MPAS Mesh Specification
Version 1.0

October 8, 2015

Contents

Chapter 1

Summary

This document will describe the required fields for a MPAS mesh. In addition, it will define required orderings when creating an MPAS mesh. These together should fully describe the MPAS mesh type and allow users to more easily understand what makes an MPAS mesh.

Chapter 2

Grid Description

This chapter provides a brief introduction to the common types of grids used in the MPAS framework.

The MPAS grid system requires the definition of seven elements. These seven elements are composed of two types of *cells*, two types of *lines*, and three types of *points*. These elements are depicted in Figure 2.1 and defined in Table 2.1. These elements can be defined on either the plane or the surface of the sphere. The two types of cells form two meshes, a primal mesh composed of Voronoi regions and a dual mesh composed of Delaunay triangles. Each corner of a primal mesh cell is uniquely associated with the "center" of a dual mesh cell and vice versa. So we define the two mesh as either a primal mesh (composed of cells P_i) or a dual mesh (composed of cells D_v). The center of any primal mesh cell, P_i, is denoted by $\mathbf{x}_i$ and the center of any the dual mesh cell, D_v, is denoted by $\mathbf{x}_v$. The boundary of a given primal mesh cell P_i is composed of the set of lines that connect the $\mathbf{x}_v$ locations of associated dual mesh cells D_v. Similarly, the boundary of a given dual mesh cell D_v is composed of the set of lines that connect the $\mathbf{x}_i$ locations of the associated primal mesh cells P_i.

As shown in Figure 2.1, a line segment that connects two primal mesh cell centers is uniquely associated with a line segment that connects two dual mesh cell centers. We assume that these two line segments cross and the point of intersection is labeled as $\mathbf{x}_e$. In addition, we assume that these two line segments are orthogonal as indicated in Figure 2.1. Each $\mathbf{x}_e$ is associated with two distances: d_e measures the distance between the primal mesh cells sharing $\mathbf{x}_e$ and l_e measures the distance between the dual mesh cells sharing $\mathbf{x}_e$.

Since the two line segments crossing at $\mathbf{x}_e$ are orthogonal, these line segments form a convenient local coordinate system for each edge. At each $\mathbf{x}_e$ location a unit vector $\mathbf{n}_e$ is defined to be parallel to the line connecting primal mesh cells. A second unit vector $\mathbf{t}_e$ is defined such that $\mathbf{t}_e = \mathbf{k} \times \mathbf{n}_e$.

In addition to these seven element types, we require the definition of *sets of elements*. In all, eight different types of sets are required and these are defined and explained in Table 2.2 and Figure 2.2. The notation is always of the form of, for example, $i \in CE(e)$, where the LHS indicates the type of element to be gathered (cells) based on the RHS relation to another type of element (edges).

Table 2.3 provides the names of all *elements* and all *sets of elements* as used in the MPAS framework. Elements appear twice in the table when described in the grid file in more than one way, e.g. points are described with both cartesian and latitude/longitude coordinates. An "ncdump -h" of any MPAS grid, output or restart file will contain all variable names shown in second column of Table 2.3.

Table 2.1: Definition of elements used to build the MPAS grid.

Element	*Type*	*Definition*
$\mathbf{x}_i$	point	location of center of primal-mesh cells
$\mathbf{x}_v$	point	location of center of dual-mesh cells
$\mathbf{x}_e$	point	location of edge points where velocity is defined
d_e	line segment	distance between neighboring $\mathbf{x}_i$ locations
l_e	line segment	distance between neighboring $\mathbf{x}_v$ locations
P_i	cell	a cell on the primal-mesh
D_v	cell	a cell on the dual-mesh

Table 2.2: Definition of element groups used to reference connections in the MPAS grid. Examples are provided in Figure 2.2.

Syntax	*ouptut*
$e \in EC(i)$	set of edges that define the boundary of P_i.
$e \in EV(v)$	set of edges that define the boundary of D_v.
$i \in CE(e)$	two primal-mesh cells that share edge e.
$i \in CV(v)$	set of primal-mesh cells that form the vertices of dual mesh cell D_v.
$v \in VE(e)$	the two dual-mesh cells that share edge e.
$v \in VI(i)$	the set of dual-mesh cells that form the vertices of primal-mesh cell P_i.
$e \in ECP(e)$	edges of cell pair meeting at edge e.
$e \in EVC(v,i)$	edge pair associated with vertex v and mesh cell i.

Table 2.3: Variable names used to describe a MPAS grid.

Element	*Name*	*Size*	*Comment*
$\mathbf{x}_i$	{x,y,z}Cell	nCells	cartesian location of $\mathbf{x}_i$
$\mathbf{x}_i$	{lon,lat}Cell	nCells	longitude and latitude of $\mathbf{x}_i$
$\mathbf{x}_v$	{x,y,z}Vertex	nVertices	cartesian location of $\mathbf{x}_v$
$\mathbf{x}_v$	{lon,lat}Vertex	nVertices	longitude and latitude of $\mathbf{x}_v$
$\mathbf{x}_e$	{x,y,z}Edge	nEdges	cartesian location of $\mathbf{x}_e$
$\mathbf{x}_e$	{lon,lat}Edge	nEdges	longitude and latitude of $\mathbf{x}_e$
d_e	dcEdge	nEdges	distance between $\mathbf{x}_i$ locations
l_e	dvEdge	nEdges	distance between $\mathbf{x}_v$ locations
$e \in EC(i)$	edgesOnCell	(nEdgesMax,nCells)	edges that define P_i.
$e \in EV(v)$	edgesOnVertex	(3,nCells)	edges that define D_v.
$i \in CE(e)$	cellsOnEdge	(2,nEdges)	primal-mesh cells that share edge e.
$i \in CV(v)$	cellsOnVertex	(3,nVertices)	primal-mesh cells that define D_v.
$v \in VE(e)$	verticesOnEdge	(2,nEdges)	dual-mesh cells that share edge e.
$v \in VI(i)$	verticesOnCell	(nEdgesMax,nCells)	vertices that define P_i.

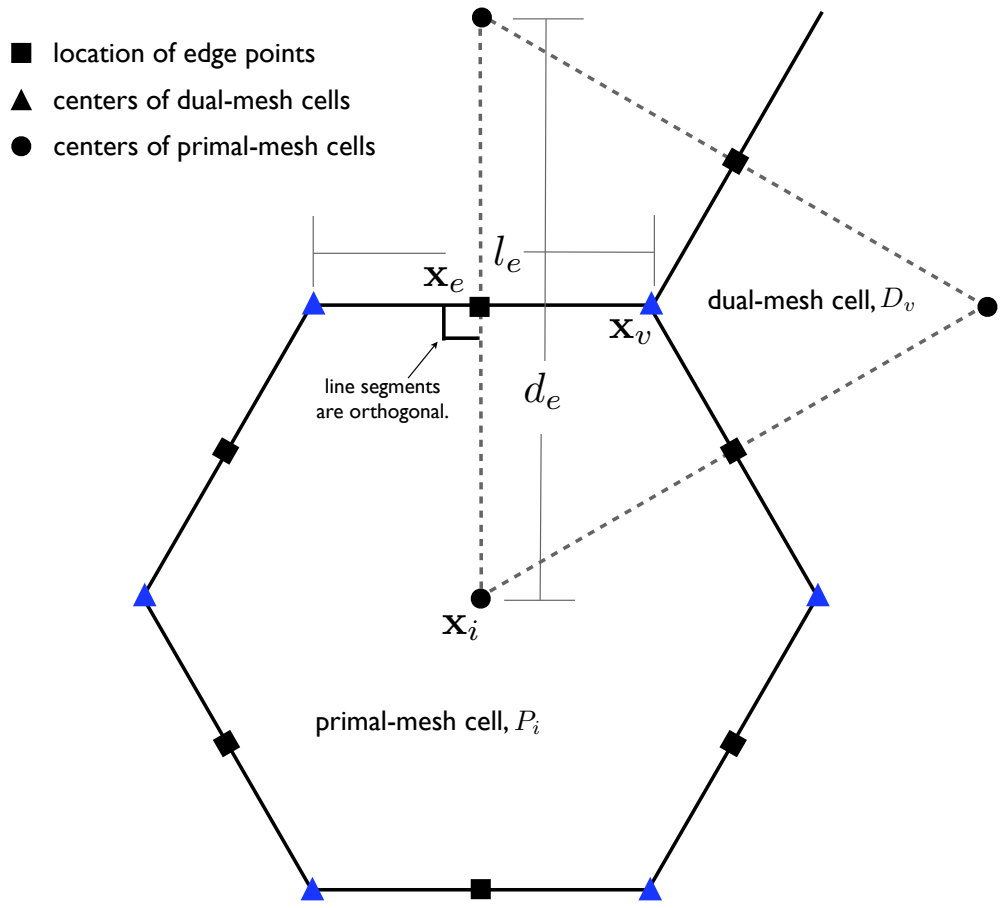


Figure 2.1: Definition of elements used to build the MPAS grid. Also see Table 2.1.

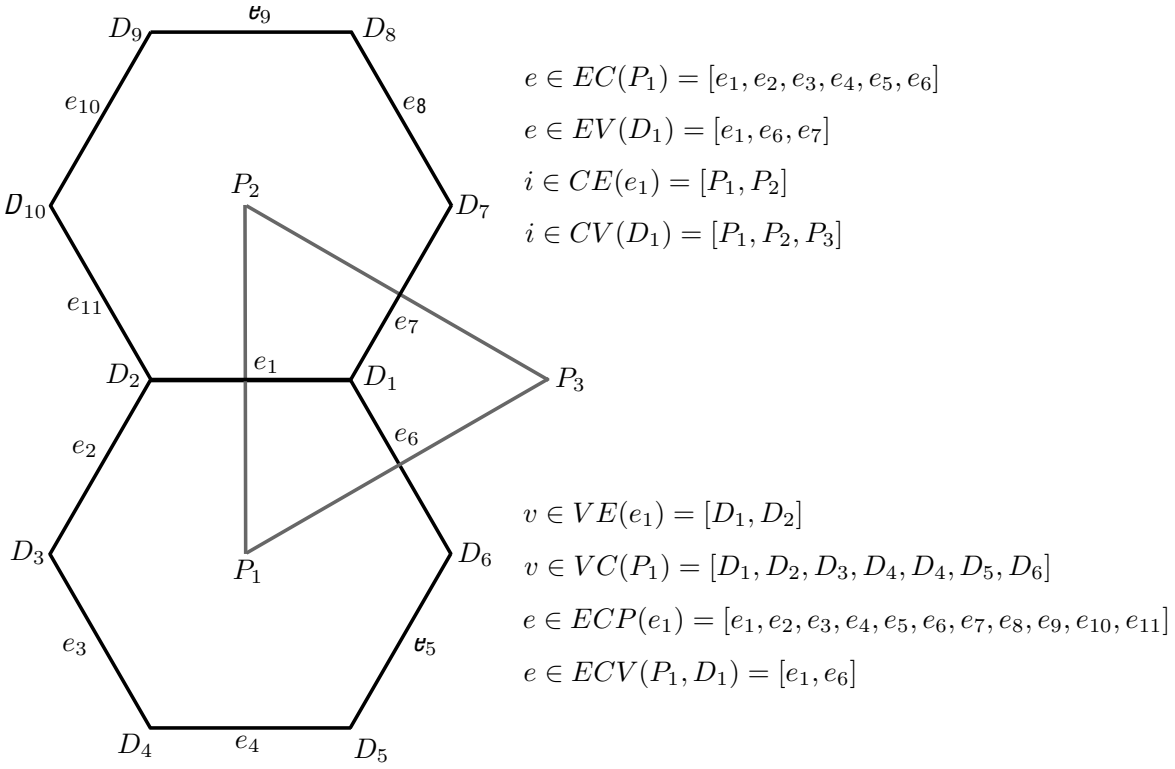


Figure 2.2: Definition of element groups used to reference connections in the MPAS grid. Also see Table 2.2.

Chapter 3

Definitions and Conventions

The meshes used by MPAS are Voronoi tessellations (VTs), in which MPAS identifies three types of elements: *cells*, *edges*, and *vertices*. *Cells* are simply the Voronoi cells in the tessellation, *edges* are the boundaries between adjacent Voronoi cells, and *vertices* are the corners of cells. In MPAS, cells are nominally located at the Voronoi generating points, which, for centroidal Voronoi tessellations, are the mass centoids of the Voronoi cells with respect to a density function, and edges are nominally located at the midpoints of edges. Figure 3.1 shows three cells with their associated edges and vertices.

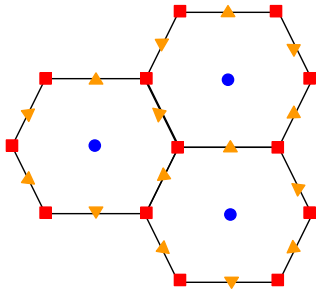
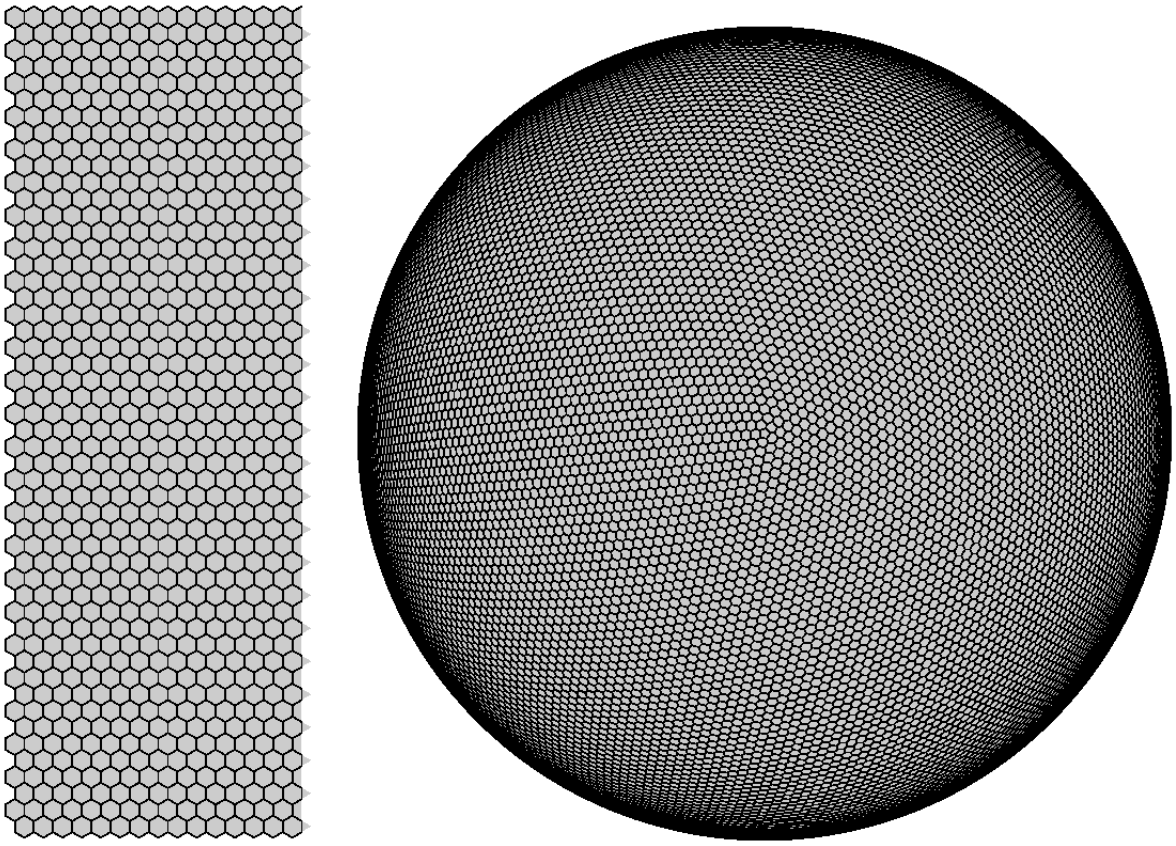


Figure 3.1: Three cell Voronoi tessellation with Cells denoted by blue circles, edges by orange triangles, and vertices by red squares.

The VT meshes used by MPAS may be defined on the Cartesian plane or on a sphere. In the case of planar meshes, areas and distances are assumed to be Euclidean, and the mesh is defined on the plane $z = 0$. In the case of spherical meshes, areas and distances are computed in spherical geometry; cells, edges, and vertices are constrained to lie on the surface of the sphere, and the sphere is centered at the origin, $(x, y, z) = (0, 0, 0)$. In all cases, the coordinate systems assumed by MPAS meshes are right-handed. Figure 3.2 provides and illustration of VT meshes on the Cartesian plane and on the sphere. There are no requirements for the radius of the sphere, however an attribute sphere_radius is required to be defined that defines the radius of the sphere.



(a) Planar Voronoi tessellation

(b) Spherical Voronoi tessellation

Figure 3.2: Examples of Voronoi tessellations

Chapter 4

Required Variables, Attributes, and Dimensions

The following list of attributes are required by the MPAS framework. All MPAS meshes should contain these global attributes.

- on_a_sphere
  Type: Character
  Valid Values: "YES" or "NO"
  Description: Defines if the mesh describes points that lie on the surface of a sphere or not.

- sphere_radius
  Type: Double
  Valid Values: Any positive real value
  Description: Value of the radius of the sphere the points are defined on. Can be set to 0 or 1 if in a plane.

- mesh_id
  Type: Character
  Valid Values: Any combination of lowercase letters and numbers.
  Description: Typically a random string use for tracking mesh provenance.

- mesh_spec
  Type: Character
  Valid Values: Any valid version of the MPAS mesh specification Description: Defines the version of the MPAS Mesh specification the mesh conforms to.

In addition to the above attributes, there are three other attributes that may be required. When on_a_sphere is set to "NO", the following attribute is required:

- is_periodic
  Type: Character
  Valid Values: "YES" or "NO"
  Description: Defines if the mesh has any periodic boundaries. This is only used to determine if other attributes should be read.

If is_periodic is set to "YES", the following two attributes are required as well:

- x_period
  Type: Double
  Valid Values: Any positive real value
  Description: Defines the period of the mesh in the X direction

- y_period
  Type: Double
  Valid Values: Any positive real value
  Description: Defines the period of the mesh in the Y direction

All MPAS cores are required to define the following dimensions for framework.

- nCells - The number of primary cells in the mesh.

- nEdges - The number of edges in the mesh.

- nVertices - The number of vertices in the mesh, or the number cells in the dual mesh.

- vertexDegree - The maximum number of cells connected to a dual cell (or the number of corners in a dual cell).

- maxEdges - The maximum number of vertices and edges on any primary cell.

- maxEdges2 - The value of maxEdges times 2.

The following list of fields are required by all MPAS cores, and the MPAS framework assumes these fields exist.

- latCell - Latitude in radians of all cell centers
  Dimensions: nCells
  Valid Values: Reals in the range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
  Special Notes: *Could be computed internally.*

- lonCell - Longitude in radians of all cell centers.
  Dimensions: nCells
  Note: In a plane this should be given a constant value of 0.
  Valid Values: Reals in the range of 0 to $2 * \pi$.
  Special Notes: *Could be computed internally.*

- xCell - x axis position of all cell centers.
  Dimensions: nCells

- yCell - y axis position of all cell centers.
  Dimensions: nCells

- zCell - z axis position of all cell centers.
  Dimensions: nCells

- indexToCellID - Global cell ID for all cell centers.
  Dimensions: nCells

- latEdge - Latitude in radians of all edge locations.
  Dimensions: nEdges
  Valid Values: Reals in the range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
  Special Notes: *Could be computed internally.*

- lonEdge - Longitude in radians of all edge locations.
  Dimensions: nEdges
  Valid Values: Reals in teh range of 0 to $2 * \pi$.
  Special Notes: *Could be computed internally.*

- xEdge - x axis position of all edge locations.
  Dimensions: nEdges

- yEdge - y axis position of all edge locations.
  Dimensions: nEdges

- zEdge - z axis position of all edge locations.
  Dimensions: nEdges

- indexToEdgeID - Global edge ID for all edge locations.
  Dimensions: nEdges

- latVertex - Latitude in radians of all cell vertices.
  Dimensions: nVertices
  Valid Valids: Reals in the range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
  Special Notes: *Could be computed internally.*

- lonVertex - Longitude in radians of all cell vertices.
  Dimensions: nVertices
  Valid Valids: Reals in the range of 0 to $2 * \pi$.
  Special Notes: *Could be computed internally.*

- xVertex - x axis position of all cell vertices.
  Dimensions: nVertices

- yVertex - y axis position of all cell vertices.
  Dimensions: nVertices

- zVertex - z axis position of all cell vertices.
  Dimensions: nVertices

- indexToVertexID - Global vertex ID for all cell vertices.
  Dimensions: nVertices

- nEdgesOnCell - Number of edges on a given cell.
  Dimensions: nCells

- nEdgesOnEdge - Number of edges on a given edge. Used to reconstruct tangential velocities.
  Dimensions: nEdges

- cellsOnEdge - Cell indices that saddle a given edge.
  Dimensions: 2 * nEdges

- edgesOnCell - Edge indices that surround a given cell.
  Dimensions: maxEdges * nCells

- edgesOnEdge - Edge indices that are used to reconstruct tangential velocities.
  Dimensions: maxEdges2 * nEdges

- cellsOnCell - Cell indices that surround a given cell.
  Dimensions: maxEdges * nCells

- verticesOnCell - Vertex indices that surround a given cell.
  Dimensions: maxEdges * nCells

- verticesOnEdge - Vertex indices that saddle a given edge.
  Dimensions: 2 * nEdges

- edgesOnVertex - Edge indices that radiate from a given vertex.
  Dimensions: vertexDegree * nVertices

- cellsOnVertex - Cell indices that radiate from a given vertex.
  Dimensions: vertexDegree * nVertices

- weightsOnEdge - Weights used to reconstruct tangential velocities.
  Dimensions: maxEdges2 * nEdges
  Special Notes: *Could be computed internally.*

- dvEdge - Distance in meters between the vertices that saddle a given edge.
  Dimensions: nEdges
  Special Notes: *Could be computed internally.*

- dcEdge - Distance in meters between the cells that saddle a given edge.
  Dimensions: nEdges
  Special Notes: *Could be computed internally.*

- angleEdge - Angle in radians an edge's normal vector makes with the local eastward direction.
  Dimensions: nEdges
  Special Notes: *Could be computed internally.*

- areaCell - Area in square meters for a given cell of the primary mesh.
  Dimensions: nCells
  Special Notes: *Could be computed internally.*

- areaTriangle - Area in square meters for a given triangle of the dual mesh.
  Dimensions: nVertices
  Special Notes: *Could be computed internally.*

- kiteAreasOnVertex - The intersection area of areaTriangle with each cell that radiates from a given vertex.
  Dimensions: vertexDegree * nVertices
  Special Notes: *Could be computed internally.*

- meshDensity - The value of the generating density function at each cell center. Dimensions: nCells

Chapter 5

Connectivity and Ordering Requirements

This chapter defines the general requirements for all MPAS meshes. Along with specific requirements for different element types (cell, edge, vertex). These include ordering specifications for one type of element relative to another.

- MPAS meshes must be defined using a right handed coordinate system.
- Spherical grids must be centered at (0,0,0).
- Two arrays that are both relative to the element type must be ordered in the exact same way if possible.
  For example: When on a particular cell, edgesOnCell(n, iCell) should be the edge between iCell and cellsOnCell(n, iCell).
- Input meshes are required to have a time dimension that is the unlimited (record) dimension.

When creating an MPAS mesh, it is recommended to ensure the correct connectivity ordering relative to edges, then vertices, then cells. Ordering things in this way simplifies the process.

5.1 Missing Values

Often when setting up a mesh for MPAS, there might be a cell that is missing a neighbor across one or more of it's edges. An example of this would be an ocean mesh where land cells are actually removed from the domain.

When this occurs, connectivity lists (for example cellsOnCell) need to be updated to account for this missing element. In this case, the entry in the connectivity list should be replaced by a value of 0.

Additionally, there are instances where an element (for example a cell) has less edges than the *maxEdges* dimension. For example, if a heptagon is present in the mesh, *maxEdges* would have a value of 7, any hexagons or pentagons in the mesh would only have 6 and 5 edges, respectively. In this case, the connectivity list for each element with less entries needs to be padded. For MPAS meshes, the padding is arbitrary (meaning MPAS doesn't care what the padding is). Some external tools have requirements about this padding (some require the last entry be repeated, while others require padding with zeroes). It's recommended to explore your entire tool chain, and ensure the padding is setup consistently for all of your tools.

5.2 Requirements relative to edges

At a given edge, two vectors $\vec{u}$ and $\vec{v}$ are defined as the normal and tangential vectors, respectively. These are defined as:

$$\vec{u} = cellsOnEdge(2, iEdge) - cellsOnEdge(1, iEdge) \tag{5.1}$$

$$\vec{v} = verticesOnEdge(2, iEdge) - verticesOnEdge(1, iEdge) \tag{5.2}$$

where cellsOnEdge and verticesOnEdge are the indices for the cells and vertices that make up an edge respectively. They should be expanded to their full coordinates in order to make a vector. Their coordinates are defined in the xCell, yCell, zCell, xVertex, yVertex, and zVertex fields.

- The surface normal vector must be defined as $\vec{u} \times \vec{v}$.
- Angle edge must be the angle in radians $\vec{u}$ makes with the local eastward direction on a sphere, or the local x direction in a plane.
- edgesOnEdge must run counter-clockwise, beginning with the edges that surround cellsOnEdge(1, iEdge) and ending with the edges that surround cellsOnEdge(2, iEdge).
  The current edge must be omitted from the list of edgesOnEdge, but can be assumed to be both the starting and ending position when checking for counter-clockwise ordering.
- weightsOnEdge must be ordered in exactly the same order as edgesOnEdge. i.e. weightsOnEdge(1, iEdge) can be assumed to apply to edgesOnEdge(1, iEdge).

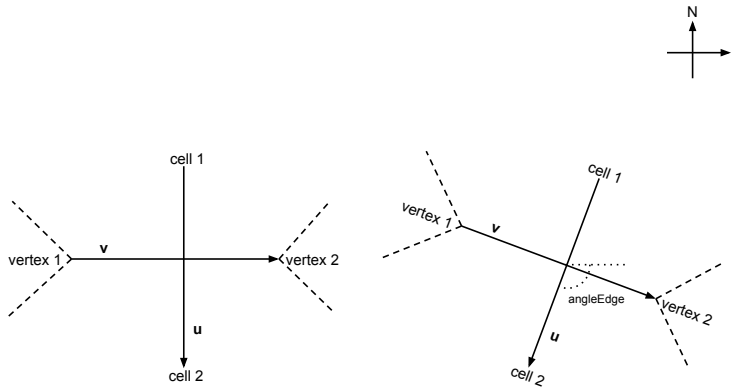


Figure 5.1: Ordering of elements relative to edges.

5.3 Requirements relative to vertices

- Cells and Edges must run counter-clockwise around a given vertex.

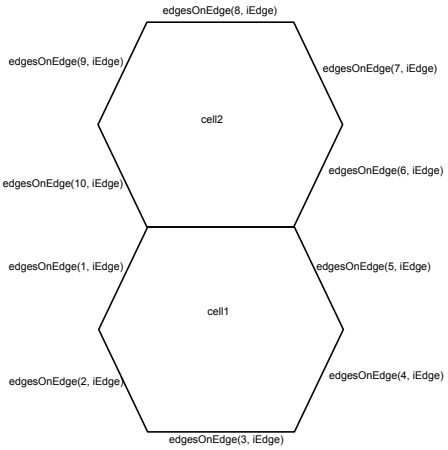


Figure 5.2: Ordering of edges relative to edges.

- Edges must lead cells as they move around a vertex.
  i.e. The vector defined by
  $(cellsOnVertex(n, iVertex) - iVertex) \times (edgesOnVertex(n, iVertex) - iVertex)$
  must be surface normal, for all values of n.

- kiteAreasOnVertex(n, iVertex) is the intersection area of areaTriangle(iVertex) with areaCell(cellsOnVertex(n,iVertex)) for all values of n.

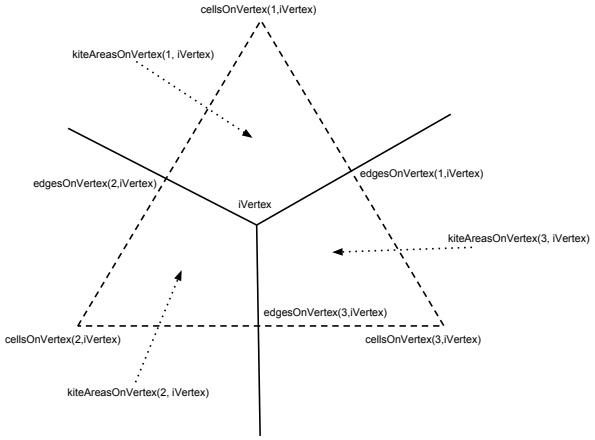


Figure 5.3: Ordering of elements relative to vertices.

5.4 Requirements relative to cells

- Cells, Edges, and Vertices all run counter-clockwise around a cell.
- The edge defined at edgesOnCell(n, iCell) must be on the edge between iCell and cellsOnCell(n, iCell) for all values of n.
- verticesOnCell(n, iCell) leads both edgesOnCell(n, iCell) and cellsOnCell(n, iCell) for all values of n.
  i.e. The vector defined by
  $(edgesOnCell(n, iCell) - iCell) \times (verticesOnCell(n, iCell) - iCell)$
  must be surface normal for all values of n, or the substitution of cellsOnCell for edgesOnCell.

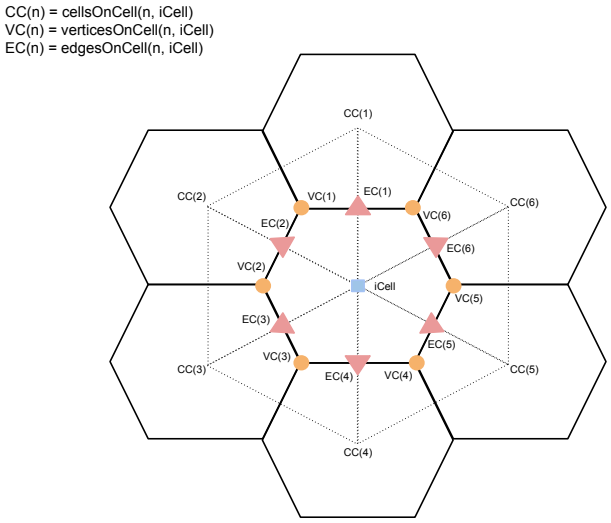


Figure 5.4: Ordering of elements relative to cells.

5.5 Description of boundaries

Some meshes require the ability to describe boundaries. These could describe the interface between two different meshes, or active and inactive cells. For example, the boundary between ocean and land, or a nested grid and it's global counterpart.

The easiest way to describe boundaries is through the edges presently. If an edge is intended to track a boundary, i.e. lies between active and inactive cells, this can be specified through cellsOnEdge. If the second cell in cellsOnEdge is given a value of zero, it is assumed that the edge falls on a boundary.

Chapter 6

Validation of meshes

Meshes can be validated by performing a simulation in the shallow water core. Preferably test cases 2 and 5. If there are problems with areas, connectivity, or weights these would create obvious errors in the output of the shallow water core.