MESH2D GRID GENERATOR DESIGN AND USE

G. P. Flach
F. G. Smith III

REPORT DATE 1/20/2012
DISCLAIMER

This work was prepared under an agreement with and funded by the U.S. Government. Neither the U. S. Government or its employees, nor any of its contractors, subcontractors or their employees, makes any express or implied:
1. warranty or assumes any legal liability for the accuracy, completeness, or for the use or results of such use of any information, product, or process disclosed; or
2. representation that such use or results of such use would not infringe privately owned rights; or
3. endorsement or recommendation of any specifically identified commercial product, process, or service.
Any views and opinions of authors expressed in this work do not necessarily state or reflect those of the United States Government, or its contractors, or subcontractors.

Printed in the United States of America

Prepared for
U.S. Department of Energy
MESH2D GRID GENERATOR DESIGN AND USE

G. P. Flach
F. G. Smith III

REPORT DATE 1/20/2012
## REVIEWS AND APPROVALS

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>G. P. Flach, Co-author</td>
<td>Radiological Performance Assessment</td>
<td></td>
</tr>
<tr>
<td>F. G. Smith, Co-author</td>
<td>Computational Engineering &amp; Sciences</td>
<td></td>
</tr>
<tr>
<td>J. M. Jordan, Peer Reviewer</td>
<td>Computational Engineering &amp; Sciences</td>
<td></td>
</tr>
<tr>
<td>D. A. Crowley, Level 4 Manager</td>
<td>Radiological Performance Assessment</td>
<td></td>
</tr>
<tr>
<td>S. J. Hensel, Level 4 Manager</td>
<td>Computational Engineering &amp; Sciences</td>
<td></td>
</tr>
<tr>
<td>R. S. Aylward, Level 3 Manager</td>
<td>Environmental Restoration Technology</td>
<td></td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>iv</td>
</tr>
<tr>
<td>LIST OF TABLES</td>
<td>iv</td>
</tr>
<tr>
<td>1.0 DESIGN OVERVIEW</td>
<td>1</td>
</tr>
<tr>
<td>2.0 USER SPECIFICATIONS</td>
<td>4</td>
</tr>
<tr>
<td>3.0 EXAMPLE GRID</td>
<td>8</td>
</tr>
<tr>
<td>APPENDIX A - FORTRAN90 SOURCE CODE</td>
<td>A-1</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

Figure 1. Grid zones subdivided in various ways................................................................. 3
Figure 2. Polynomial grid skewing defined by Equations (1) and (2) for $s = 1.5.$ ............ 3
Figure 3. Example two-dimensional grid generated by Mesh2d. ........................................ 9

LIST OF TABLES

Table 1. Superfile format. ........................................................................................................ 4
Table 2. File format for specifying the $x(i)$ grid coordinates.................................................. 5
Table 3. File format for specifying the $y(i,j)$ grid coordinates............................................. 6
Table 4. File format for specifying the $mtyp(i,j)$ material assignments................................. 7
Table 5. superfile for example two-dimensional mesh............................................................. 10
Table 6. $x$Mesh.dat file for example two-dimensional mesh................................................. 10
Table 7. $y$Mesh.dat file for example two-dimensional mesh................................................ 11
Table 8. $mtyp$Mesh.dat file for example two-dimensional mesh........................................... 12
Table 9. example.ply file referenced in $mtyp$Mesh.dat for example two-dimensional mesh. ..................................................................................................................... 12
1.0 DESIGN OVERVIEW

Mesh2d is a Fortran90 program designed to generate two-dimensional structured grids of the form \([x(i), y(i,j)]\) where \([x,y]\) are grid coordinates identified by indices \((i,j)\). The \(x(i)\) coordinates alone can be used to specify a one-dimensional grid. Because the \(x\)-coordinates vary only with the \(i\) index, a two-dimensional grid is composed in part of straight vertical lines. However, the nominally horizontal \(y(i,j_0)\) coordinates along index \(i\) are permitted to undulate or otherwise vary. Mesh2d also assigns an integer material type to each grid cell, \(m typ(i,j)\), in a user-specified manner. The complete grid is specified through three separate input files defining the \(x(i), y(i,j),\) and \(m typ(i,j)\) variations.

The overall mesh is constructed from grid zones that are typically then subdivided into a collection of smaller grid cells. The grid zones usually correspond to distinct materials or larger-scale geometric shapes. The structured grid zones are identified through uppercase indices \((I,J)\). Subdivision of zonal regions into grid cells can be done uniformly, or non-uniformly using either a polynomial or geometric skewing algorithm. Grid cells may be concentrated backward, forward, or toward both ends. Figure 1 illustrates the above concepts in the context of a simple four zone grid.

For non-uniform grid cell distribution, the original polynomial skewing algorithm implemented in Mesh2d is fundamentally defined by

\[
\eta = \begin{cases} 
\xi^s & \text{negative} \\
1 - (1 - \xi)^s & \text{positive} \\
(3 - 2s)\xi + 6(s - 1)\xi^2 + 4(1 - s)\xi^3 & \text{central}
\end{cases}
\]

where \(\xi\) is the normalized distance across the grid zone (ranging from 0 to 1), \(\eta\) is the transformed (skewed) position, and \(s\) is the skewing parameter. "Negative" skewing concentrates grid cells in the negative direction (backward), and "Positive" has the opposite effect. "Central" skewing pushes grid cells away from the center. To avoid excessive skewing, \(s \leq 2\) should be chosen for negative or positive skewing or \(s \leq 1.5\) for central skewing. Figure 2 illustrates the three polynomials defined by Equation (1) for \(s = 1.5\). The figure also shows an example of negative skewing for three grid points initially placed at \(\xi = 0.25, 0.50\) and 0.75. The transformed coordinates become \(\eta = 0.125, 0.354\) and 0.650.

The revised, and generally preferred, polynomial skewing algorithm retains the negative and positive skewing algorithm indicated by Equation (1) but uses a different central skewing approach defined by
\[ \eta = \begin{cases} 
\frac{(2\xi)^s}{2} & \xi \leq 0.5 \\
1 - \frac{(2 - 2\xi)^s}{2} & \xi > 0.5 
\end{cases} \quad (2) \]

Thus the revised polynomial approach uses the negative skewing algorithm over the first half of the zone and the positive algorithm over the second half. This approach has the advantage that the skewing parameter (s) has the same meaning and limit (< 2) for negative, positive, and central skewing. The previous polynomial algorithm is identified by the demoted label "prev" (previous) whereas the revised approach is labeled "poly" in user inputs to Mesh2d.

In addition to the two polynomial-based skewing schemes, non-uniform gridding may be specified through a geometric growth factor. Skewing in the negative direction is specified through the recursive relationship

\[ \Delta \eta_i = s \Delta \eta_{i-1} \quad (3) \]

and positive skewing by

\[ \Delta \eta_i = \Delta \eta_{i-1} / s \quad (4) \]

Central skewing is performed by using negative skewing over the first half of the zone and positive skewing over the second half, similar to the revised polynomial scheme. Both even and odd numbers of grid cells are permitted, and the starting cell size is determined by interval length and a specified total number of cells. When the number of subdivisions is odd, the middle cell spanning the two halves of the zone has the same size as the two adjoining cells.

Material type assignments can be specified through index ranges for grid zones (I,J) or cells (i,j), or through trapezoids with vertical sides or general polygons defined in terms of (x,y) coordinates. Integers used to identify material type can take on arbitrary values and be assigned in any order. With the trapezoid option, any grid cell with a center point (defined as the average of the four corner points) within the trapezoid is included in the selection.

Mesh2d produces grid coordinate, material type, and visualization files in multiple formats and coordinate systems. The software currently supported are the PORFLOW and STADIUM® computational codes, and the Tecplot, VisIt, Paraview, and Gnuplot plotting software. Cartesian and cylindrical coordinate systems are supported for PORFLOW.

A source code listing for Mesh2d is provided in Appendix A. This product includes software produced by Savannah River Nuclear Solutions, LLC under Contract No. DE-AC09-08SR22470 with the United States Department of Energy.
Figure 1. Grid zones subdivided in various ways.

Figure 2. Polynomial grid skewing defined by Equations (1) and (2) for s = 1.5.
2.0 USER SPECIFICATIONS

The top-tier input to Mesh2d is a *superfile* containing only filenames and high-level option specifications, in the order indicated by Table 1. The input file containing the x(i) grid specifications can be assigned an arbitrary name, but is conventionally called *xMesh.dat*. Similarly, *yMesh.dat* and *mtypMesh.dat* are the traditional names for the y(i,j) and mtyp(i,j) specifications. These conventional names will be used for convenience in the remaining discussion.

The formats of *xMesh.dat*, *yMesh.dat* and *mtypMesh.dat* are defined in Table 2 through Table 4. All three files may contain empty lines and/or comment lines denoted by an exclamation point in column 1, as any such lines are discarded during input processing. Material type specifications in *mtypMesh.dat* overwrite any prior specifications.

Table 1. Superfile format.

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Mesh2d inputs:</strong></td>
<td></td>
</tr>
<tr>
<td>e.g. <em>xMesh.dat</em></td>
<td>Input file containing x(i) grid specifications</td>
</tr>
<tr>
<td>e.g. <em>yMesh.dat</em></td>
<td>Input file containing y(i,j) grid specifications</td>
</tr>
<tr>
<td>e.g. <em>mtypMesh.dat</em></td>
<td>Input file containing mtyp(i,j) grid specifications</td>
</tr>
<tr>
<td>xyz or xrt</td>
<td>Cartesian (xyz) or cylindrical (xrt) coordinates for PORFLOW</td>
</tr>
<tr>
<td><strong>Mesh2d outputs:</strong></td>
<td></td>
</tr>
<tr>
<td>e.g. <em>Mesh2d.log</em></td>
<td>Log file</td>
</tr>
<tr>
<td>e.g. <em>Mesh2d.dat</em></td>
<td>PORFLOW high-level grid specification statements</td>
</tr>
<tr>
<td>e.g. <em>COOR.dat</em></td>
<td>PORFLOW coordinates file</td>
</tr>
<tr>
<td>e.g. <em>TYPE.dat</em></td>
<td>PORFLOW material type file</td>
</tr>
<tr>
<td>e.g. <em>Stadium.cor</em></td>
<td>Stadium coordinate file</td>
</tr>
<tr>
<td>e.g. <em>Stadium.ele</em></td>
<td>Stadium element connectivity and material file</td>
</tr>
<tr>
<td>e.g. <em>Mesh2d.tec</em></td>
<td>Tecplot graphics file containing grid data points</td>
</tr>
<tr>
<td>e.g. <em>Geometry.tec</em></td>
<td>Tecplot graphics file containing &quot;TextGeom&quot; geometries outlining grid zones and material boundaries</td>
</tr>
<tr>
<td>e.g. <em>polygon.tec</em></td>
<td>Tecplot graphics file containing &quot;TextGeom&quot; geometries for polygons (if used) to define materials</td>
</tr>
<tr>
<td>e.g. <em>Mesh2d.vts</em></td>
<td>VTK graphics file suitable for VisIt and Paraview plotting software</td>
</tr>
<tr>
<td>e.g. <em>Mesh2d.gnu</em></td>
<td>Gnuplot graphics file</td>
</tr>
</tbody>
</table>
Table 2. File format for specifying the x(i) grid coordinates.

<table>
<thead>
<tr>
<th>Line</th>
<th>Format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 0</td>
<td>scale x0</td>
<td></td>
</tr>
<tr>
<td>Line I, I=1→M</td>
<td>iFlag nx dir scheme skew scale dx</td>
<td>x</td>
</tr>
</tbody>
</table>

**Input**

- **iFlag**: Mode flag (integer = 0 or 1):
  - 0 = Interval size specification for grid zone in field 7 (dx)
  - 1 = End position of grid zone specification in field 7 (x)
- **nx**: Number of grid cells within the grid zone (integer ≥ 1)
- **dir**: Skewing direction (character):
  - n = negative
  - p = positive
  - c = central
  - d = disabled (uniform gridding)
- **scheme**: Skewing scheme (4 character string):
  - prev = (previous) polynomial skewing scheme defined by Equation (1)
  - poly = polynomial skewing scheme defined by Equations (1) and (2)
  - geom = geometric skewing scheme defined by Equations (3) and (4)
- **skew**: Skewing parameter, s, in Equations (1) through (4) (real < 2.0)
- **scale**: Multiplier to dx|x typically used for units conversions (real > 0.0), e.g., scale = 30.48 to convert dx|x in feet to centimeters for grid coordinates.
- **x0**: Line 0: The origin of the x coordinate system (x0) (real)
- **dx|x**: Line I > 0: The size (dx) or ending position (x) of grid zone I (real, dx > 0.0)
Table 3. File format for specifying the \( y(i,j) \) grid coordinates.

<table>
<thead>
<tr>
<th>Line</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line 0</td>
<td>iFlag scale y0|nPts</td>
</tr>
<tr>
<td>Line 0.1 ... 0.( \text{nPts} )</td>
<td>( x ) ( y ) (if iFlag = 1) ... (( \text{nPts} ) instances)</td>
</tr>
<tr>
<td>Line ( J, J=1\rightarrow N ) ... ( J.\text{nPts} )</td>
<td>iFlag ny dir scheme skew scale dy|y|nPts ( \text{(one line per each of } N \text{ grid zones in the } y\text{-direction)} ) ( x ) dy|y (if iFlag = 2|3) ... (( \text{nPts} ) instances)</td>
</tr>
</tbody>
</table>

**Input Description**

- **iFlag**
  - Mode flag (integer = 0 to 3):
  - For Line 0,
    - 0 = Fixed baseline for grid in field 3 (\( y0 \))
    - 1 = Variable baseline for grid in field 3 (\( \text{nPts} \))
  - For Line \( J \),
    - 0 = Fixed interval size specification for grid zone in field 7 (dy)
    - 1 = Fixed end position of grid zone specification in field 7 (y)
    - 2 = Variable interval size specification for grid zone in field 7 (\( \text{nPts} \))
    - 3 = Variable end position of grid zone specification in field 7 (\( \text{nPts} \))

- **ny**
  - Number of grid cells within the grid zone (integer \( \geq 1 \))

- **dir**
  - Skewing direction (character):
    - \( n \) = negative
    - \( p \) = positive
    - \( c \) = central
    - \( d \) = disabled (uniform gridding)

- **scheme**
  - Skewing scheme (4 character string):
    - \( \text{prev} \) = (previous) polynomial skewing scheme defined by Equation (1)
    - \( \text{poly} \) = polynomial skewing scheme defined by Equations (1) and (2)
    - \( \text{geom} \) = geometric skewing scheme defined by Equations (3) and (4)

- **skew**
  - Skewing parameter, \( s \), in Equations (1) through (4) (real \( \leq 2.0 \))

- **scale**
  - Multiplier to dy\|y typically used for units conversions (real \( > 0.0 \)), e.g., scale = 30.48 to convert dy\|y in feet to centimeters for grid coordinates. The scaling factor is also applied to any x values specified (e.g. iFlag = 2\|3).

- **y0\|nPts**
  - Line 0: The origin of the y coordinate system (y0) (real)
  - Line \( i > 0 \) and iFlag = 0\|1: The size (dy) or ending position (y) of grid zone \( J \) (real, dy \( > 0.0 \))
  - Line \( i > 0 \) and iFlag = 2\|3: The number of (x,dy\|y) pairs to follow (integer \( \geq 0 \))

- **x dy\|y**
  - For iFlag = 2\|3, (x,dy\|y) pairs, \( \text{nPts} \) in total, one pair per line
Table 4. File format for specifying the mtyp(i,j) material assignments.

<table>
<thead>
<tr>
<th>Line K</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Line K.2 if iFlag = 2</td>
<td>iFlag im</td>
</tr>
<tr>
<td>Line K.2 if iFlag = 3</td>
<td>polygon (filename)</td>
</tr>
<tr>
<td>repeat as needed</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>iFlag</td>
<td>Mode flag (integer = 0 to 3):</td>
</tr>
<tr>
<td></td>
<td>0 = Material zone specified using grid zone indices (I,J)</td>
</tr>
<tr>
<td></td>
<td>1 = Material zone specified using grid element indices (i,j)</td>
</tr>
<tr>
<td></td>
<td>2 = Material zone specified by trapezoid with vertical sides using (x,y) points</td>
</tr>
<tr>
<td></td>
<td>3 = Material zone specified by general polygon using (x,y) points</td>
</tr>
<tr>
<td>im</td>
<td>xm</td>
</tr>
<tr>
<td></td>
<td>iFlag = 2: left or x-coordinate of trapezoid (real)</td>
</tr>
<tr>
<td></td>
<td>iFlag = 3: read but ignored</td>
</tr>
<tr>
<td>ip</td>
<td>xp</td>
</tr>
<tr>
<td></td>
<td>iFlag = 2: right or x+ coordinate of trapezoid (real)</td>
</tr>
<tr>
<td></td>
<td>iFlag = 3: read but ignored</td>
</tr>
<tr>
<td>jm</td>
<td>ymm</td>
</tr>
<tr>
<td></td>
<td>iFlag = 2: lower left y coordinate of trapezoid (real)</td>
</tr>
<tr>
<td></td>
<td>iFlag = 3: read but ignored</td>
</tr>
<tr>
<td>jp</td>
<td>ypm</td>
</tr>
<tr>
<td></td>
<td>iFlag = 2: lower right y coordinate of trapezoid (real)</td>
</tr>
<tr>
<td></td>
<td>iFlag = 3: read but ignored</td>
</tr>
<tr>
<td>mZone</td>
<td>Material type identification number (arbitrary integer)</td>
</tr>
<tr>
<td>ymp</td>
<td>iFlag = 2: upper left y coordinate of trapezoid (real)</td>
</tr>
<tr>
<td>ypp</td>
<td>iFlag = 2: upper right y coordinate of trapezoid (real)</td>
</tr>
<tr>
<td>polygon</td>
<td>iFlag = 3: filename of polygon with (x,y) vertices. Empty lines and lines with 'P', '#' or a blank in column 1 are ignored. The first vertex is repeated to close the polygon, e.g.,</td>
</tr>
<tr>
<td></td>
<td>(x1,y1)</td>
</tr>
<tr>
<td></td>
<td>(x2,y2)</td>
</tr>
<tr>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>(x1,y1)</td>
</tr>
</tbody>
</table>
3.0 EXAMPLE GRID

Figure 3 illustrates an example two-dimensional grid and material type assignment that utilizes many of the grid specification options described above. Table 5 through Table 9 list the contents of the Mesh2d input files used to generate the example mesh. The plot was generated from the Tecplot output files Mesh2d.tec, Geometry.tec and polygon.tec named in superfile Mesh2d.sup (Table 5).

In the x-direction the grid is composed of 4 zones, each subdivided into 5 elements or cells (Table 6). The uniform gridding is accomplished in the first zone by disabling skewing (dir = d). The same effect can be achieved by setting the skewing parameter to 1.0. Non-uniform gridding is applied to the remaining three zones using negative, central and positive polynomial skewing in succession. Scaling is used to convert user inputs in inches and feet to centimeters for grid coordinates.

The grid is also composed of 4 zones in the y-direction (Table 7). A variable baseline is defined through interpolation and extrapolation using two points. Linear interpolation is used between points and flat line extrapolation before and after the endpoints. Geometric skewing with a growth factor of 1.5 is specified for all zones, but skewing is disabled in the first zone. Note that the first zone is not subdivided (nx = 1).

Every grid cell is initially assigned a material type number of 1 (active line 1 of mtypMesh.dat, Table 8). The next 15 active lines, assign sequential material type numbers to each of the grid zones, thus overwriting the initial assignment. The final active lines overwrite the prior material type assignments using selection by cell indices, a trapezoid, and a general polygon. The polygon points are listed in Table 9. In Figure 3, the trapezoid is depicted in blue and the polygon in green.
Figure 3. Example two-dimensional grid generated by Mesh2d.
Table 5. superfile for example two-dimensional mesh.

|xMesh.dat
|yMesh.dat
|mtypMesh.dat
|xyz
|Mesh2d.log
|Mesh2d.dat
|COOR.dat
|TYPE.dat
|Stadium.cor
|Stadium.ele
|Mesh2d.tec
|Geometry.tec
|polygon.tec
|Mesh2d.vts
|Mesh2d.gnu

Table 6. xMesh.dat file for example two-dimensional mesh.

<table>
<thead>
<tr>
<th>!iFlag key:</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>!iFlag key:</td>
<td></td>
</tr>
<tr>
<td>0=constant interval</td>
<td></td>
</tr>
<tr>
<td>1=constant position</td>
<td></td>
</tr>
<tr>
<td>!dir key:</td>
<td></td>
</tr>
<tr>
<td>p=positive</td>
<td></td>
</tr>
<tr>
<td>n=negative</td>
<td></td>
</tr>
<tr>
<td>c=center</td>
<td></td>
</tr>
<tr>
<td>d=disabled</td>
<td></td>
</tr>
<tr>
<td>!scheme key:</td>
<td></td>
</tr>
<tr>
<td>geom=geometric growth factor</td>
<td></td>
</tr>
<tr>
<td>poly=polynominal transformation</td>
<td></td>
</tr>
<tr>
<td>prev=previous polynomial transformation</td>
<td></td>
</tr>
<tr>
<td>!iFlag</td>
<td>nx</td>
</tr>
<tr>
<td>-------</td>
<td>----</td>
</tr>
<tr>
<td>I=0: origin at x=-5'*30.48cm/ft</td>
<td>30.48 -5</td>
</tr>
<tr>
<td>I=1: step to x=0'*30.48cm/ft</td>
<td>30.48 0</td>
</tr>
<tr>
<td>I=2: add dx=2'*30.48cm/ft</td>
<td>30.48 2</td>
</tr>
<tr>
<td>I=3: add dx=12&quot;*2.54cm/in</td>
<td>2.54 12</td>
</tr>
<tr>
<td>I=4: add dx=2'*30.48cm/ft</td>
<td>30.48 2</td>
</tr>
</tbody>
</table>
Table 7. yMesh.dat file for example two-dimensional mesh.

```
!iFlag key:
!       0=constant interval  
!       1=constant position  
!       2=variable interval  
!       3=variable position  
!dir key:
!       p=positive  
!       n=negative  
!       c=center  
!       d=disabled  
!scheme key:
!       geom=geometric growth factor  
!       poly=polynomial transformation  
!       prev=previous polynomial transformation  

!iFlag  nx  dir  scheme  skew  scale  dy|y|nPts  (x,y)
1       40  2    geom  30.48  2    !J=0: variable y0*30.48cm/ft
       -2  0    ! (x1,y1)
       +2 -2    ! (x2,y2)
0       5   d    geom  1.5    30.48  2    !J=1: add dy=2'*30.48cm/ft
1       5   p    geom  1.5    30.48  4    !J=2: step to y=4'*30.48cm/ft
2       5   c    geom  1.5    30.48  3    !J=3: add variable dy*30.48cm/ft
       -2  1    ! (x1,dy1)
       0  2    ! (x2,dy2)
       2  1    ! (x3,dy3)
3       5   n    geom  1.5    30.48  3    !J=3: step to variable y*30.48cm/ft
       -2 10    ! (x1,y1)
       0  9    ! (x2,y2)
       2 10    ! (x3,y3)
```
Table 8.  mtypMesh.dat file for example two-dimensional mesh.

<table>
<thead>
<tr>
<th>!iFlag</th>
<th>im</th>
<th>ip</th>
<th>xm</th>
<th>xp</th>
<th>jm</th>
<th>ymm</th>
<th>jp</th>
<th>ypm</th>
<th>ypp</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>99</td>
<td>1</td>
<td>99</td>
<td>1</td>
<td>!selection of entire grid by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>6</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>7</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>8</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
<td>9</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>10</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>11</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>12</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>4</td>
<td>13</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>14</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
<td>15</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>16</td>
<td>!selection of one zone by zone indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>17</td>
<td>!selection of four cells by cell indices</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>-50</td>
<td>-10</td>
<td>120</td>
<td>130</td>
<td>18</td>
<td>!selection of cells by trapezoid</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>-99</td>
<td>-99</td>
<td>-99</td>
<td>-99</td>
<td>19</td>
<td>!selection of cells by polygon</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 9.  example.ply file referenced in mtypMesh.dat for example two-dimensional mesh.

<table>
<thead>
<tr>
<th>Polygon file</th>
</tr>
</thead>
<tbody>
<tr>
<td>-120 180</td>
</tr>
<tr>
<td>-30 180</td>
</tr>
<tr>
<td>-75 280</td>
</tr>
<tr>
<td>-120 180</td>
</tr>
</tbody>
</table>
Appendix A - Fortran90 source code

File Mesh2d.f90  -----------------------------------------------------------

program Mesh2d

!Copyright © 2012, Savannah River Nuclear Solutions, LLC
!
!Distribution Statement:
!This computer software has been developed under sponsorship of the U.S.
!Department of Energy. Any further distribution or use by anyone other than the
!named licensee of this software package or any data contained therein, unless
!otherwise specifically provided for, is prohibited without the approval of the
!Office of Scientific and Technical Information. Requests for DOE developed
!computer software shall be referred to the Energy Science and Technology
!Software Center at the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831 1020.
!This product includes software produced by Savannah River Nuclear Solutions,
!LLC under Contract No. DE-AC09-08SR22470 with the United States Department of
!Energy.
!
!Disclaimer:
!This material was prepared as an account of work sponsored by an agency of the
!United States Government. Neither the United States Government nor the United
!States Department of Energy, nor any of their employees, makes any warranty,
!express or implied, or assumes any legal liability or responsibility for the
!accuracy, completeness, or usefulness of any information, apparatus, product,
or process disclosed, or represents that its use would not infringe privately
!owned rights.
!
!iFlag for mesh generation:
!  0  constant thickness
!  1  constant elevation
!  2  variable thickness
!  3  variable elevation
!
!iFlag for material type:
!  0  zone indices from x and y mesh input
!  1  element indices
!  2  (x,y) trapezoidal zone with vertical sides
!  3  selection by polygon
!
! direction key for mesh skewing:
!  p  positive
!  n  negative
!  c  centered
!  d  disabled

use global
use stuff
use Bspline
use Polyg

implicit none

integer, parameter :: LineLength=80, FilenameLength=80, PathLength=500, &
                      FullLength=580

integer, parameter :: isup=10, &
                      iinp=11, &
                      jinp=12, &
                      minp=13, &
                      ilog=21, &
integer :: iostat_flag, iunt, inp
integer :: nx, ny, npx, npx, nnx, nny,nex,ney
integer :: nex2, ney2
integer :: i, j, k, im, ip, jm, jp
integer :: i2, j2
integer :: iFlag, nPts, iZone, jZone, mZone, iIndex, jIndex, nargs, ipos

integer, dimension(:) , allocatable :: iz, jz
integer, dimension(:,:), allocatable :: mtyp
real(LONG) :: tmp, dx, dy, yint, scale, skew
real(LONG) :: tmp1, tmp2, tmp3, tmp4, xm, xp, ym, yp, ymm, ypm, ymp, ypp
real(LONG) :: xoff, yoff, sint, cost, srse, srsn, xCenter, yCenter
real(LONG), dimension(:) , allocatable :: xPts, yPts
real(LONG), dimension(:) , allocatable :: x, x2
real(LONG), dimension(:,:), allocatable :: y, y2
character(len=FilenameLength) :: Filename
character(len=LineLength) :: Line
character(len=PathLength) :: Path
character(len=FullLength) :: Fullpath
character(len=1) :: dir
character(len=3) :: xyz
character(len=4) :: scheme
character(len=6) :: color
logical :: exit_flag, is3d, inplg, first

data is3d/.false./
data color/"GREEN"/

!OPEN FILES AND READ INPUT DATA
nargs = command_argument_count()
if (nargs .eq. 1) then
  call get_command_argument(1,Fullpath)
ipos = scan(Fullpath, "/", .true.)
Path = Fullpath(:ipos)
Filename = Fullpath(ipos+1:)
call chdir(trim(Path))
iunt = isup
open (unit=iunt, file=trim(Filename), status='old', action='read')
else if (nargs .eq. 0) then
  iunt = 5
else
  stop "unexpected number of command arguments"
end if

do inp=iinp, minp
    read (iunt,'(a)') Filename
    open (unit=inp, file=trim(Filename), status='old', action='read')
end do

read(iunt,'(a)') xyz

do inp=ilog, ityp
    read (iunt,'(a)') Filename
    open (unit=inp, file=trim(Filename), status='unknown', action='write')
end do

write(ilog,'("!number of command lines arguments: ",i2)') narg

!STADIUM files . . .
read (iunt,'(a)') Filename
open (unit=icoor, file=trim(Filename), status='unknown', action='write')

read (iunt,'(a)') Filename
open (unit=ielem, file=trim(Filename), status='unknown', action='write')

!Plotting files . . .
do inp=itec, ignu
    read (iunt,'(a)') Filename
    open (unit=inp, file=trim(Filename), status='unknown', action='write')
end do

!CREATE X MESH
i = 0; i2 = 0

do
    read (iinp,'(a)', iostat=iostat_flag) Line
    call IostatCheck (iostat_flag, "iinp", exit_flag)
    if (exit_flag) exit !end-of-file
    if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle !ignore comment or blank line
    read (Line,*) tmp
    exit
end do

do
    read (iinp,'(a)', iostat=iostat_flag) Line
    call IostatCheck (iostat_flag, "iinp", exit_flag)
    if (exit_flag) exit !end-of-file
    if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle !ignore comment or blank line
    read (Line,*) iFlag, nx, dir, scheme, skew, scale, tmp
    i = i + nx; i2 = i2 + 1
end do

nex = i; nex2 = i2
nx = i + 1
npx = i + 2
allocate (x(0:nex)); allocate (x2(0:nex2))
allocate (iz(0:nex))
x = 0; x2 = 0
iz = -999

rewind(iinp)
i = 0; i2 = 0
iZone = 0
do
  read (iinp,'(a)', iostat=iostat_flag) Line
  call IostatCheck (iostat_flag, "iinp", exit_flag)
  if (exit_flag) exit !end-of-file
  if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle !ignore comment or blank
  line
  read (Line,*) scale, tmp
  x(0) = scale*tmp; x2(0) = scale*tmp
  exit
end do

do
  read (iinp,'(a)', iostat=iostat_flag) Line
  call IostatCheck (iostat_flag, "iinp", exit_flag)
  if (exit_flag) exit !end-of-file
  if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
  read (Line,*) iFlag, nx, dir, scheme, skew, scale, tmp
  if (iFlag .eq. 0) then
    dx = scale*tmp
  else if (iFlag .eq. 1) then
    dx = scale*tmp - x(i)
  else
    stop "invalid iFlag value"
  end if
  iZone = iZone + 1
  if (scheme .eq. "prev") then
    call prev (x, nx, nex, iz, iZone, i, dx, dir, skew)
  else if (scheme .eq. "poly") then
    call poly (x, nx, nex, iz, iZone, i, dx, dir, skew)
  else if (scheme .eq. "geom") then
    call geom (x, nx, nex, iz, iZone, i, dx, dir, skew)
  else
    stop "invalid scheme"
  end if
  x2(i2+1) = x2(i2) + dx
  i = i + nx; i2 = i2 + 1
end do

write(ilog,'(a,3i4)') "!nex, nnx, npx: ", nex, nnx, npx
write(ilog,'(a,3i4)') "!nex2: ", nex2

!CREATE Y MESH
j = 0; j2 = 0

do
  read (jinp,'(a)', iostat=iostat_flag) Line
  call IostatCheck (iostat_flag, "jinp", exit_flag)
  if (exit_flag) exit !end-of-file
  if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
  read (Line,*) iFlag, scale, tmp
  if (iFlag .eq. 1) then
    nPts = int(tmp)
    do k=1,nPts
      read (jinp,'(a)', iostat=iostat_flag) Line
      call IostatCheck (iostat_flag, "jinp", exit_flag)
      if (exit_flag) stop "not enough points" !end-of-file
      if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
    end do
  end if
end do
end if
exit
end do

do
read (jinp,'(a)', iostat=iostat_flag) Line
call IostatCheck (iostat_flag, "jinp", exit_flag)
if (exit_flag) exit !end-of-file
if (Line(1:1) .eq. '!'. .or. len(trim(Line)) .eq. 0) cycle
read (Line,*) iFlag, ny, dir, scheme, skew, scale, tmp
if (iFlag .eq. 2 .or. iFlag .eq. 3) then
nPts = int(tmp)
do k=1,nPts
read (jinp,'(a)', iostat=iostat_flag) Line
if (Line(1:1) .eq. '!'. .or. len(trim(Line)) .eq. 0) cycle
end do
end if
j = j + ny; j2 = j2 + 1
end do

ney = j; ney2 = j2
nny = j + 1
npy = j + 2
allocate (y(0:nex,0:ney)); allocate (y2(0:nex2,0:ney2))
allocate (jz(0:ney))
y = 0; y2 = 0
jz = -999
rewind(jinp)
j = 0; j2 = 0
jZone = 0
do
read (jinp,'(a)', iostat=iostat_flag) Line
if (exit_flag) exit !end-of-file
if (Line(1:1) .eq. '!'. .or. len(trim(Line)) .eq. 0) cycle
read (Line,*) iFlag, scale, tmp
if (iFlag .eq. 0) then
y(0:nex,0) = scale*tmp; y2(0:nex2,0) = scale*tmp
else if (iFlag .eq. 1) then
nPts = int(tmp)
allocate(xPts(nPts))
allocate(yPts(nPts))
do x=1,nPts
read (jinp,'(a)', iostat=iostat_flag) Line
if (exit_flag) stop "not enough points" !end-of-file
if (Line(1:1) .eq. '!'. .or. len(trim(Line)) .eq. 0) cycle
read (Line,*) xPts(k), yPts(k)
xPts(k) = scale*xPts(k)
yPts(k) = scale*yPts(k)
end do
do i=0,nex
y(i,0) = yint
end do
do i2=0,nex2
do i=0,nex
y2(i2,i) = yint
deallocate(xPts)
deallocate(yPts)
else
  stop "invalid iFlag value"
end if
exit
end do

do
  read (jinp,'(a)', iostat=iostat_flag) Line
  call IostatCheck (iostat_flag, "jinp", exit_flag)
  if (exit_flag) exit !end-of-file
  if (Line(1:1).eq. '!' .or. len(trim(Line)) .eq. 0) cycle
  read (Line,*) iFlag, ny, dir, scheme, skew, scale, tmp
  if (iFlag.eq. 0) then
    dy = scale*tmp
  else if (iFlag .eq. 1) then
    continue
  else if (iFlag .eq. 2 .or. iFlag .eq. 3) then
    nPts = int(tmp)
    allocate(xPts(nPts))
    allocate(yPts(nPts))
    do k=1,nPts
      read (jinp,'(a)', iostat=iostat_flag) Line
      call IostatCheck (iostat_flag, "jinp", exit_flag)
      if (exit_flag) stop "not enough points" !end-of-file
      if (Line(1:1).eq. '!' .or. len(trim(Line)) .eq. 0) cycle
      read (Line,*) xPts(k), yPts(k)
      xPts(k) = scale*xPts(k)
      yPts(k) = scale*yPts(k)
    end do
  else
    stop "invalid iFlag value"
  end if
  jZone = jZone + 1
  do i=0,nex
    if (iFlag .eq. 1) then
      dy = scale*tmp - y(i,j)
    else if (iFlag .eq. 2) then
      call Interp(nPts,xPts,yPts,x(i),yint)
      dy = yint
    else if (iFlag .eq. 3) then
      call Interp(nPts,xPts,yPts,x(i),yint)
      dy = yint - y(i,j)
    end if
    if (scheme .eq. "prev") then
      call prev (y(i,0:ney), ny, ney, jz, jZone, j, dy, dir, skew)
    else if (scheme .eq. "poly") then
      call poly (y(i,0:ney), ny, ney, jz, jZone, j, dy, dir, skew)
    else if (scheme .eq. "geom") then
      call geom (y(i,0:ney), ny, ney, jz, jZone, j, dy, dir, skew)
    else
      stop "invalid scheme"
    end if
  end do
  do i2=0,nex2
    if (iFlag .eq. 1) then
      dy = scale*tmp - y2(i2,j2)
    else if (iFlag .eq. 2) then
      continue
    else if (iFlag .eq. 3) then
      nPts = int(tmp)
      allocate(xPts(nPts))
      allocate(yPts(nPts))
      do k=1,nPts
call Interp(nPts,xPts,yPts,x2(i2),yint)
dy = yint
else if (iFlag .eq. 3) then
    call Interp(nPts,xPts,yPts,x2(i2),yint)
dy = yint - y2(i2,j2)
end if
y2(i2,j2+1) = y2(i2,j2) + dy
end do

j = j + ny; j2 = j2 + 1
if (iFlag .eq. 2 .or. iFlag .eq. 3) then
    deallocate(xPts)
deallocate(yPts)
end if
end do

write(ilog,'(a,3i4)') '!ney, nny, npy: ', ney, nny, npy
write(ilog,'(a,3i4)') '!ney2: ', ney2
write(ilog,'(a,3i4)') '!nex*ney: ', nex*ney
write(ilog,'(a,3i4)') '!nnx*nny: ', nnx*nny
write(ilog,'(a,3i4)') '!npx*npy: ', npx*npy

!CREATE MATERIAL TYPES
allocate (mtyp(0:nex,0:ney))
mtyp = -999
do
    read (minp,'(a)',iostat=iostat_flag) Line
    call IostatCheck (iostat_flag,"minp", exit_flag)
    if (exit_flag) exit !end-of-file
    if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
    read (Line,*) iFlag, tmp1, tmp2, tmp3, tmp4, mZone
    if (iFlag .le. 1) then
        im = nint(tmp1)
        ip = nint(tmp2)
        jm = nint(tmp3)
        jp = nint(tmp4)
    else if (iFlag .eq. 2) then
        xm  = tmp1
        xp  = tmp2
        ymm = tmp3
        ypm = tmp4
        read (minp,'(a)',iostat=iostat_flag) Line
        call IostatCheck (iostat_flag,"minp", exit_flag)
        if (exit_flag) exit !end-of-file
        if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
        read (Line,*) ymp, ypp
        !WRITE POLYGON PLOTTING FILE FOR TECFLOT (in model coords.)
        write (ite3,'(a)') "GEOMETRY X=0, Y=0, T=LINE, M=GRID, C=BLUE"
        write (ite3,'(1x,i3)') 1
        write (ite3,'(1x,i6)') 5
        write (ite3,'(1x,3(2x,f9.2))') xm, ymm
        write (ite3,'(1x,3(2x,f9.2))') xp, ypm
        write (ite3,'(1x,3(2x,f9.2))') xp, ypp
        write (ite3,'(1x,3(2x,f9.2))') xm, ymp
        write (ite3,'(1x,3(2x,f9.2))') xm, ymm
    else if (iFlag .eq. 3) then
        read (minp,'(a)',iostat=iostat_flag) Line
        call IostatCheck (iostat_flag,"minp", exit_flag)
        if (exit_flag) exit !end-of-file
        if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
        Filename = trim(adjustl(Line))
        open (unit=iply, file=Filename, status='old', action='read')
        first = .true.
end if

do j=0,ney-1
  do i=0,nex-1
    if (iFlag .le. 1) then
      if (iFlag .eq. 0) then
        iIndex = iz(i)
        jIndex = jz(j)
      else if (iFlag .eq. 1) then
        iIndex = i+1
        jIndex = j+1
      end if
      if (iIndex.ge.im .and. iIndex.le.ip .and. &
          jIndex.ge.jm .and. jIndex.le.jp ) then
        mtyp(i,j) = mZone
      end if
    else if (iFlag .eq. 2) then
      xCenter = 0.50*(x(i) + x(i+1))
      yCenter = 0.25*(y(i,j  ) + y(i+1,j  ) + &
                      y(i,j+1) + y(i+1,j+1) )
      ym = (ypm-ymm)/(xp-xm) *(xCenter-xm) + ymm
      yp = (ypp-ymy)/(xp-xm) *(xCenter-xm) + ymp
      if (xCenter.ge.xm .and. xCenter.le.xp .and. &
          yCenter.ge.ym .and. yCenter.le.yp ) then
        mtyp(i,j) = mZone
      end if
    else if (iFlag .eq. 3) then
      xCenter = 0.50*(x(i) + x(i+1))
      yCenter = 0.25*(y(i,j  ) + y(i+1,j  ) + &
                      y(i,j+1) + y(i+1,j+1) )
      srse = xCenter
      srsn = yCenter
      xoff=0; yoff=0
      call polygon (srse,srsn,first,iply,ite3,xoff,yoff,sint,cost,is3d,color,inplg)
      if (inplg) mtyp(i,j) = mZone
    end if
  end do
end do

call WritePorflow (imsh,icor,ityp,xyz,x,y,mtyp,nex,ney,npx,npy)
call WriteStadium (icoor,ielem,xyz,x,mtyp,nex,ney,nnx)
call WriteTecplot (itec,ite2,x,y,mtyp,nex,ney,nnx,nnx, &
                        x2,y2,nex2,ney2)
call WriteParaview (ipar,x,y,mtyp,nex,ney)
call WriteGnuplot (ignu,x,y,mtyp,nex,ney)
end program Mesh2d

!// /////////////////////////////////////////////////////////////////////
subroutine prev (x, nx, nex, iz, iZone, i, dx, dir, skew)
  use global
  implicit none
  integer :: nx, nex
  integer :: ii, iZone, i
  integer :: iz(0:nex)
real(LONG) :: dx, skew, xi, factor, a, b, c, d
real(LONG) :: x(0:nex)

character(len=1) :: dir

do ii=1,nx
   xi = real(ii) / real(nx)
   
   if (dir .eq. "n") then
      if (skew .lt. one) stop "specify skew >=1"
      factor = xi**skew
   else if (dir .eq. "p") then
      if (skew .lt. one) stop "specify skew >=1"
      factor = one - (one - xi)**skew
   else if (dir .eq. "c") then
      if (xi .le. half) then
         factor = (two*xi)**skew / two
      else
         factor = one - (two - two*xi)**skew / two
      end if
   else
      stop "invalid dir"
   end if

   x(i+ii) = x(i) + dx * factor
   iz(i+ii-1) = iZone
end do
end subroutine prev

!----------------------------------------------------------------------
subroutine poly (x, nx, nex, iz, iZone, i, dx, dir, skew)

use global

implicit none

integer :: nx, nex
integer :: ii, iZone, i
integer :: iz(0:nex)

real(LONG) :: dx, skew, xi, factor
real(LONG) :: x(0:nex)

character(len=1) :: dir

if (skew .lt. one) stop "specify skew >=1"
do ii=1,nx
   xi = real(ii) / real(nx)
   
   if (dir .eq. "n") then
      factor = xi**skew
   else if (dir .eq. "p") then
      factor = one - (one - xi)**skew
   else if (dir .eq. "c") then
      if (xi .le. half) then
         factor = (two*xi)**skew / two
      else
         factor = one - (two - two*xi)**skew / two
      end if
   else
      stop "invalid dir"
   end if

   x(i+ii) = x(i) + dx * factor
   iz(i+ii-1) = iZone
end do
end if
else if (dir .eq. "d") then
    factor = xi
else
    stop "invalid dir"
end if

x(i+ii) = x(i) + dx * factor
iz(i+ii-1) = iZone
end do

end subroutine poly

subroutine geom (x, nx, nex, iz, iZone, i, dx, dir, skew)
use global
implicit none
integer :: nx, nex
integer :: ii, iZone, i
integer :: iz(0:nex)
real(LONG) :: dx, skew, sum, dx1
real(LONG) :: x(0:nex)
character(len=1) :: dir

if (dir .eq. "n") then
    call skewer (nx, skew, dx, dx1, one)
call gridrgt (nx, skew, dx1, x, iz, nex, i, iZone)
else if (dir .eq. "p") then
    call skewer (nx, skew, dx, dx1, one)
call gridlft (nx, skew, dx1, x, iz, nex, i, iZone)
else if (dir .eq. "c") then
    if (nx/2*2 .eq. nx) then
        call skewer (nx/2, skew, dx, dx1, half)
call gridrgt (nx/2, skew, dx1, x, iz, nex, i, iZone)
call gridlft (nx/2, skew, dx1, x, iz, nex, i + nx/2, iZone)
    else
        sum = zero
do ii=1,(nx-1)/2
        sum = sum + skew**(ii-1)
    end do
    sum = sum + 0.5 * skew**((nx-1)/2-1)
dx1 = 0.5 * dx / sum

call gridrgt ((nx-1)/2, skew, dx1, x, iz, nex, i, iZone)
x(i+(nx-1)/2+1) = x(i+(nx-1)/2) + dx1 * skew**((nx-1)/2-1)
iz(i+(nx-1)/2) = iZone

call gridlft ((nx-1)/2, skew, dx1, x, iz, nex, i + (nx-1)/2 + 1, iZone)
end if
else if (dir .eq. "d") then
    dx1 = dx / nx
call gridrgt (nx, one, dx1, x, iz, nex, i, iZone)
else
    stop "invalid dir"
end if

end subroutine geom

!/***********************************************************************************/
subroutine skewer (nx, skew, dx, dx1, factor)

use global

implicit none

integer :: ii, nx
real(LONG) :: dx, skew, sum, dx1, factor

sum = zero
do ii=1,nx
    sum = sum + skew**(ii-1)
end do

dx1 = factor * dx / sum

end subroutine skewer

!/***********************************************************************************/
subroutine gridrgt (nx, skew, dx1, x, iz, nex, i, iZone)

use global

implicit none

integer :: ii, nx, nex, i, iZone
real(LONG) :: skew, dx1
integer :: iz(0:nex)
real(LONG) :: x(0:nex)

do ii=1,nx
    x(i+ii)   = x(i+ii-1) + dx1 * skew**(ii-1)
    iz(i+ii-1) = iZone
end do

end subroutine gridrgt

!/***********************************************************************************/
subroutine gridlft (nx, skew, dx1, x, iz, nex, i, iZone)

use global

implicit none

integer :: ii, nx, nex, i, iZone
real(LONG) :: skew, dx1
integer :: iz(0:nex)
real(LONG) :: x(0:nex)

do ii=1,nx
    x(i+ii)   = x(i+ii-1) + dx1 * skew**(nx-ii)
    iz(i+ii-1) = iZone
end do
end subroutine gridlft

!******************************************************************************
subroutine WritePorflow (imsh, icor, ityp, xyz, x, y, mtyp, nex, ney, npx, npy)

use global
implicit none
integer :: imsh, icor, ityp
integer :: npx, npy, nex, ney
integer :: i, j, ii, jj
integer :: mtyp(0:nex,0:ney)
integer, dimension(:,,:), allocatable :: ntyp
real(LONG) x(0:nex), y(0:nex,0:ney)
character(len=3) :: xyz

!WRITE PORFLOW MESH SIZE FILE
if (xyz .eq. 'xyz') then
  write(imsh,'(a,i3,a,i3,a)') "GRID is ",npx," by ",npy," NODEs"
  write(imsh,'(a)') 'COORdinates X Y from file "../../Common/COOR.dat"
  write(imsh,'(a,i3,a,i3,a)') "LOCAte ID=DOMAIN as nodes (1,1) to (", &
    npx, 
    ", 
    npy, 
    ")
write(imsh,'(a,i3,a,i3,a)') "LOCAte ID=INSIDE as nodes (1,1) to (", &
    npx, 
    ", 
    npy, 
    ")", FIELD only"
else if (xyz .eq. 'xrt') then
  write(imsh,'(a,i3,a,i3,a)') "GRID is ",npy," by ",npx," NODEs"
  write(imsh,'(a)') 'COORdinates CYLIndrical X R from file "../../Common/COOR.dat"
  write(imsh,'(a,i3,a,i3,a)') "LOCAte ID=DOMAIN as nodes (1,1) to (", &
    npy, 
    ", 
    npx, 
    ")"
write(imsh,'(a,i3,a,i3,a)') "LOCAte ID=INSIDE as nodes (1,1) to (", &
    npy, 
    ", 
    npx, 
    ")", FIELD only"
else
  stop "invalid xyz value"
end if

!WRITE PORFLOW COORDINATE FILE
if (xyz .eq. 'xyz') then
  write (icor,'(2(g14.6))') ((x(i), y(i,j), i=0,nex), j=0,ney)
else if (xyz .eq. 'xrt') then
  write (icor,'(2(g14.6))') ((y(i,j), x(i), j=0,ney), i=0,nex)
else
  stop "invalid xyz value"
end if

!WRITE PORFLOW MTYP FILE
allocate(ntyp(npx,npy))
do j=1,npy
   jj = j - 2
   jj = max(jj,0)
   jj = min(jj,ney-1)
do i=1,npx
   ii = i - 2
   ii = max(ii,0)
   ii = min(ii,nex-1)
   ntyp(i,j) = mtyp(ii,jj)
end do
end do
if (xyz .eq. 'xyz') then
   write (ityp,'(16i5)') ((ntyp(i,j), i=1,npx), j=1,npy)
else if (xyz .eq. 'xrt') then
   write (ityp,'(16i5)') ((ntyp(i,j), j=1,npy), i=1,npx)
else
   stop "invalid xyz value"
end if
deallocate(ntyp)
end subroutine WritePorflow

!SUBROUTINE WriteStadium (icoor,ielem,xyz,x,mtyp,nex,ney,nnx)
use global
implicit none
integer :: icoor, ielem
integer :: i, im, ip, nex, ney, nnx
integer :: mtyp(0:nex,0:ney)
real(LONG) x(0:nex)
character(len=3) :: xyz
character(len=1) :: tab
data tab/
                   
!WRITE STADIUM COORDINATE FILE (1D for now)
if (xyz .eq. 'xyz' .or. xyz .eq. 'xrt') then
   write(icoor,'(a,i1)') "Dimensions: ", 1
   write(icoor,'(a,i6)') "Nodes: ", nnx
   write(icoor,'(i6,a,a, 3(a,i6))') (i+1, tab, x(i), i=0,nex)
else
   stop "invalid xyz value"
end if

!WRITE STADIUM CONNECTIVITY FILE
if (xyz .eq. 'xyz' .or. xyz .eq. 'xrt') then
   write(ielem,'(a,i6)') "Nelements: ", nex
   do i=1,nex
      im = i; ip = i+1
      write(ielem,'(i6,a,a, 3(a,i6))') i, tab,"L2", tab,im, tab,ip, tab,mtyp(i-1,0)
   end do
else
   stop "invalid xyz value"
end if
end subroutine WriteStadium

!////////////////////////////////////////////////////////////////////
subroutine WriteTecplot (itec,ite2,x,y,mtyp,nex,ney,nnx,nny, &
 x2,y2,nex2,ney2)

use global
implicit none
integer :: itec, ite2
integer :: nex, ney, nex2, ney2, nnx, nny
integer :: i, j, i2, j2
integer :: mtyp(0:nex,0:ney)
real(LONG)  x(0:nex),  y(0:nex,0:ney)
real(LONG) x2(0:nex2), y2(0:nex2,0:ney2)

logical writeln
!
WRITE TECPLT FILE
write(itec,'(a)') 'TITLE = "PORFLOW Grid"
write(itec,'(a)') 'VARIABLES = "x" "y" "mtyp"
write(itec,991) "Corner", nnx, nny, "POINT", &
   ", DT=(DOUBLE,DOUBLE,SHORTINT)"
write(itec,'(2(g14.6),i7)') ((x(i), y(i,j), mtyp(i,j), i=0,nex), j=0,ney)
!
WRITE TEXTGEOM FILE
do i2=0,nex2
   do j2=1,ney2
      write(ite2,'(a)') 'GEOMETRY X=0, Y=0, CS=GRID, T=LINE, C=BLACK'
      write(ite2,'(i1)') 1
      write(ite2,'(i3)') 2
      write(ite2,'(2(g14.6))') x2(i2), y2(i2,j2-1)
      write(ite2,'(2(g14.6))') x2(i2), y2(i2,j2  )
   end do
end do
!
do j2=0,ney2
   do i2=1,nex2
      write(ite2,'(a)') 'GEOMETRY X=0, Y=0, CS=GRID, T=LINE, C=BLACK'
      write(ite2,'(i1)') 1
      write(ite2,'(i3)') 2
      write(ite2,'(2(g14.6),i7)') x2(i2-1), y2(i2-1,j2)
      write(ite2,'(2(g14.6),i7)') x2(i2  ), y2(i2  ,j2)
   end do
end do
!
do i=0,nex
   do j=1,ney
      writeLine = .false.
      if (i.eq.0 .or. i.eq.nex) then
         writeLine = .true.
      else if (mtyp(i-1,j-1) .ne. mtyp(i,j-1)) then
         writeLine = .true.
      end if
      if (writeLine) then
         write(ite2,'(a)') 'GEOMETRY X=0, Y=0, CS=GRID, T=LINE, C=RED'
         write(ite2,'(i1)') 1
         write(ite2,'(i3)') 2
      end if
   end do
end do
write(ite2,'(2(g14.6),i7)') x(i), y(i,j-1)
write(ite2,'(2(g14.6),i7)') x(i), y(i,j)
end if
end do
end do

do j=0,ney
  do i=1,nex
    writeLine = .false.
    if (j.eq.0 .or. j.eq.ney) then
      writeLine = .true.
      else if (mtyp(i-1,j-1) .ne. mtyp(i-1,j)) then
      writeLine = .true.
    end if
    if (writeLine) then
      write(ite2,'(a)') 'GEOMETRY X=0, Y=0, CS=GRID, T=LINE, C=RED'
      write(ite2,'(i1)') 1
      write(ite2,'(i3)') 2
      write(ite2,'(2(g14.6),i7)') x(i-1), y(i-1,j)
      write(ite2,'(2(g14.6),i7)') x(i ), y(i ,j)
    end if
  end do
end do

!FORMATS
991 format ('ZONE T="',a,'"', &
  ', 'I=',i3,   &
  ', 'J=',i3,   &
  ', 'F=',a, &
a)
end subroutine WriteTecplot

!////////////////////////////////////////////////////////////////////
subroutine WriteParaview (ipar,x,y,mtyp,nex,ney)
use global
implicit none
integer :: ipar
integer :: nex, ney
integer :: i, j
integer :: mtyp(0:nex,0:ney)
real(LONG) x(0:nex), y(0:nex,0:ney)

!WRITE PARAVIEW XML DATA FILE
write(ipar,'(a)') '&
  '<VTKFile type="StructuredGrid" version="0.1" byte_order="LittleEndian">'
write(ipar,'(a,i6,a,i6,a,i6,a)') &
  '<StructuredGrid WholeExtent="0 ', &
    nex, &
    ' 0 ', &
    ney, &
    ' 0 ', &
  '1, &
  '"'>'
write(ipar,'(a,i6,a,i6,a,i6,a)') &
write(ipar,'(a)') '      <Points Scalars="Grid">'
write(ipar,'(a)') &
   <DataArray type="Float32" Name="Points" NumberOfComponents="3"
format="ascii">'
write(ipar,'(3(g14.6))') ((x(i), y(i,j), zero, i=0,nex), j=0,ney)
write(ipar,'(3(g14.6))') ((x(i), y(i,j), one, i=0,nex), j=0,ney)
write(ipar,'(a)') '      </Points>'
write(ipar,'(a)') '      <PointData Scalars="Grid">'
write(ipar,'(a)') &
   <DataArray type="Float32" Name="PointData" NumberOfComponents="1"
format="ascii">'
write(ipar,'(g14.6)') ((zero, i=0,nex), j=0,ney)
write(ipar,'(g14.6)') ((one, i=0,nex), j=0,ney)
write(ipar,'(a)') '      </PointData>'
write(ipar,'(a)') '      <CellData Scalars="Grid">'
write(ipar,'(a)') &
   <DataArray type="Float32" Name="CellData" NumberOfComponents="1"
format="ascii">'
write(ipar,'(i6)') ((mtyp(i-1,j-1), i=1,nex), j=1,ney)
write(ipar,'(a)') '      </CellData>'
write(ipar,'(a)') '      </Piece>'
write(ipar,'(a)') '</StructuredGrid>'
write(ipar,'(a)') '</VTKFile>'
end subroutine WriteParaview

!////////////////////////////////////////////////////////////////////
subroutine WriteGnuplot (ignu,x,y,mtyp,nex,ney)
use global
implicit none
integer :: ignu
integer :: nex, ney
integer :: i, j
integer :: mtyp(0:nex,0:ney)
real(LONG) x(0:nex), y(0:nex,0:ney)
!WRITE GNUPLOT DATA FILE
write(ignu,'(a)') "#x y mtyp"
do j=0,ney
    write(ignu,'(2(g14.6),i7)') (x(i), y(i,j), mtyp(i,j), i=0,nex)
    write(ignu,'(a)') ""
end do
end subroutine WriteGnuplot

---

File global.f90

module global
!
!Copyright © 2012, Savannah River Nuclear Solutions, LLC
!
!Distribution Statement:
!This computer software has been developed under sponsorship of the U.S.
!Department of Energy. Any further distribution or use by anyone other than the
!named licensee of this software package or any data contained therein, unless
!otherwise specifically provided for, is prohibited without the approval of the
!Office of Scientific and Technical Information. Requests for DOE developed
!computer software shall be referred to the Energy Science and Technology
!Software Center at the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831 1020.
!This product includes software produced by Savannah River Nuclear Solutions,
!LLC under Contract No. DE-AC09-08SR22470 with the United States Department of
!Energy.
!
!Disclaimer:
!This material was prepared as an account of work sponsored by an agency of the
!United States Government. Neither the United States Government nor the United
!States Department of Energy, nor any of their employees, makes any warranty,
!express or implied, or assumes any legal liability or responsibility for the
!accuracy, completeness, or usefulness of any information, apparatus, product,
or process disclosed, or represents that its use would not infringe privately
!owned rights.

    integer, parameter :: LONG=SELECTED_REAL_KIND(9,99)
    real(LONG), parameter :: zero=0_LONG, one=1_LONG, two=2_LONG, half=0.5_LONG
end module global

---

File Module.f90

module stuff
!
!Copyright © 2012, Savannah River Nuclear Solutions, LLC
!
!Distribution Statement:
!This computer software has been developed under sponsorship of the U.S.
!Department of Energy. Any further distribution or use by anyone other than the
!named licensee of this software package or any data contained therein, unless
!otherwise specifically provided for, is prohibited without the approval of the
!Office of Scientific and Technical Information. Requests for DOE developed
!computer software shall be referred to the Energy Science and Technology
!Software Center at the Office of Scientific and Technical Information,
P.O. Box 62, Oak Ridge, TN 37831 1020.
!This product includes software produced by Savannah River Nuclear Solutions,
!LLC under Contract No. DE-AC09-08SR22470 with the United States Department of
!Energy.
!
!Disclaimer:
!This material was prepared as an account of work sponsored by an agency of the
!United States Government. Neither the United States Government nor the United
!States Department of Energy, nor any of their employees, makes any warranty,
contains

subroutine IostatCheck (iostat_flag, string, exit_flag)

integer , intent(in) :: iostat_flag
character, intent(in) :: string
logical, intent(out) ::  exit_flag

if      (iostat_flag .eq. 0) then !OK
  exit_flag = .false.
else if (iostat_flag .lt. 0) then !END-OF-FILE
  exit_flag = .true.
else if (iostat_flag .gt. 0) then !ERROR
  write (*,'(2a)') '*** READ ERROR *** ', string
  stop

end if

end subroutine IostatCheck

end module stuff

File Bspline.f90

contains

subroutine Interp(n,x,y,xint,yint)
use global

implicit none

integer, parameter :: k=2     !second-order splines / linear interpolation
integer, intent(in) :: n      !number of data points
integer :: i

real(LONG), dimension(:) :: x, y
real(LONG), dimension(:,), allocatable :: t
real(LONG), intent(in) :: xint
real(LONG), intent(out) :: yint

!DEFINE SPLINE KNOTS
allocate(t(n+2))
do i=2,n+1
   t(i) = x(i-1)
end do
t(1) = t(2)
t(n+2) = t(n+1)
!INTERPOLATE USING B- SPLINES
yint = zero
do i=1,n
   yint = yint + y(i)*b0(i,k,xint,t)
end do

!EXTEND SPLINES OUTSIDE DATA RANGE
if (xint .lt. x(1)) yint = y(1)
if (xint .ge. x(n)) yint = y(n)
deallocate(t)
return
end subroutine Interp

function b0(i,k,x,t)
! (0)
! B (x) function = b(i,k,x,t)
! i,k
!
! Input:
! i = ith B-spline
! k = B-spline order
! x = B-spline argument
! t = vector of B-spline knots
!
! Output:
! b = B-spline value

use global

implicit none

integer, intent(in) :: i, k
integer :: incr, ii, kk
real(LONG), intent(in) :: x
real(LONG), dimension(:,), intent(in) :: t
real(LONG), dimension(0:3) :: bs
real(LONG) :: b0

if (x.lt.t(i) .or. x.ge.t(i+k)) then
    b0 = zero
    return
end if

do incr=0,k-1
    ii = i + incr
    if (x.lt.t(ii) .or. x.ge.t(ii+1)) then
        bs(incr) = zero
    else
        bs(incr) = one
    end if
end do

do kk=2,k
    do incr=0,k-kk
        ii = i + incr
        if (bs(incr) .ne. zero) then
            bs(incr) = (x - t(ii))/(t(ii+kk-1) - t(ii))*bs(incr)
        end if
    end do
    if (bs(incr+1) .ne. zero) then
        bs(incr) = bs(incr) &
                    + (t(ii+kk) - x)/(t(ii+kk) - t(ii+1))*bs(incr+1)
    end if
end do

b0 = bs(0)
return
end function b0
end module Bspline

File Polygon.f90 -----------------------------------------------------------

module polyg

!Copyright © 2012, Savannah River Nuclear Solutions, LLC
!
!Distribution Statement:
!This computer software has been developed under sponsorship of the U.S.
!Department of Energy. Any further distribution or use by anyone other than the
!named licensee of this software package or any data contained therein, unless
!otherwise specifically provided for, is prohibited without the approval of the
!Office of Scientific and Technical Information. Requests for DOE developed
!computer software shall be referred to the Energy Science and Technology
!Software Center at the Office of Scientific and Technical Information,
!P.O. Box 62, Oak Ridge, TN 37831 1020.
!This product includes software produced by Savannah River Nuclear Solutions,
!LLC under Contract No. DE-AC09-08SR22470 with the United States Department of
!Energy.
!
!Disclaimer:
!This material was prepared as an account of work sponsored by an agency of the
!United States Government. Neither the United States Government nor the United
!States Department of Energy, nor any of their employees, makes any warranty,
!express or implied, or assumes any legal liability or responsibility for the
!accuracy, completeness, or usefulness of any information, apparatus, product,
or process disclosed, or represents that its use would not infringe privately
owned rights.

use stuff

implicit none

contains

subroutine polygon (x,y,first,ioin,ioout,xoff,yoff,sint,cost,is3d,color,inplg)

integer, parameter :: LONG=SELECTED_REAL_KIND(9,99)
integer, parameter :: nvmax=1000
integer :: ioin, ioout
integer :: nv, i, icross
real(LONG) :: x,y, xmin,ymin,xmax,ymax, xc, xm,xp,ym,yp
real(LONG) :: xoff,yoff,sint,cost
real(LONG), dimension(nvmax) :: xv,yv, xq,yq
character(len=6) :: color
logical :: first, inplg, is3d

save nv,xv,yv, xmin,ymin,xmax,ymax

inplg = .false.

!IF FIRST CALL TO POLYGON, READ-IN POLYGON DATA
if (first) then
    first = .false.
    call input (ioin,ioout, xoff,yoff,sint,cost, is3d, color, &
               nv,xv,yv, xmin,ymin,xmax,ymax, nvmax)
end if

!MAKE QUICK CHECK FOR POINT FAR FROM POLYGON
if (x .lt. xmin) return
if (y .lt. ymin) return
if (x .gt. xmax) return
if (y .gt. ymax) return

!TRANSLATE COORDINATES SO POINT IN QUESTION IS ORIGIN

do i=1,nv+1
    xq(i) = xv(i) - x
    yq(i) = yv(i) - y
end do

!CHECK EACH EDGE TO IF IT CROSSES RAY FROM ORIGIN TO x=+inf (x axis)

icross = 0
do i=1,nv
    xm = xq(i)
xp = xq(i+1)
ym = yq(i)
yp = yq(i+1)
    if (yp.gt.0 .and. ym.le.0 .or. &
        ym.gt.0 .and. yp.le.0 ) then
xc = (xm*yp - xp*ym)/(yp - ym)
if (xc .gt. 0) icross = icross + 1
end if
end do

!ODD NUMBER OF CROSSINGS IMPLIES POINT IN POLYGON
if (icross .ne. icross/2*2 .and. icross .gt. 0) inplg = .true.
end subroutine polygon

!----------------------------------------------------------------------
subroutine input (ioin,ioout, xoff,yoff,sint,cost, is3d,color, &
vx,vy, xmin,ymin,xmax,ymax, nvmax)

integer, parameter :: LONG=SELECTED_REAL_KIND(9,99)
integer :: nvmax
integer :: ioin, ioout
integer :: iostat_flag
integer :: nv, i

real(LONG), parameter :: big=1.e20_LONG
real(LONG) :: xmin,ymin,xmax,y, xmodel,ymodel, xoff,yoff,sint,cost, x,y
real(LONG), dimension(nvmax) :: vx,vy
character(len=80) :: line
character(len=6) :: color
logical :: is3d, exit_flag

xmin = +big
ymin = +big
xmax = -big
ymax = -big

!READ POLYGON VERTICES
nv = 0
do
read (ioin,'(a)',iostat=iostat_flag) line
call IostatCheck (iostat_flag,"ioin", exit_flag)
if (exit_flag) exit !end of file
if (line(1:1).eq.'#' .or. line(1:1).eq.'P' .or. line(1:1).eq.' '     ) cycle
read (line,*) x,y
nv = nv + 1
vx(nv) = x
vy(nv) = y
xmin = min(xmin,x)
ymin = min(ymin,y)
xmax = max(xmax,x)
ymax = max(ymax,y)
end do

!WRITE POLYGON PLOTTING FILE FOR TECPLOT (in model coords.)
if (is3d) then
write (ioout,903) color
903 format ('GEOMETRY X=0, Y=0, Z=0, T=LINE3D, M=GRID, C=',a,/,'1')
else
write (ioout,902) color
902 format ('GEOMETRY X=0, Y=0, T=LINE, M=GRID, C=',a,/,',1')
end if

write (ioout,'(1x,i6)') nv

do i=1,nv
  xmodel = (xv(i) - xoff)*cost + (yv(i) - yoff)*sint
  ymodel = -(xv(i) - xoff)*sint + (yv(i) - yoff)*cost
  if (is3d) then
    write (ioout,'(1x,3(2x,f9.2))') xmodel, ymodel, 350.
  else
    write (ioout,'(1x,3(2x,f9.2))') xmodel, ymodel
  end if
end do

!SET nv TO NUMBER OF VERTICES

nv = nv - 1

end subroutine input

end module polyg