MPAS Mesh Specification
Version 1.0

January 7, 2014

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

Definitions and Conventions

The meshes used by MPAS are centroidal Voronoi tessellations (CVTs), 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 2.1 shows three cells with their associated edges and vertices.

Figure 2.1: Blah.

The CVT 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 2.2 provides and illustration of CVT meshes on the Cartesian plane and on the sphere.

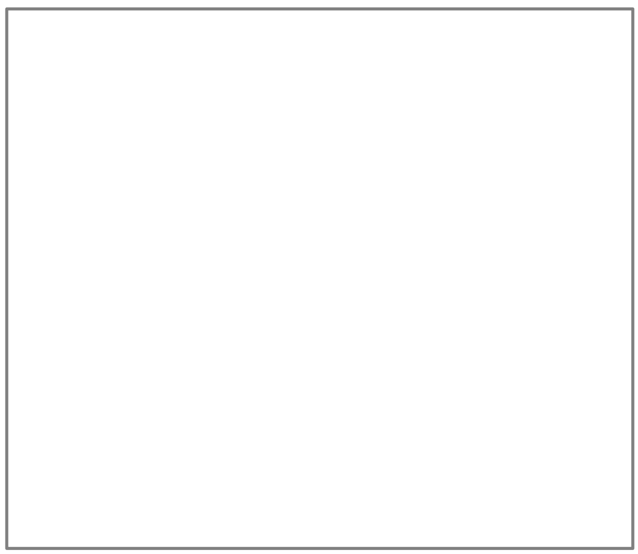

Figure 2.2: Blah.

Chapter 3

General Mesh 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.
- 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 ordering relative to edges, then vertices, then cells. Ordering things in this way simplifies the process.

3.1 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:

$$\vec{u} = cellsOnEdge(2, iEdge) - cellsOnEdge(1, iEdge) \tag{3.1}$$

$$\vec{v} = verticesOnEdge(2, iEdge) - verticesOnEdge(1, iEdge) \tag{3.2}$$

- 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.
- 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).

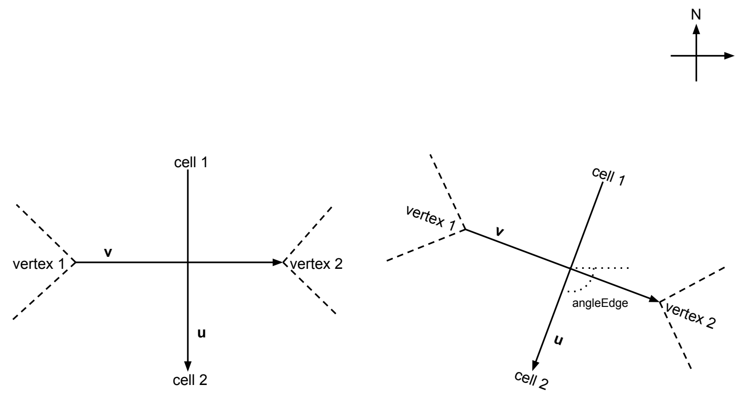

Diagram.pdf

Figure 3.1: Ordering of elements relative to edges.

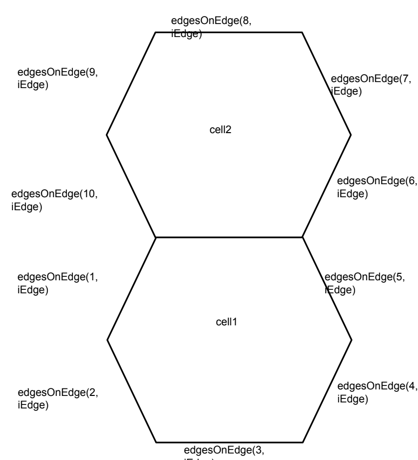

Diagram.pdf

Figure 3.2: Ordering of edges relative to edges.

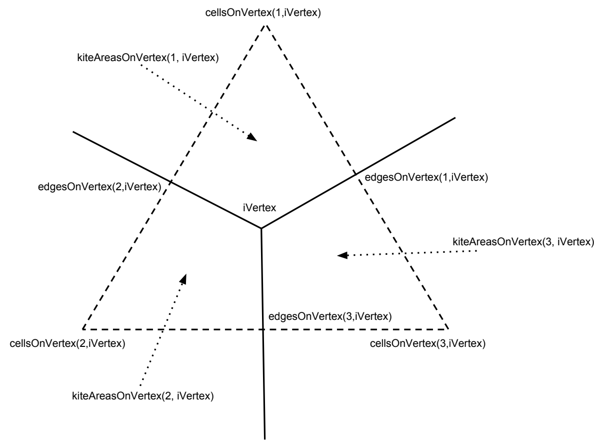

Diagram.pdf

Figure 3.3: Ordering of elements relative to vertices.

3.2 Requirements relative to vertices

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

3.3 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.

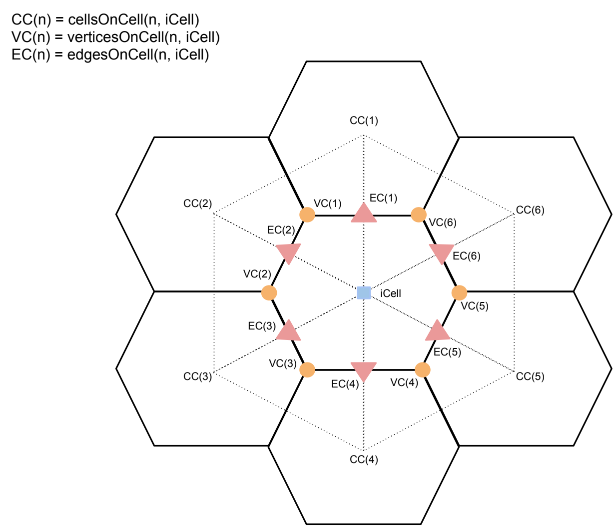

Diagram.pdf

Figure 3.4: Ordering of elements relative to cells.

Chapter 4

Requirements

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. Valid range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
  Dimensions: nCells
  Could be computed internally.
- lonCell - Longitude in radians of all cell centers. Valid range of 0 to π.
  Dimensions: nCells
  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. Valid range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
  Dimensions: nEdges Could be computed internally.
- lonEdge - Longitude in radians of all edge locations. Valid range of 0 to π.
  Dimensions: nEdges 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. Valid range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$.
  Dimensions: nVertices Could be computed internally.
- lonVertex - Longitude in radians of all cell vertices. Valid range of 0 to π.
  Dimensions: nVertices 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
- cellsOnEdge - Cell indices that saddle a given edge.
  Dimensions: 2 * nEdges
- nEdgesOnCell - Number of edges on a given cell.
  Dimensions: maxEdges * nCells
- nEdgesOnEdge - Number of edges on a given edge. Used to reconstruct tangential velocities.
  Dimensions: maxEdges2 * 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
- weightsOnEdge - Weights used to reconstruct tangential velocities.
  Dimensions: maxEdges2 * nEdges Could be computed internally.
- dvEdge - Distance in meters between the vertices that saddle a given edge.
  Dimensions: nEdges Could be computed internally.
- dcEdge - Distance in meters between the cells that saddle a given edge.
  Dimensions: nEdges Could be computed internally.
- angleEdge - Angle in radians an edge's normal vector makes with the local eastward direction.
  Dimensions: nEdges Could be computed internally.
- areaCell - Area in square meters for a given cell of the primary mesh.
  Dimensions: nCells Could be computed internally.
- areaTriangle - Area in square meters for a given triangle of the dual mesh.
  Dimensions: nVertices Could be computed internally.

- edgeNormalVectors - Cartesian coordinates for the normal vector of a given edge.
  Dimensions: 3 * nEdges Could be computed internally.
- edgeTangentVectors - Cartesian coordinates for the tangent vector of a given edge.
  Dimensions: 3 * nEdges Could be computed internally.
- localVerticalUnitVectors - Unit surface normal vectors defined at a given cell.
  Dimensions: 3 * nCells Could be computed internally.
- cellTangentPlane - The two vectors that define a tangent plane at a given cell.
  Dimensions: 3 * 2 * nCells Could be computed internally.
- 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
- kiteAreasOnVertex - The intersection area of areaTriangle with each cell that radiates from a given vertex.
  Dimensions: vertexDegree * nVertices Could be computed internally.
- coeffs_reconstruct - Coefficients to reconstruct velocity vectors at cells centers.
  Dimensions: 3 * maxEdges * nCells Computed internally.