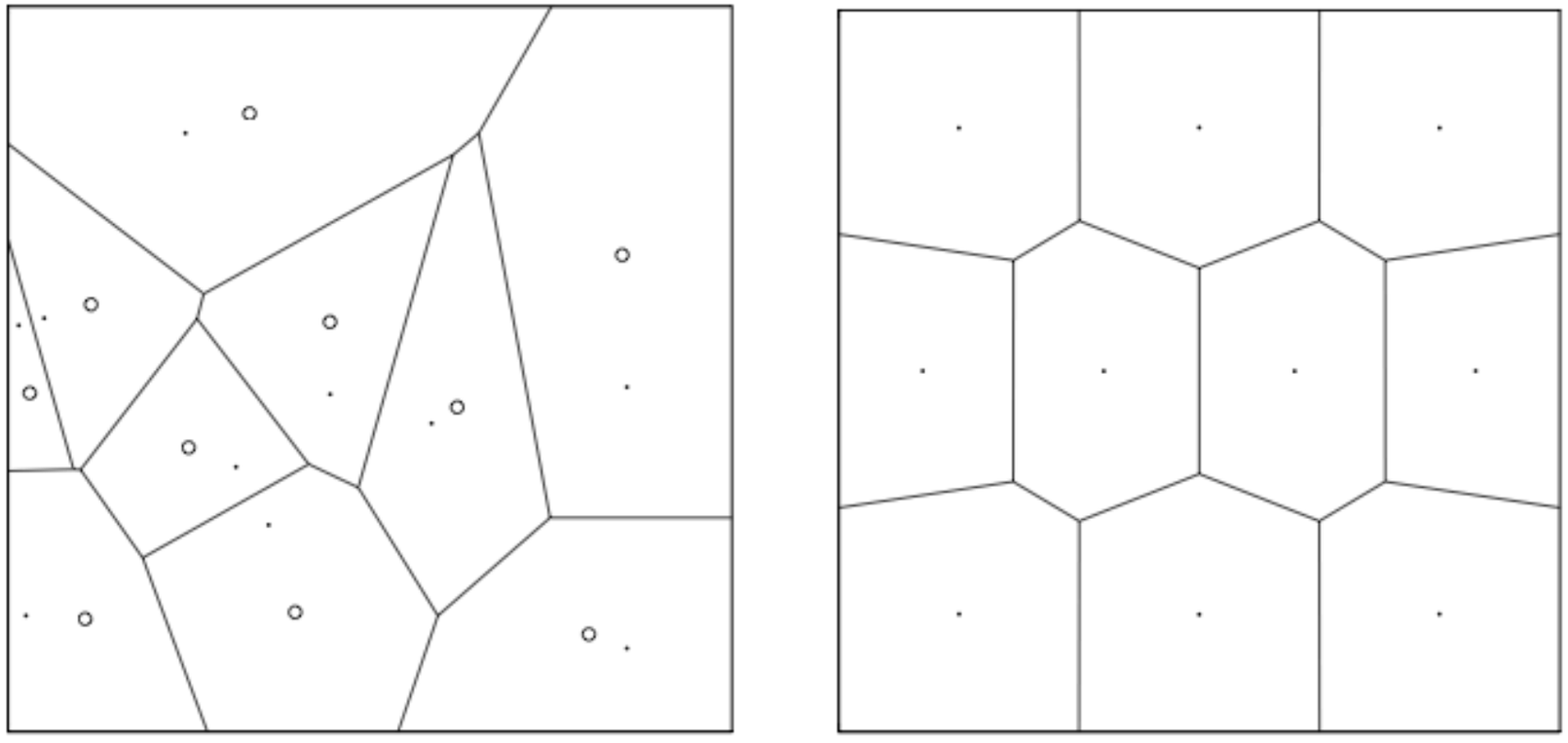
- The vertices of these triangular meshes can be used as the generating points for a spherical Voronoi tessellation
- To create a spherical ***centroidal*** Voronoi tessellation, some iteration is required



From Lipscomb and Ringler (2005)

Given an initial set of generating points, Lloyd's method may be used to arrive at a CVT:

1. Begin with any set of initial points ("generating points")
2. Construct a Voronoi diagram for the point set
3. Locate the mass centroid of each Voronoi cell
4. Move each generating point to the mass centroid of its Voronoi cell
5. Repeat 2-4 to convergence

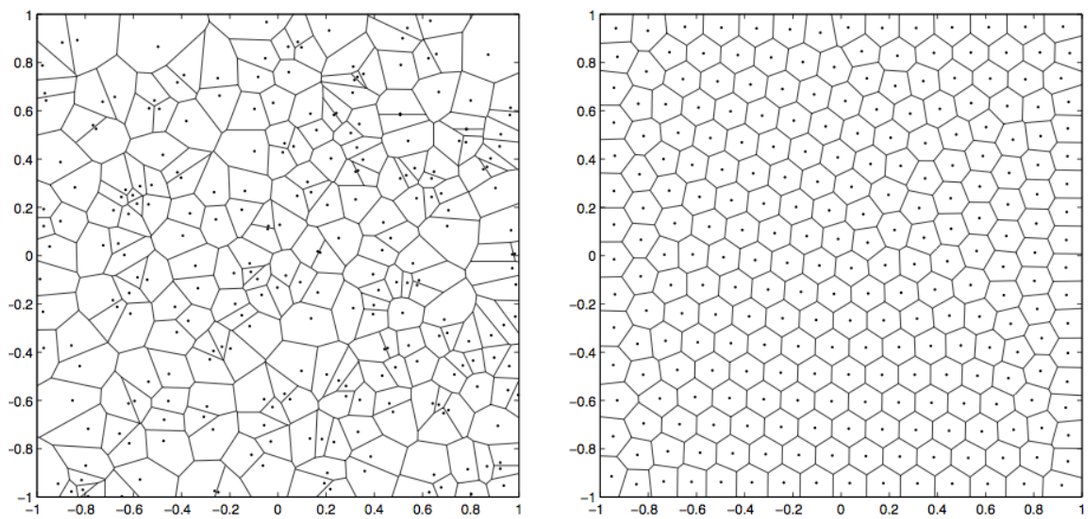


From Du et al. (1999)

- MacQueen's method, an randomized alternative to Lloyd's method may also be used; no Voronoi diagrams need to be constructed, but convergence is generally much slower

Lloyd's method can be viewed as the minimization of an energy functional; in the plane, it can be shown that hexagonal Voronoi cells provide the minimum energy configuration *for constant density*

To create regions of grid refinement, we simply define a non-uniform density function over the domain, and use this when computing the mass centroids of Voronoi cells in Lloyd's method



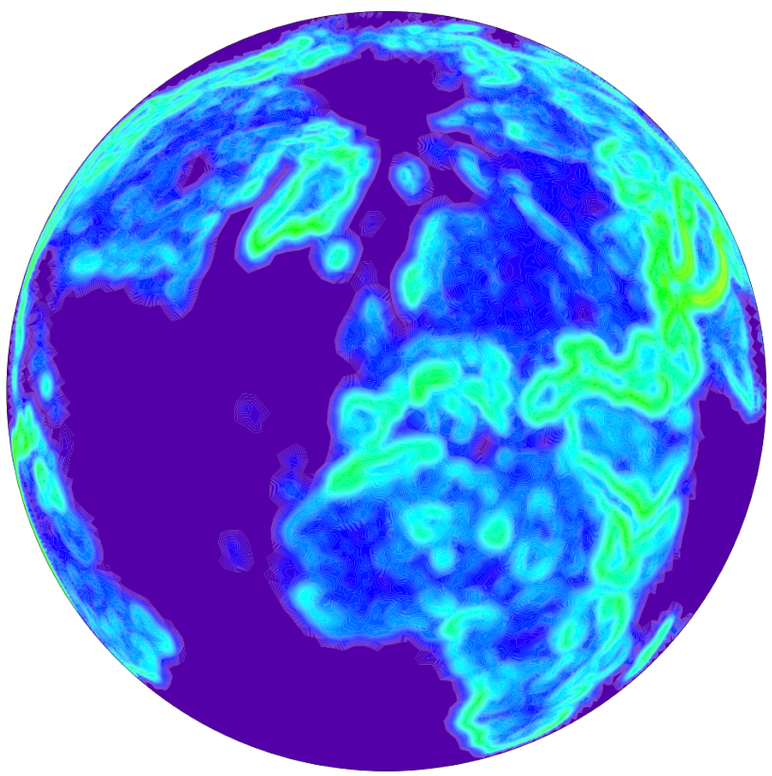
From Du et al. (1999)

For a density function ρ(***x***) > 0, it is conjectured (Ju et al. (2010)) that, as the number of Voronoi cells increases, the diameters, *h*, of Voronoi cells are related by

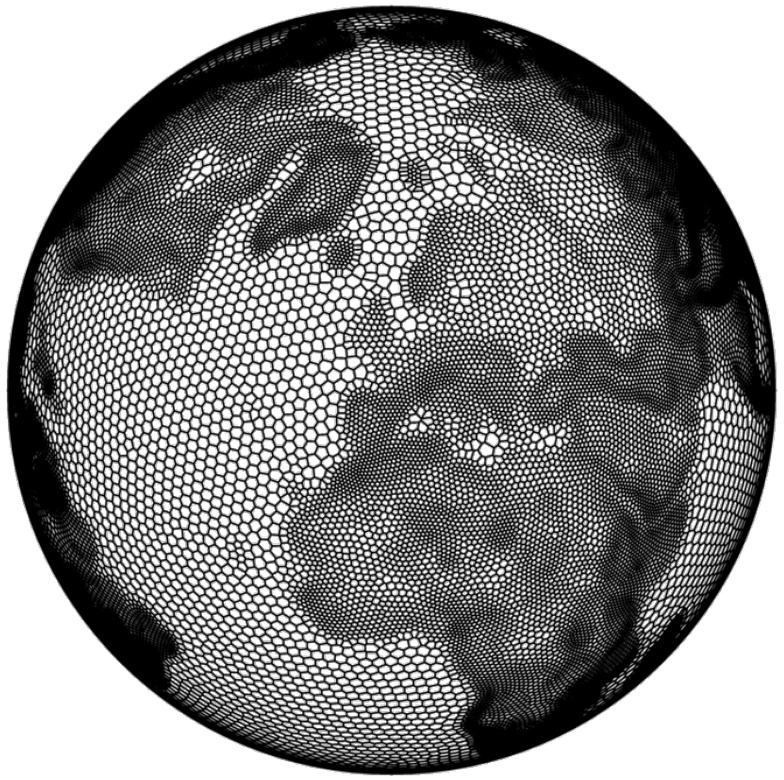
$$\frac{h_i}{h_j} \approx \left(\frac{\rho(\mathbf{x}_j)}{\rho(\mathbf{x}_i)} \right)^{1/(d'+2)}$$

where *d'* is 2.

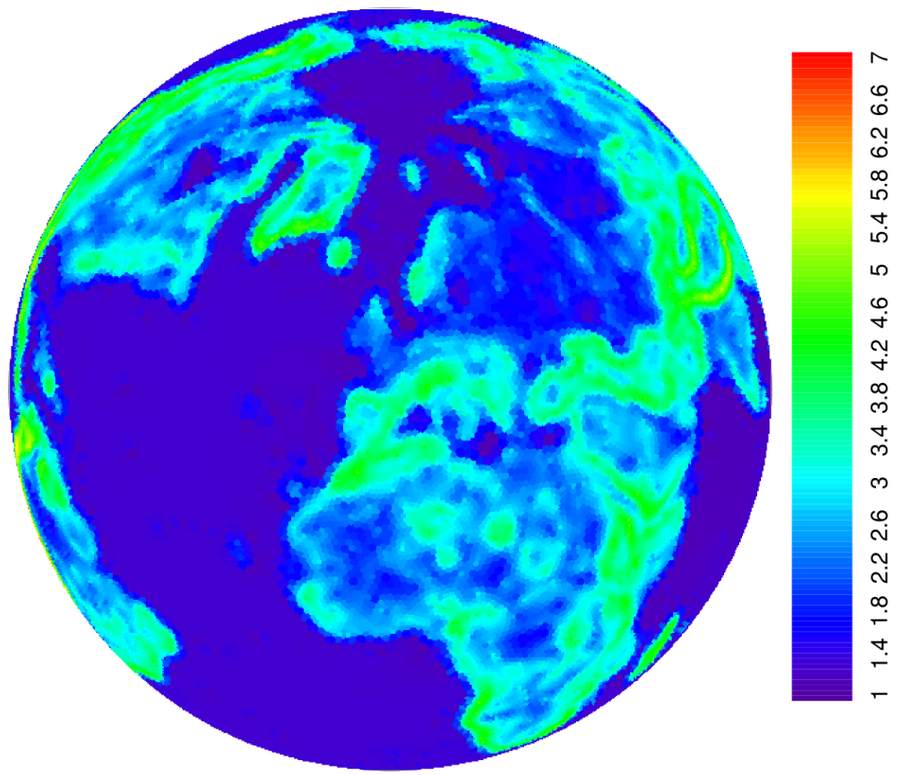
Define ρ(***x***) as a function of the magnitude of the topography gradient, for example:



Fourth root of density function



Resulting SCVT



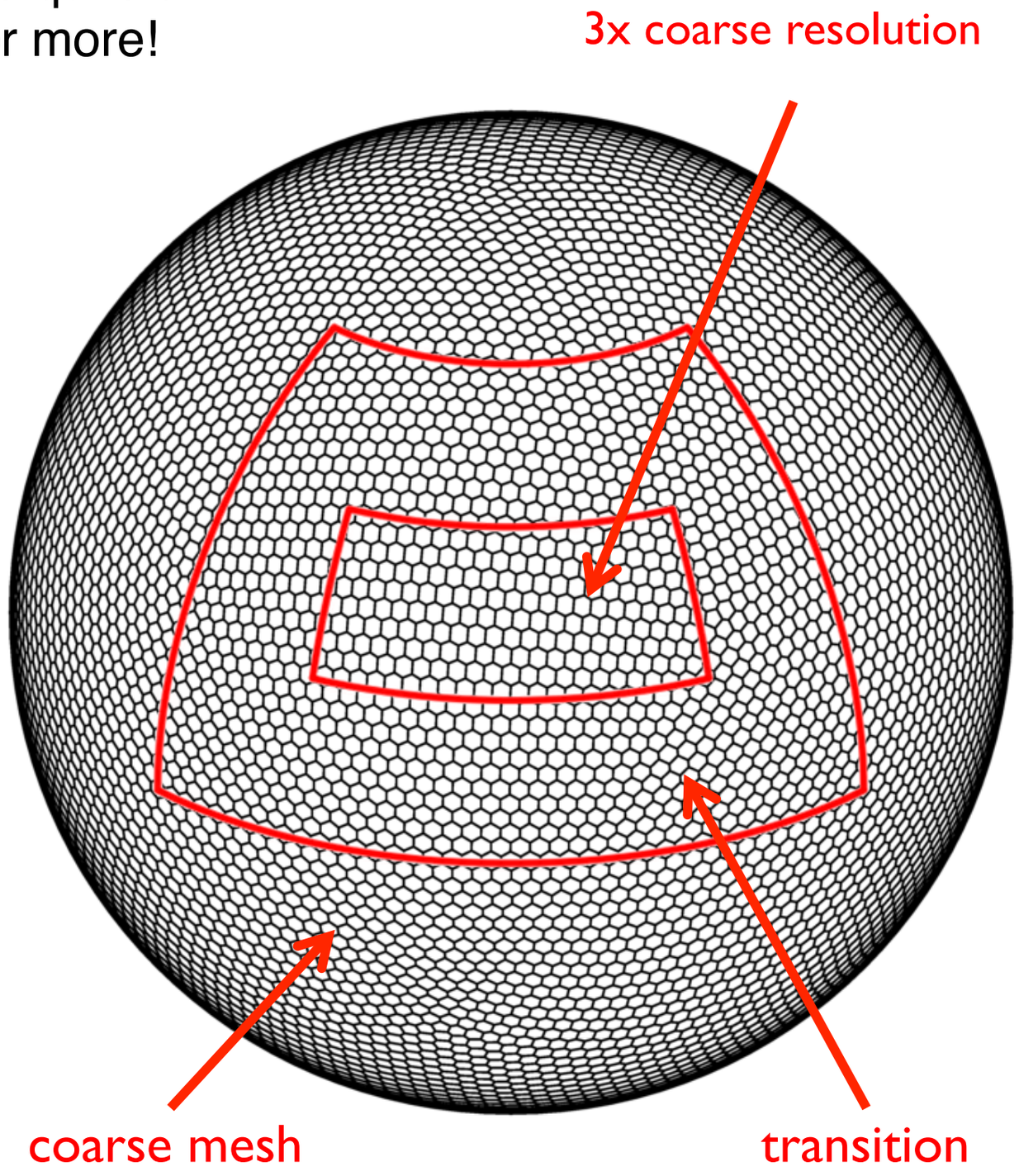
Normalized inverse mean distance of cell centroid from each of its neighboring centroids

Evidence from initial testing of the MPAS non-hydrostatic atmosphere code on multi-resolution meshes on the Cartesian plane suggests that smoother refinement (i.e., less abrupt transitions) produces better results

Meshes with more than $O(10^5)$ cells take quite a while to generate – around a month or more!

Typical procedure:

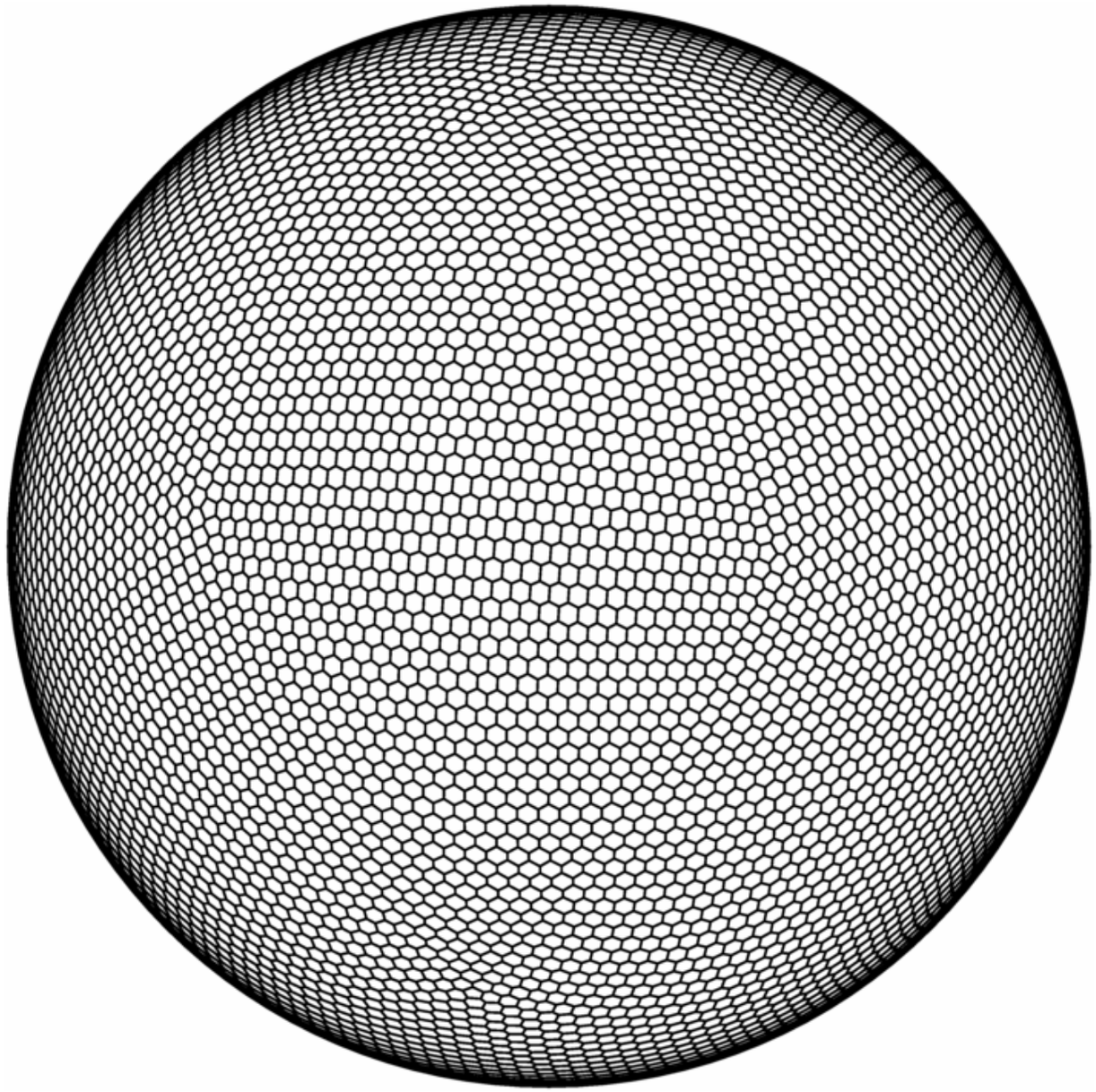
1. Decide on size/shape of refinement region, mesh resolution
2. Estimate the total number of cells needed in the mesh
3. Work backwards from final number of cells to $O(100)$ cells
   - Coarsening a mesh by a factor of two gives $(n+6)/4$ cells
4. Iterate to convergence, bisect the mesh, repeat



Meshes with more than $O(10^5)$ cells take quite a while to generate – around a month or more!

Typical procedure:

1. Decide on size/shape of refinement region, mesh resolution
2. Estimate the total number of cells needed in the mesh
3. Work backwards from final number of cells to $O(100)$ cells
   - Coarsening a mesh by a factor of two gives $(n+6)/4$ cells
4. Iterate to convergence, bisect the mesh, repeat

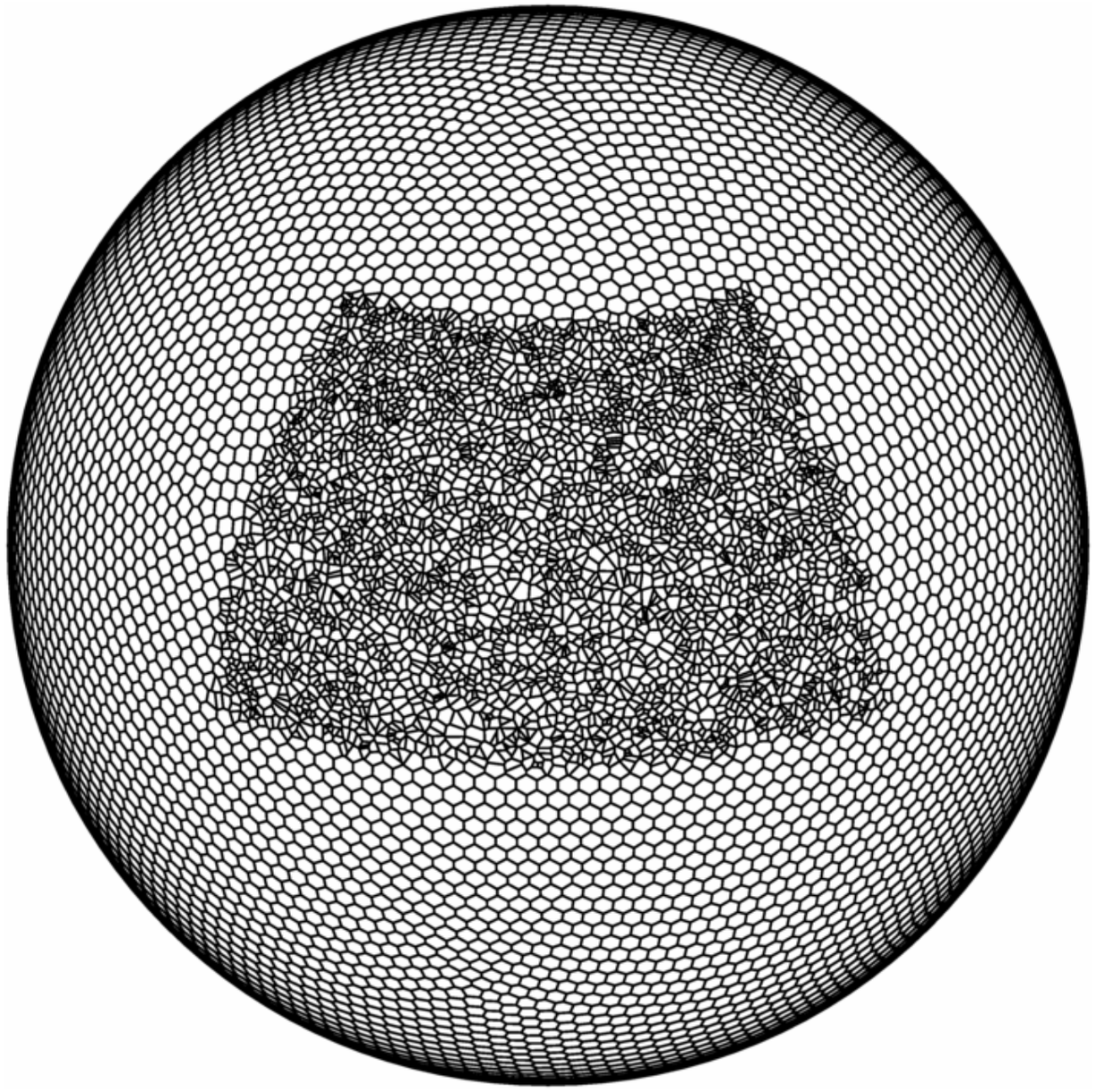


[Above: Animation of Lloyd iteration]

An alternative approach is to add refinement cells in the refinement region, and hold the rest of the mesh fixed

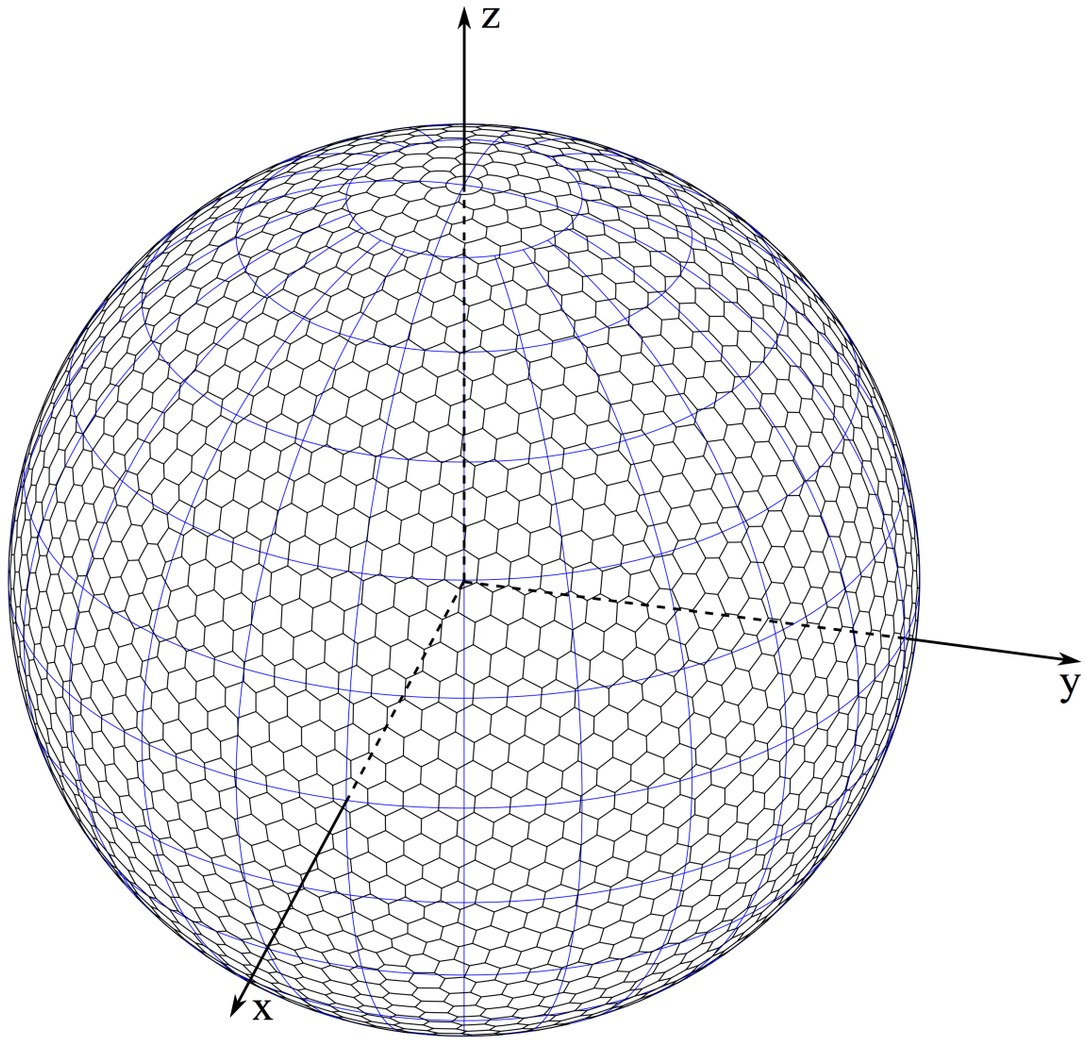
Simple Monte-Carlo method to estimate how many points (nCells) are needed to achieve a specified absolute resolution, given a density function:

ds_{min} = min. desired grid distance in the mesh
N_s = desired # samples
$A_s = 4\pi R^2 / N_s$
for 1 to N_s
  p = random point on sphere
  r = density(p)
  $ds = ds_{min} / r^{0.25}$
  $nCells = nCells + A_s /$ hexagon_area(ds)
End



[Above: Animation of Lloyd iteration]

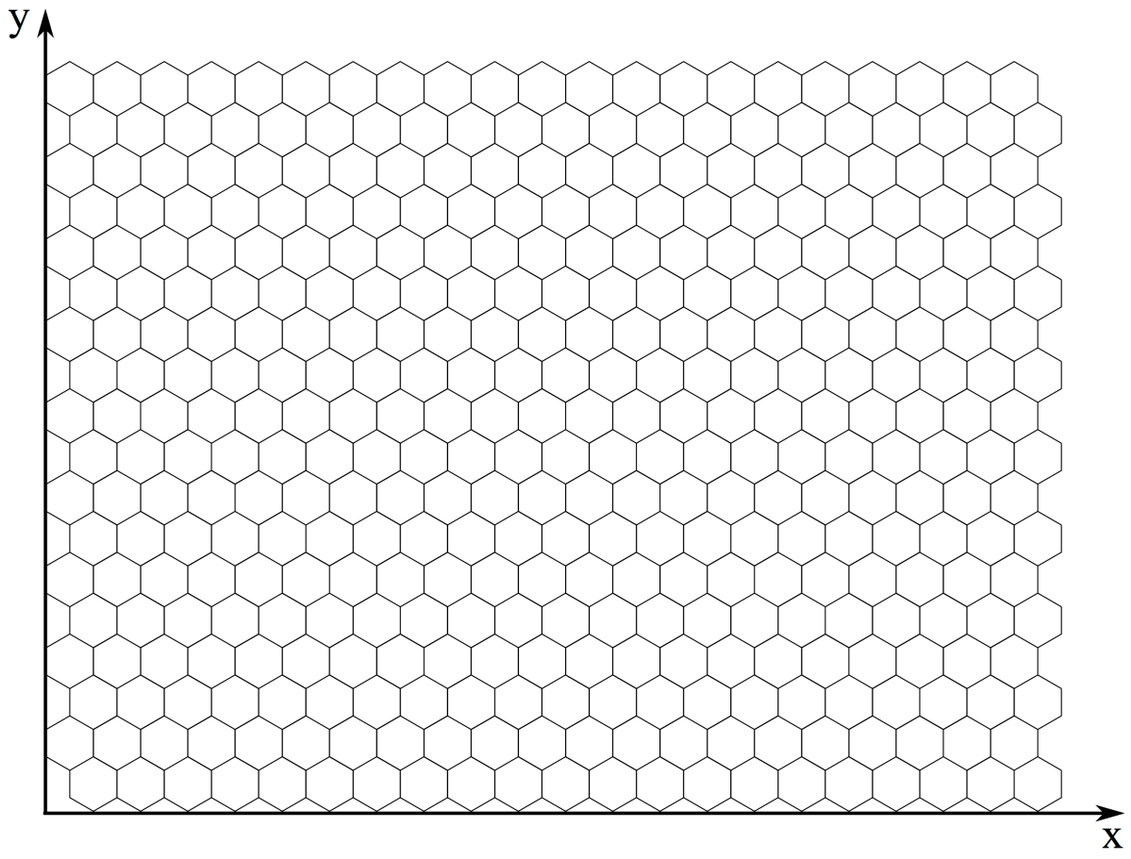
MPAS-Atmosphere supports meshes in two geometries:



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

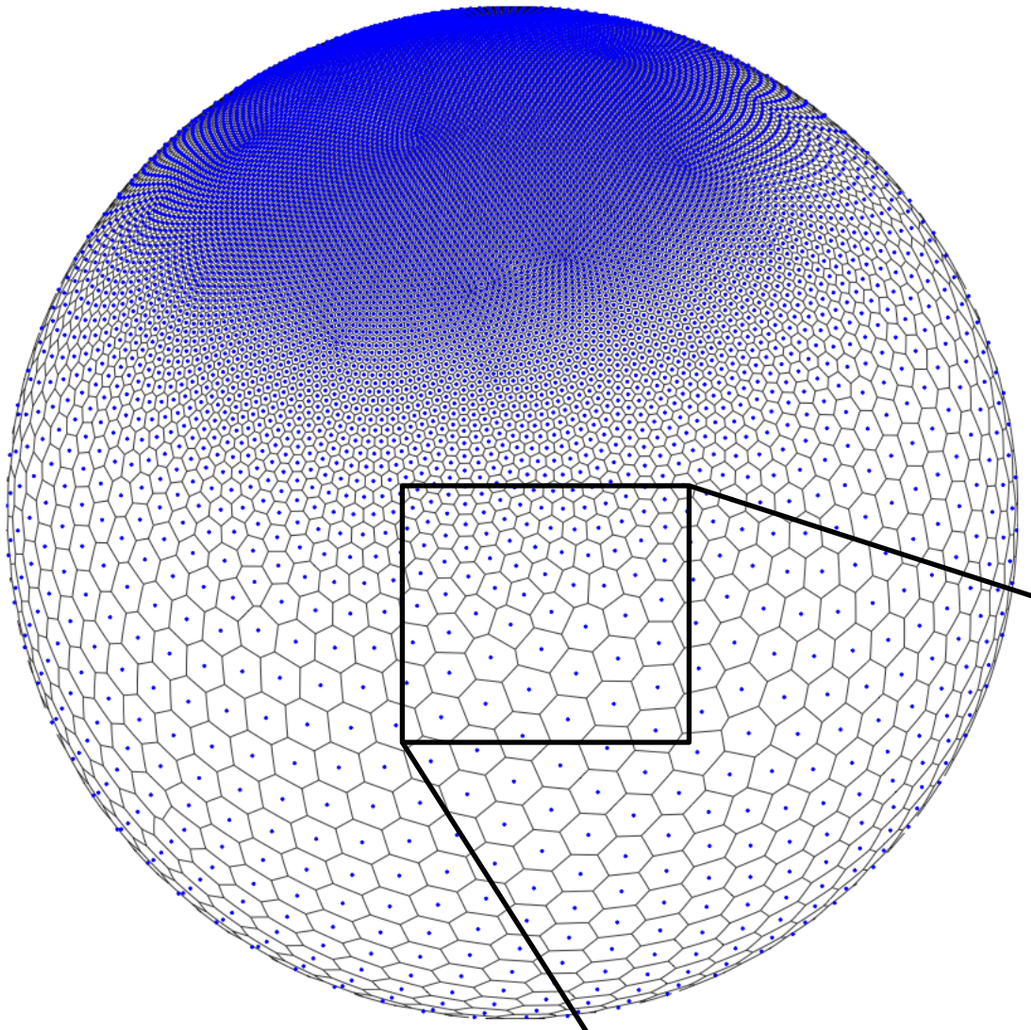
$$x = r\cos(\lambda)\cos(\phi)$$
$$y = r\sin(\lambda)\cos(\phi)$$
$$z = r\sin(\phi)$$

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



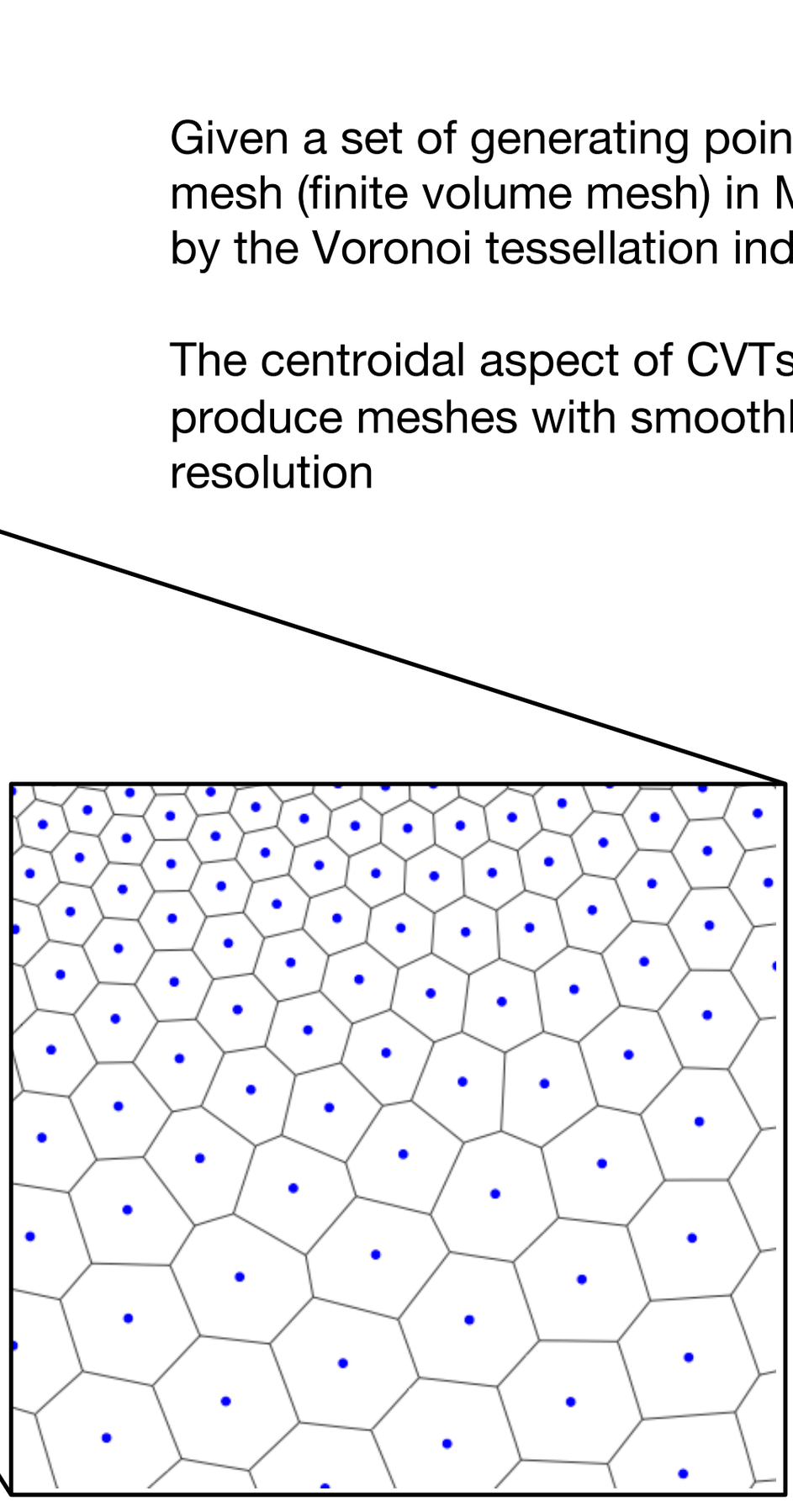
In the Cartesian plane: all distances and areas are computed in Euclidean geometry.

In the plane, only doubly periodic boundaries are currently supported.

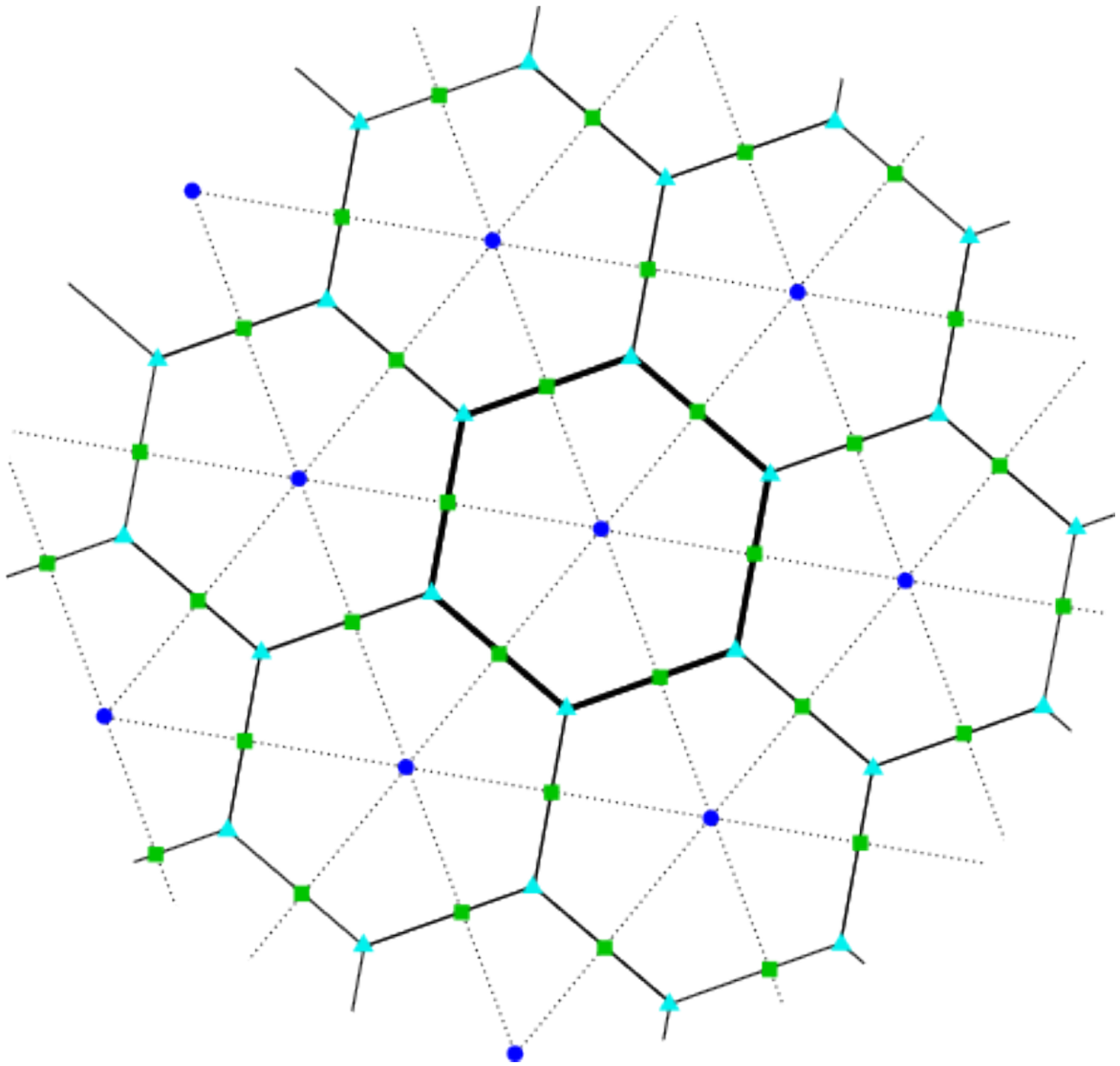


Given a set of generating points, the primal mesh (finite volume mesh) in MPAS is defined by the Voronoi tessellation induced by this set

The centroidal aspect of CVTs is used to produce meshes with smoothly-varying resolution



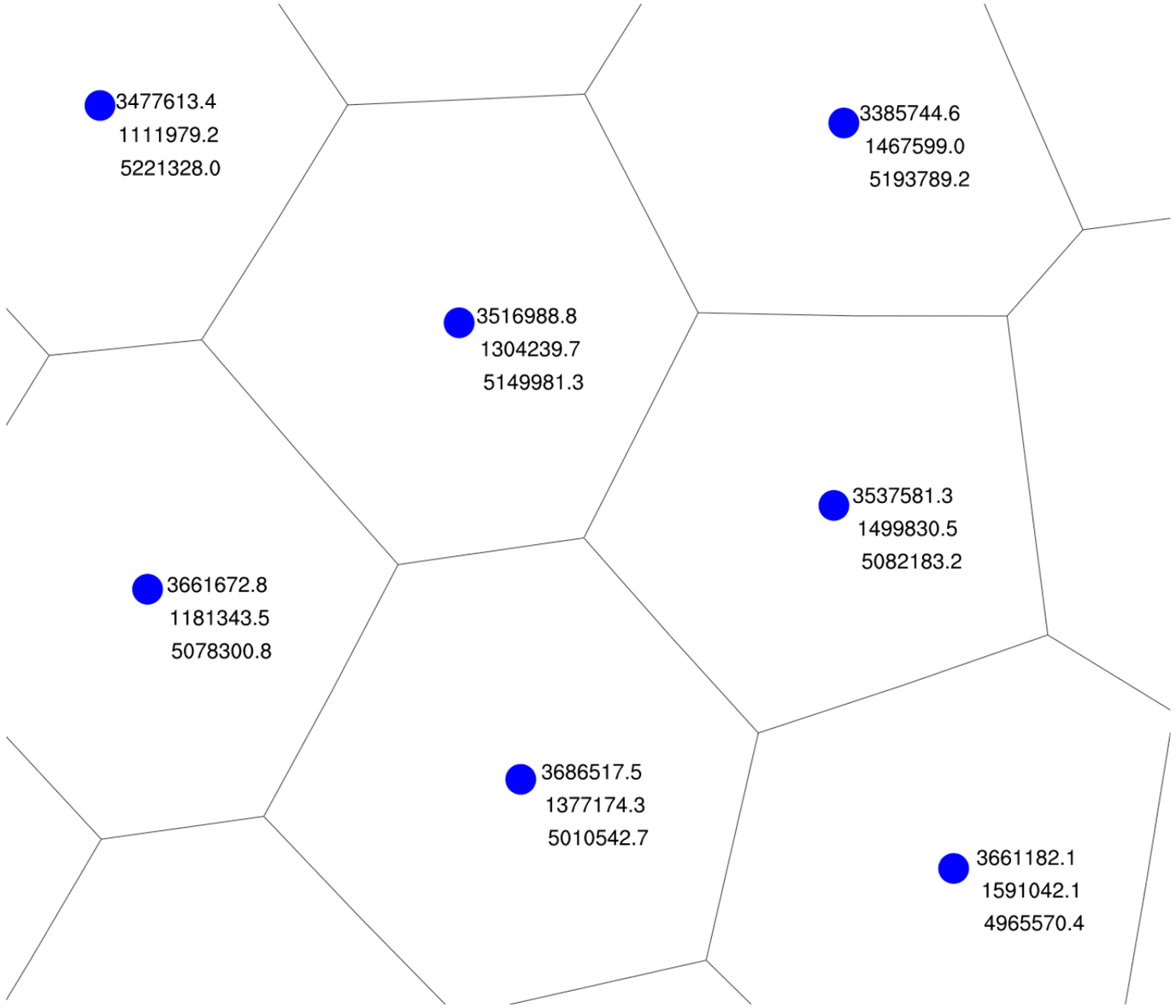
Note that we have a fully unstructured horizontal mesh, not just a deformation of the icosahedral mesh! Cells may have 5, 6, 7, or more sides.



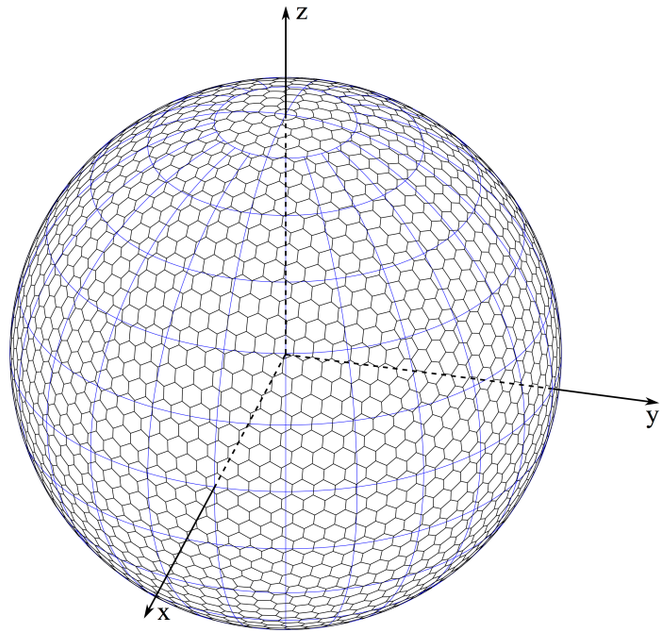
Fields in MPAS-Atmosphere may be defined at

- **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

In MPAS-A, these locations are used to implement a C-staggered grid based on the Voronoi tessellation: prognosed normal velocities are located at *edges*, and other prognosed quantities are nominally located at *cells*.



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



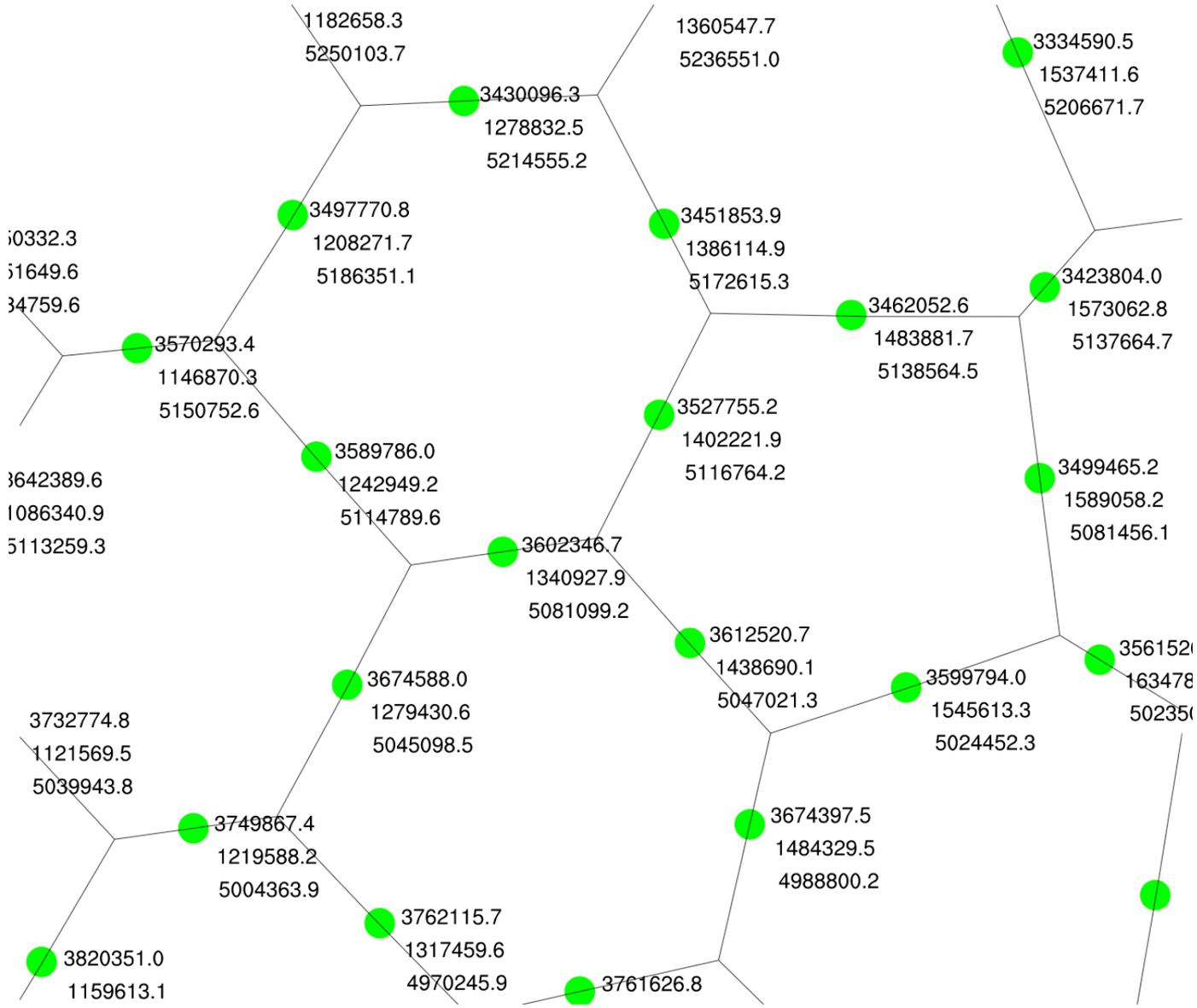
Global Cartesian coordinates are computed for each element

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

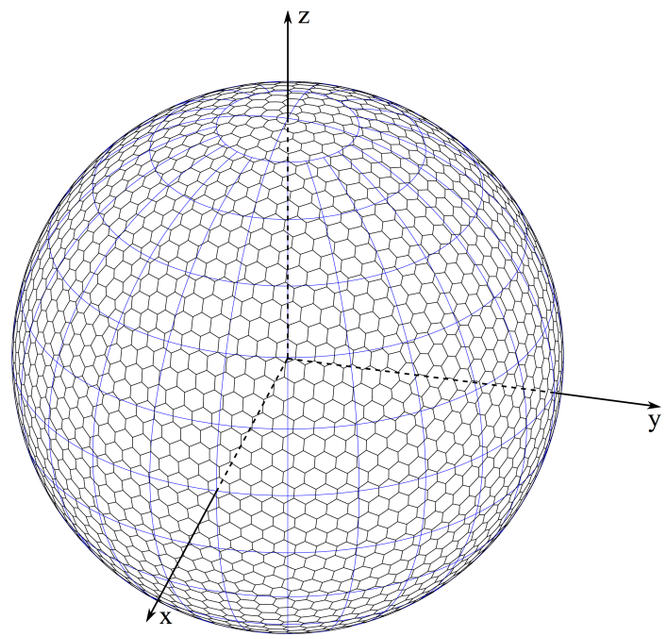
For cells: **xCell**, **yCell**, **zCell**

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



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



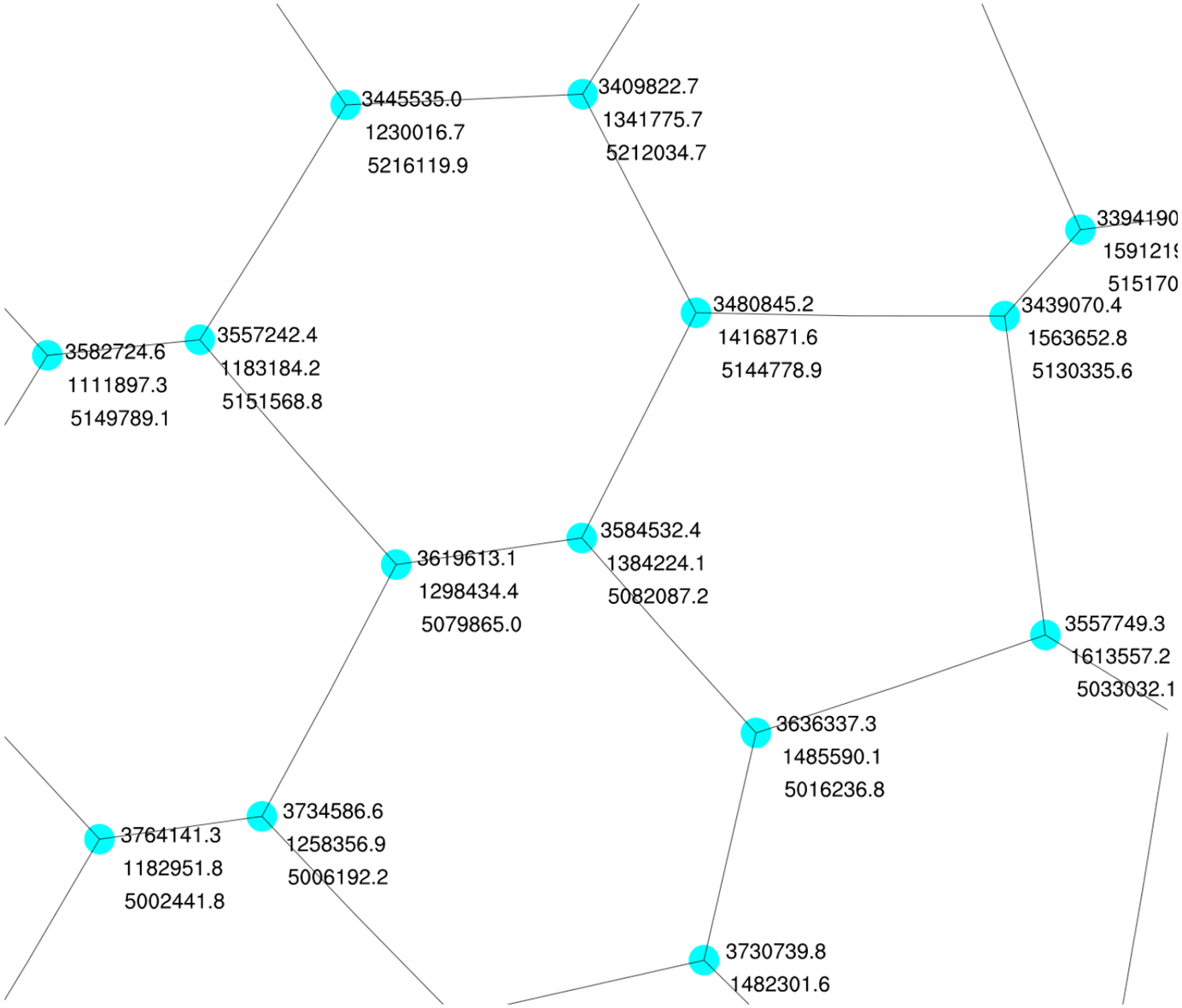
Global Cartesian coordinates are computed for each element

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

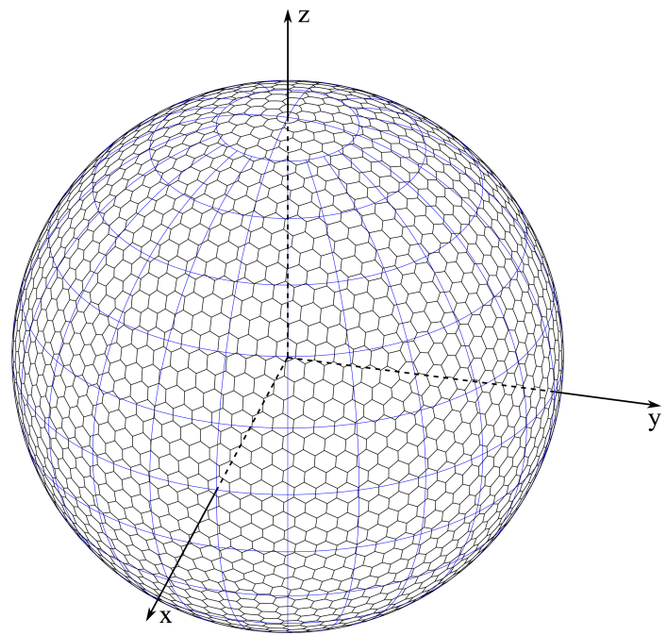
For cells: **xEdge**, **yEdge**, **zEdge**

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



Above: Cartesian coordiantes 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

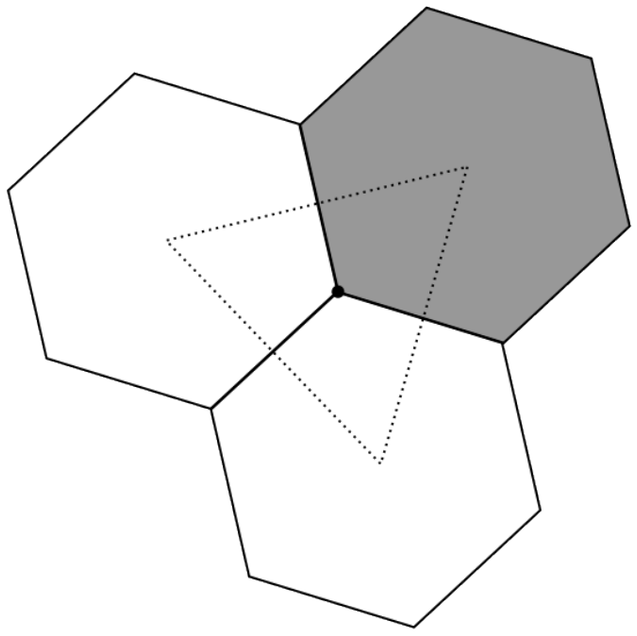
- For planar meshes, coordinates lie in the plane z=0
- For spherical meshes, coordinates lie on the surface of the sphere

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

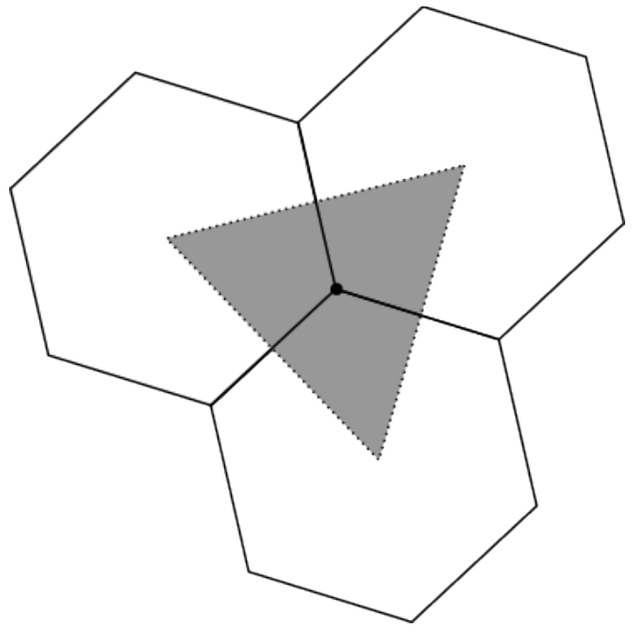
Latitudes and longitudes are computed from Cartesian coordinates as described earlier

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

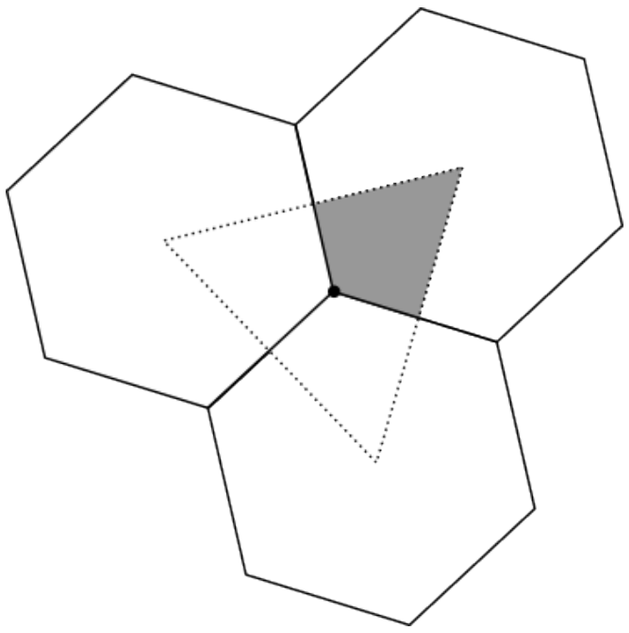
Geometry fields



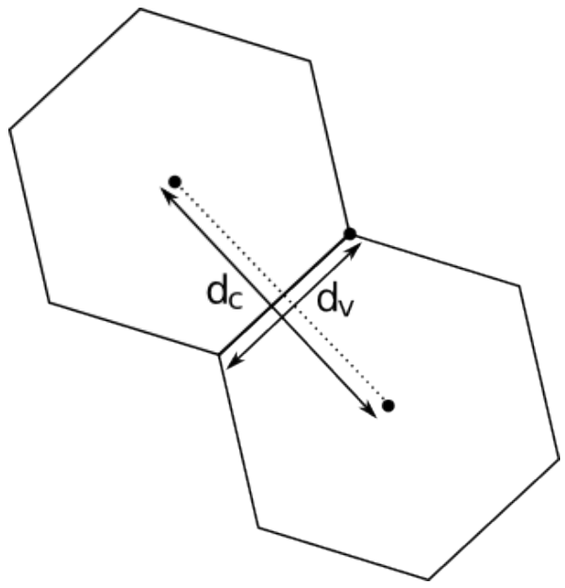
areaCell(nCells) – area of each cell



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



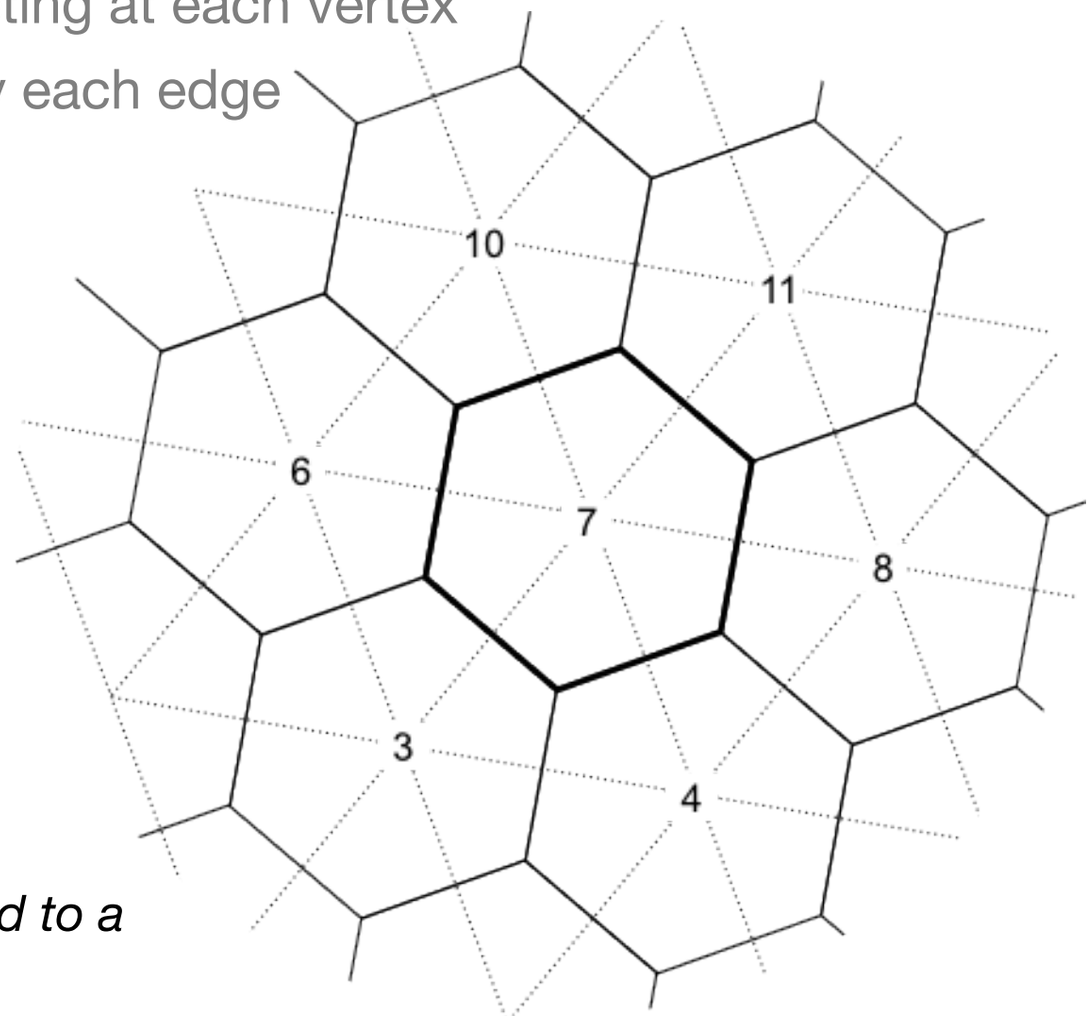
kiteAreasOnVertex(3,nVertices) – area of intersection between dual- and primal-mesh cells



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

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(3,nVertices) – the indices of edges incident with each vertex
- verticesOnEdge(2,nEdges) – the indices of endpoint vertices for each edge
- cellsOnVertex(3,nVertices) – the indices of cells meeting at each vertex
- cellsOnEdge(2,nEdges) – the indices of cells split by each edge

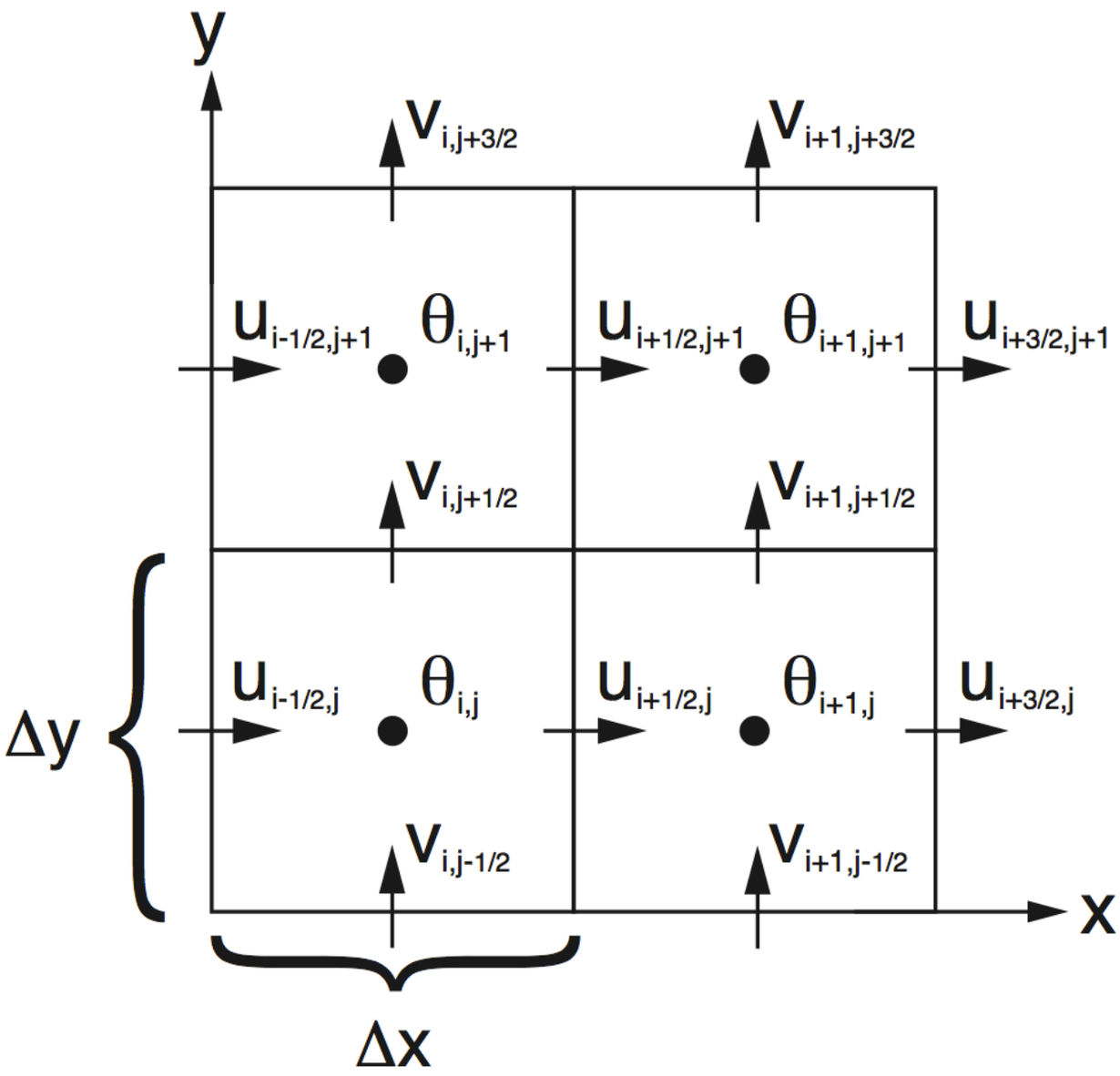


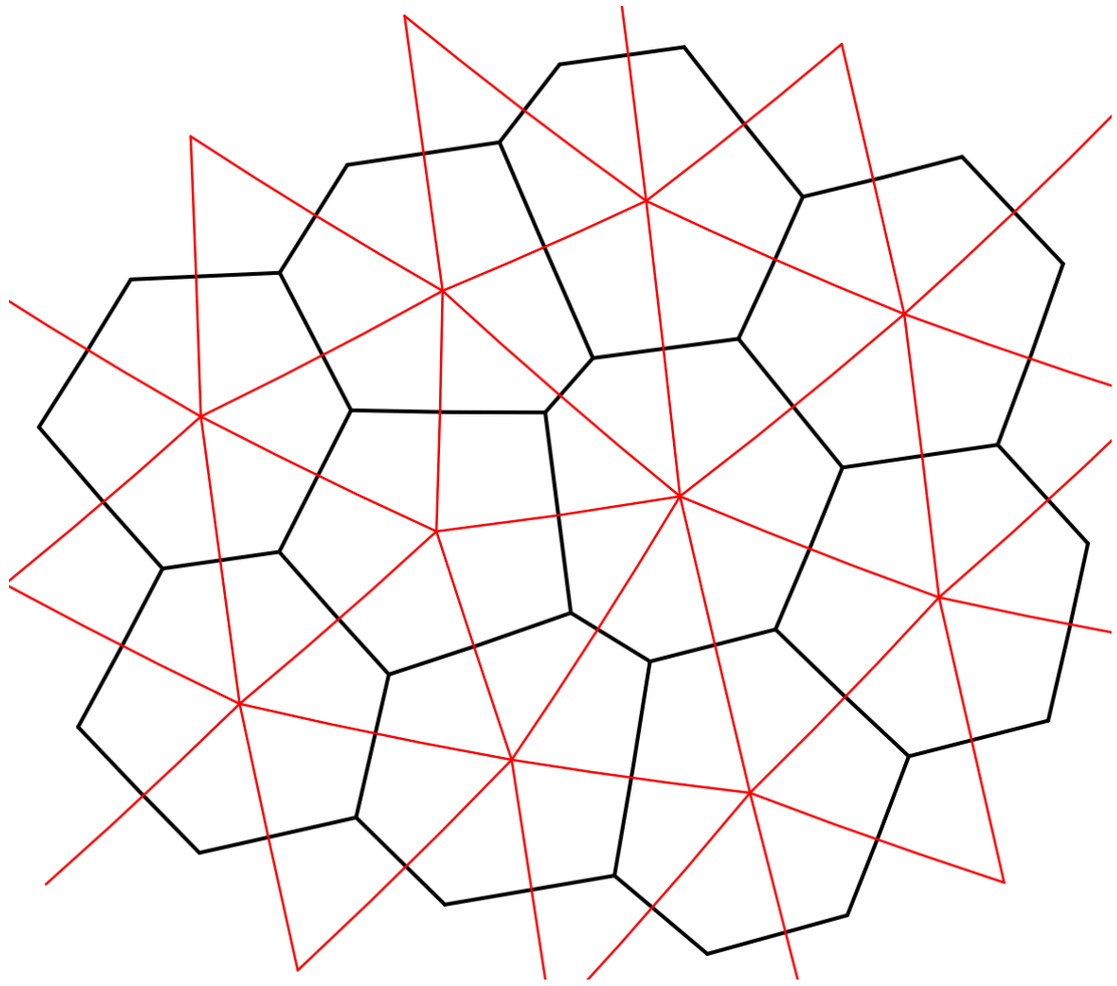

```
nEdgesOnCell(7)=6   cellsOnCell(1,7)=8
                    cellsOnCell(2,7)=11
                    cellsOnCell(3,7)=10
                    cellsOnCell(4,7)=6
                    cellsOnCell(5,7)=3
                    cellsOnCell(6,7)=4
```

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

Horizontal wind vectors

In a WRF grid, which directions represent positive U and positive V?

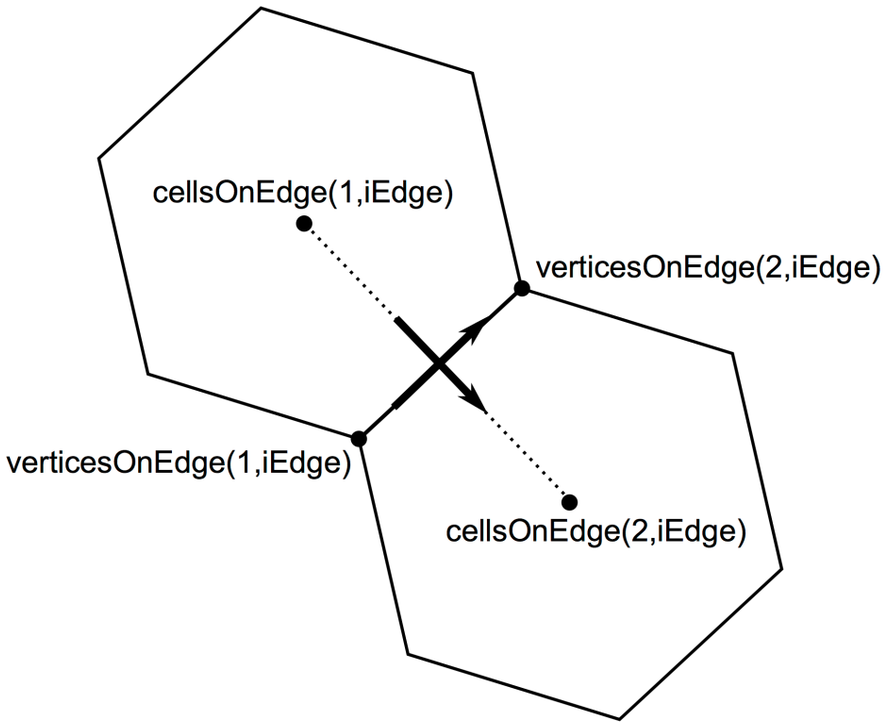




On a rectangular grid, one might say that positive U flows from left to right, and positive V flows from bottom to top when looking down on the xy-plane.

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



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

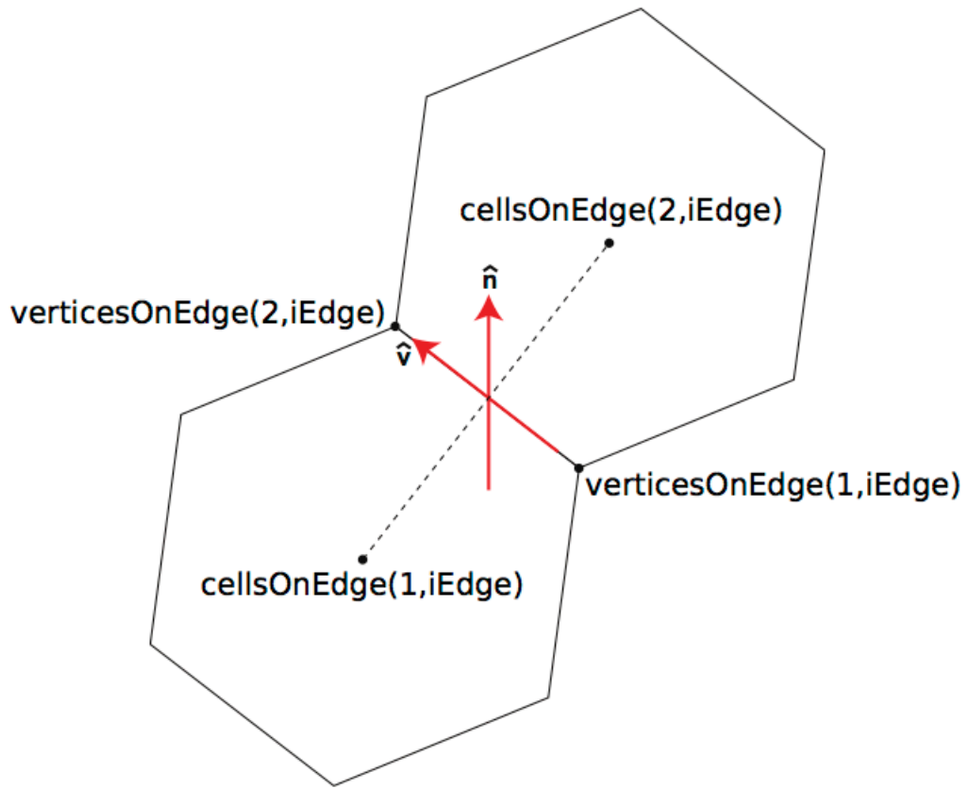
Positive v (tangential) velocity is always defined as flow from verticesOnEdge(1,iEdge) to verticesOnEdge(2,iEdge) for edge iEdge

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

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

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

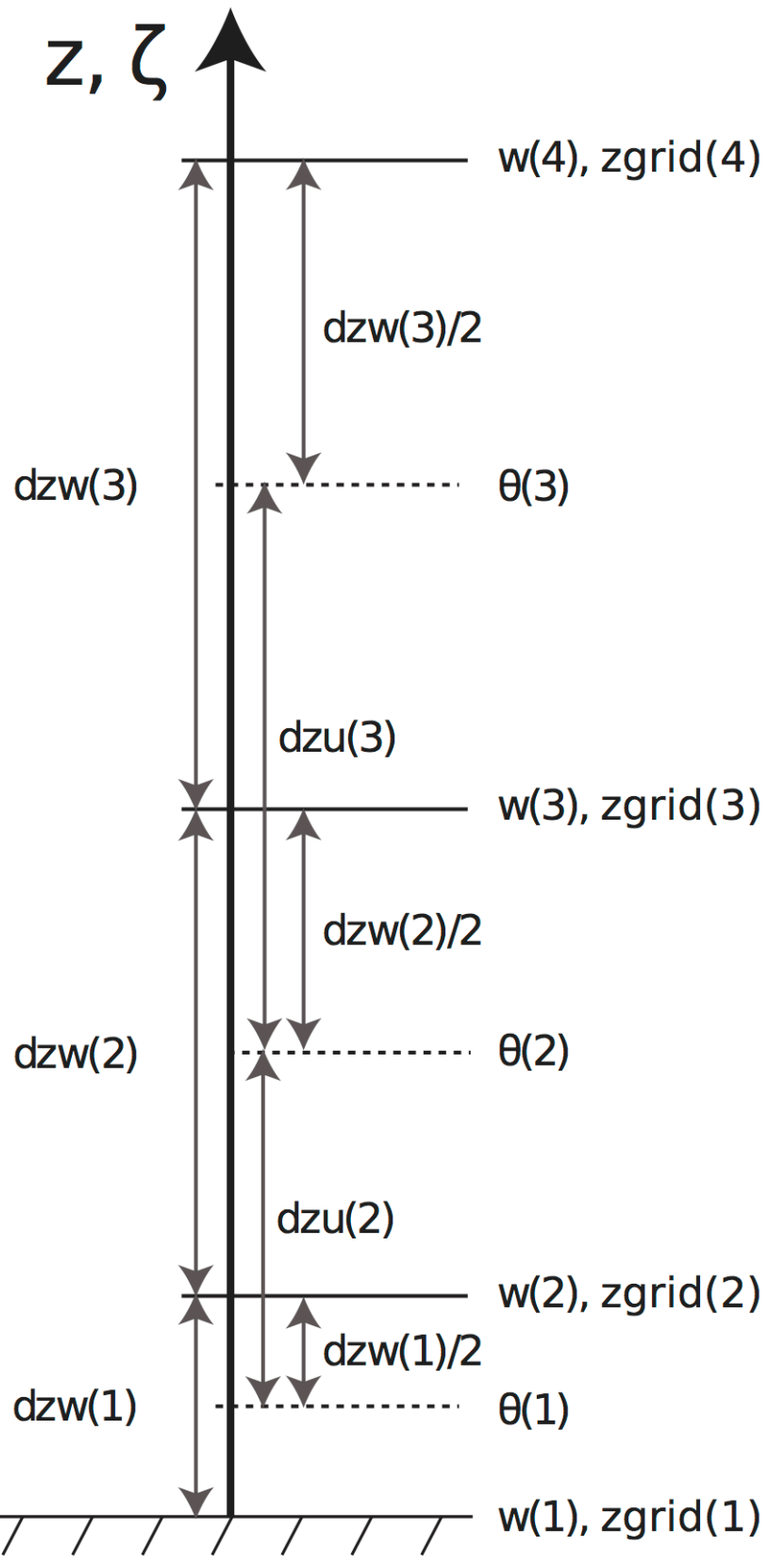
where α is angleEdge.



angleEdge(nEdges) – distances between cell centers

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

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

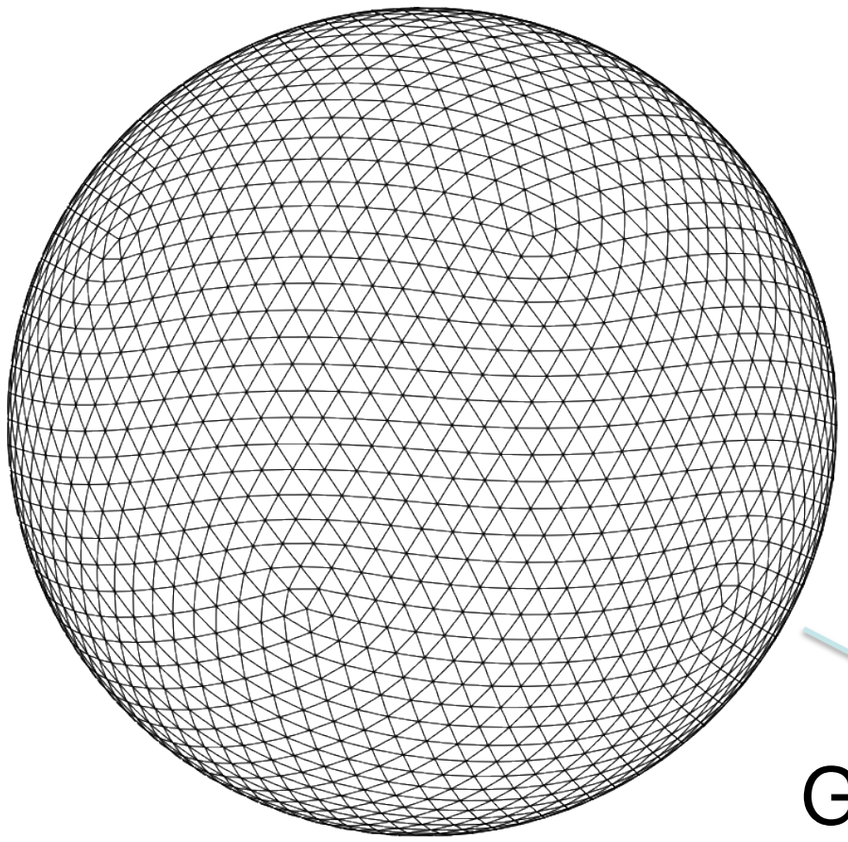
Θ levels lie at the midpoints of bracketing w levels

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

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

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

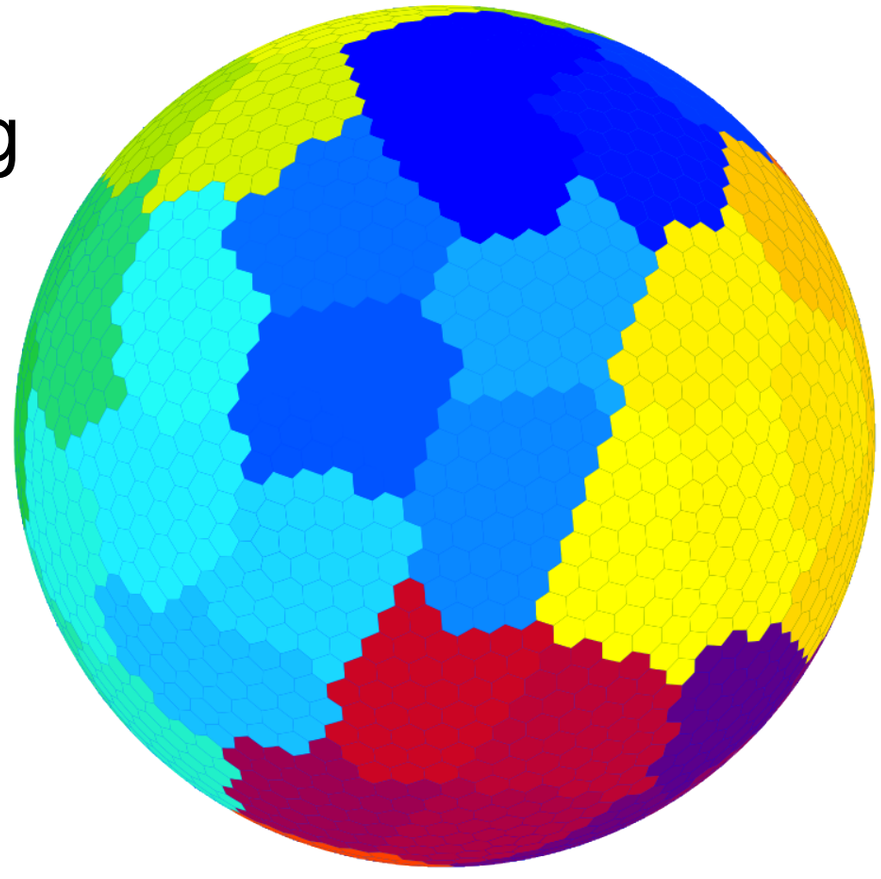
Parallel decomposition



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

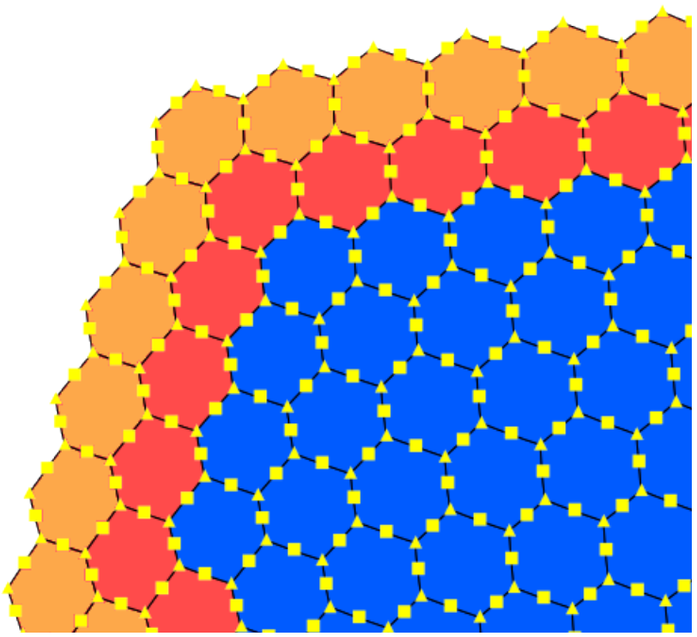
- Currently done as a pre-processing step, but could be done “on-line”

Metis also handles weighted graph partitioning

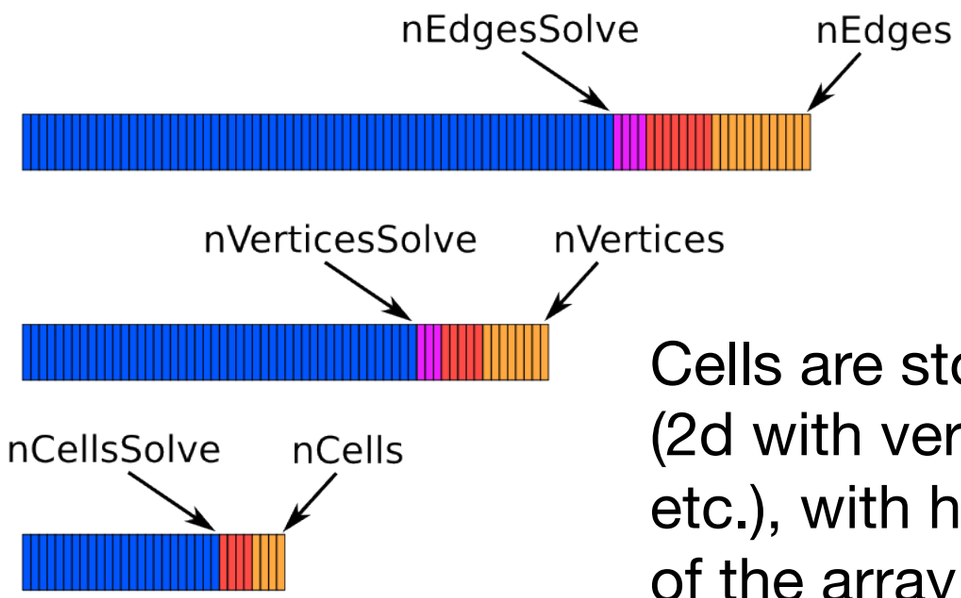
- Given *a priori* estimates for the computational costs of each grid cell, we can better balance the load among processes

Parallel decomposition

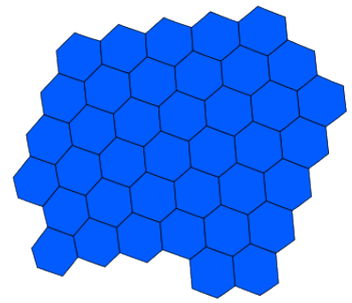
Given an assignment of cells to a process, any number of layers of halo (ghost) cells may be added



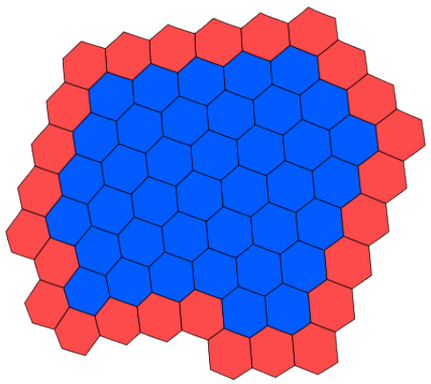
With a complete list of cells stored in a block, adjacent edge and vertex locations can be found; we apply a simple rule to determine ownership of edges and vertices adjacent to real cells in different blocks



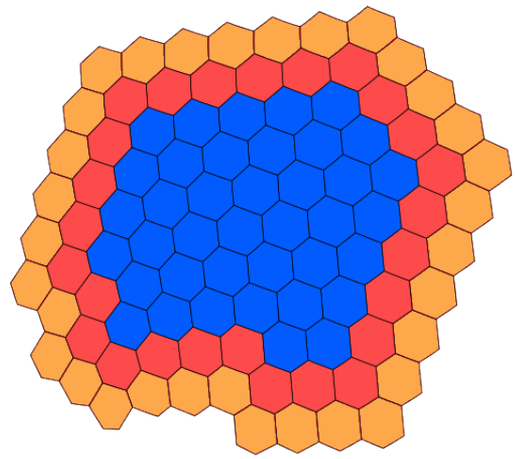
Cells are stored in a 1d array (2d with vertical dimension, etc.), with halo cells at the end of the array



Block of cells owned by a process



Block plus one layer of halo/ghost cells

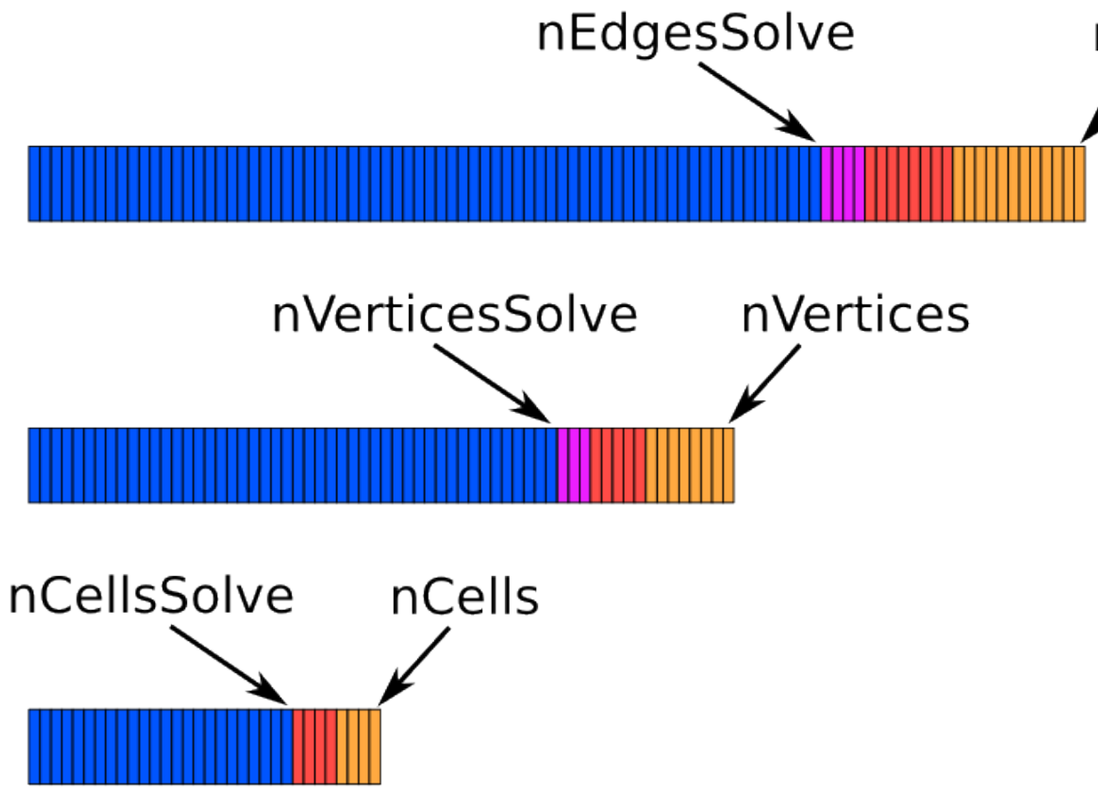


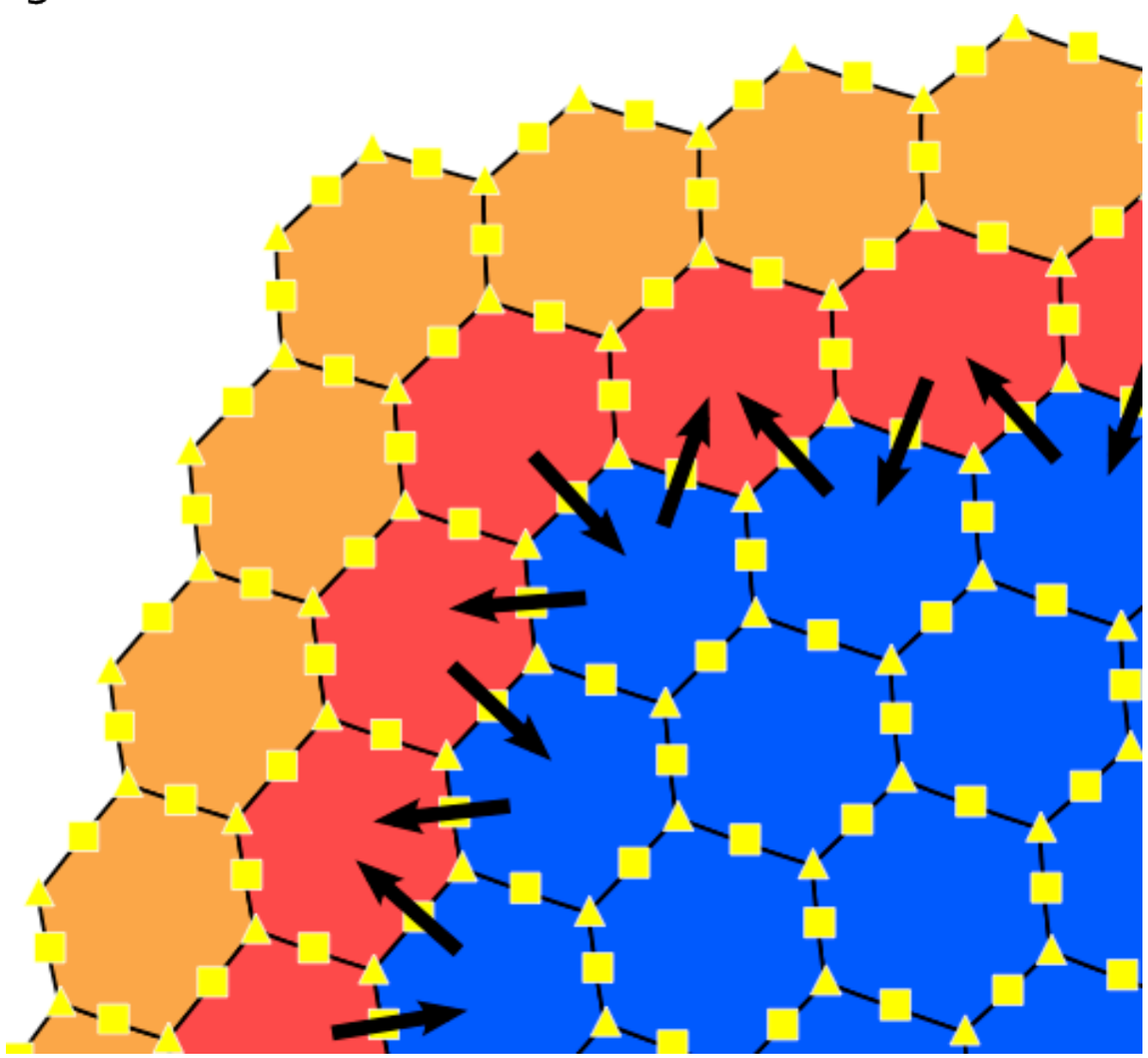
Block plus two layers of halo/ghost cells

Parallel decomposition

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.