Contract No:

This document was prepared in conjunction with work accomplished under Contract No. DE-AC09-08SR22470 with the U.S. Department of Energy (DOE) Office of Environmental Management (EM).

Disclaimer:

We put science to work.™

Savannah River National Laboratory™

OPERATED BY SAVANNAH RIVER NUCLEAR SOLUTIONS

A U.S. DEPARTMENT OF ENERGY NATIONAL LABORATORY • SAVANNAH RIVER SITE • AIKEN, SC

MESH2D Grid Generator Design and Use

G. P. Flach

October 31, 2017

SRNL-STI-2012-00005 Revision 1

SRNL.DOE.GOV

DISCLAIMER

Printed in the United States of America

Prepared for
U.S. Department of Energy

SRNL-STI-2012-00005
Revision 1

Keywords: *Performance Assessment Grid Generation*

Retention: *Permanent*

MESH2D Grid Generator Design and Use

G. P. Flach

October 31, 2017

Prepared for the U.S. Department of Energy under contract number DE-AC09-08SR22470.

REVIEWS AND APPROVALS

AUTHORS:

_____

G. P. Flach, Environmental Modeling Date

TECHNICAL REVIEW:

_____

T. Hang, Environmental Modeling, Reviewed per E7 2.60 Date

APPROVAL:

_____

D. A. Crowley, Manager, Environmental Modeling Date

_____

L. T. Reid, Director, Environmental Restoration Technology Date

ACKNOWLEDGEMENTS

Frank G. Smith III, now retired from SRNL, performed most of the Fortran coding associated with this Version 2.0 upgrade to Mesh2d.

REVISIONS

Revision 0 Issued January 20, 2012.

Revision 1 Revised user instructions to incorporate new options and added an additional example to illustrate the use of the new options.

EXECUTIVE SUMMARY

Mesh2d is a Fortran90 program originally 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). x-coordinates depending only on index i implies strictly vertical x-grid lines, whereas the y-grid lines can undulate. Mesh2d also assigns an integer material type to each grid cell, $mtyp(i,j)$, in a user-specified manner. The complete grid is specified through three separate input files defining the $x(i)$, $y(i,j)$, and $mtyp(i,j)$ variations. Since the original development effort, Mesh2d has been extended to more general two-dimensional structured grids of the form $[x(i,j), y(i,j)]$.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

SRNS	Savannah River Nuclear Solutions

1.0 Software Design

Mesh2d is a Fortran90 program initially 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, $mtyp(i,j)$, in a user-specified manner. The complete grid is specified through three separate input files defining the $x(i)$, $y(i,j)$, and $mtyp(i,j)$ variations. Since Flach and Smith (2012), Mesh2d has been extended to more general two-dimensional structured grids of the form $[x(i,j), y(i,j)]$.

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 1-1 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

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

where ξ is the normalized distance across the grid zone (ranging from 0 to 1), η 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 1-2 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 ξ = 0.25, 0.50 and 0.75. The transformed coordinates become η = 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

$$\eta = \begin{cases} \dfrac{(2\xi)^s}{2} & \xi \leq 0.5 \\ 1 - \dfrac{(2-2\xi)^2}{2} & \xi > 0.5 \end{cases} \qquad (2)$$

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

$$\Delta\eta_i = s\Delta\eta_{i-1} \tag{3}$$

and positive skewing by

$$\Delta\eta_i = \Delta\eta_{i-1}/s \tag{4}$$

Central skewing is performed by using negative skewing over the first half of the zone and positive skewing over the second half, like 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.

A source code listing for Version 2.0 of Mesh2d is provided in Appendix A. This product includes software produced by Savannah River Nuclear Solutions (SRNS), LLC under Contract No. DE-AC09-08SR22470 with the United States Department of Energy.

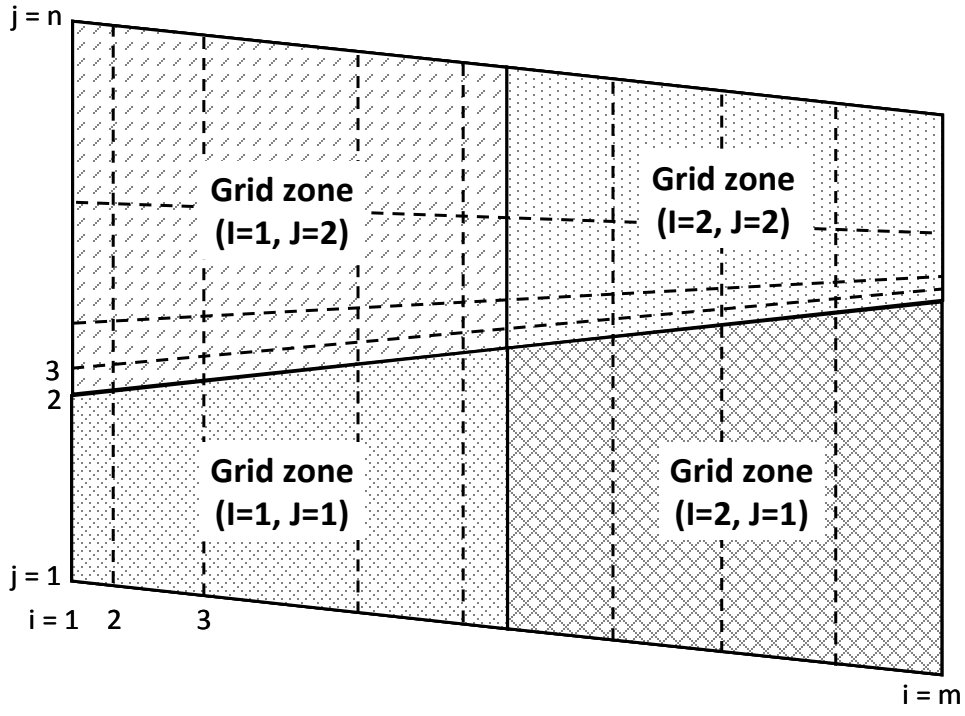


Figure 1-1. Grid zones subdivided in various ways.

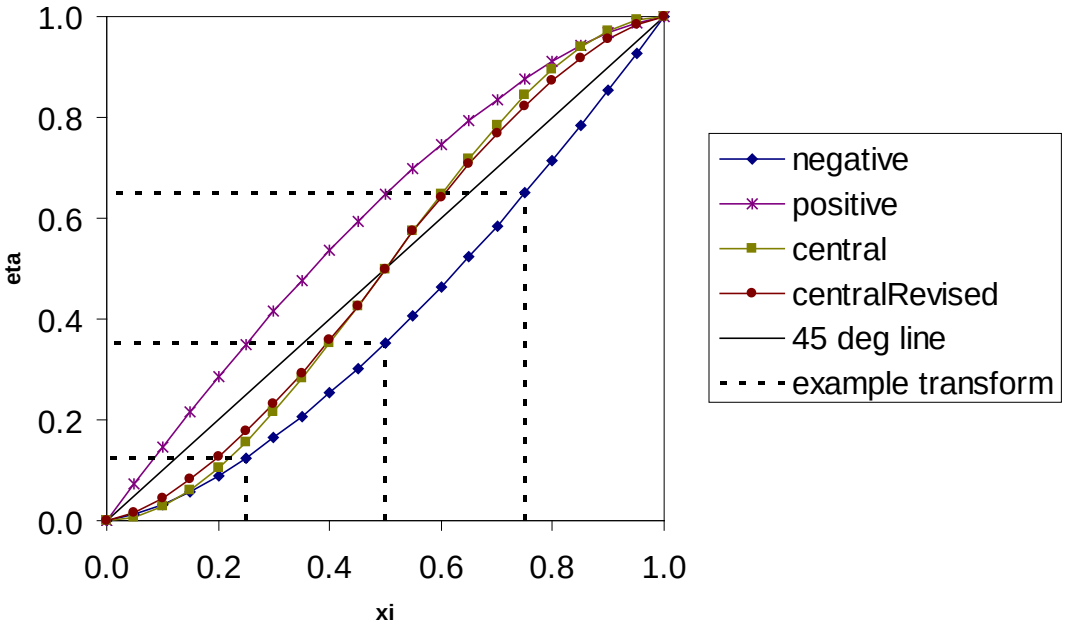


Figure 1-2. Polynomial grid skewing defined by Equations (1) and (2) for s = 1.5.

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 2-1. The input file containing the $x(i)$ or $x(i,j)$ 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 2-2 through Table 2-4. Options 4, 5 and 6 are new features of Mesh2d that extend potential x-coordinate grid variation from $x(i)$ (strictly vertical x-grid lines) to $x(i,j)$. The x- and y-coordinate grid generation options are similar, the difference being that y-direction grid zones can be defined through interpolation from a table of (x,y) values. 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 2-1. Superfile format.

Input	Description
	Mesh2d inputs:
e.g. *xMesh.dat*	Input file containing $x(i)$ or $x(i,j)$ grid specifications
e.g. *yMesh.dat*	Input file containing $y(i,j)$ grid specifications
e.g. *mtypMesh.dat*	Input file containing $mtyp(i,j)$ grid specifications
xyz or *xrt*	Cartesian (*xyz*) or cylindrical (*xrt*) coordinates for PORFLOW
	Mesh2d outputs:
e.g. *Mesh2d.log*	Log file
e.g. *Mesh2d.dat*	PORFLOW high-level grid specification statements
e.g. *COOR.dat*	PORFLOW coordinates file
e.g. *TYPE.dat*	PORFLOW material type file
e.g. *Stadium.cor*	Stadium coordinate file
e.g. *Stadium.ele*	Stadium element connectivity and material file
e.g. *Mesh2d.tec*	Tecplot graphics file containing grid data points
e.g. *Geometry.tec*	Tecplot graphics file containing "TextGeom" geometries outlining grid zones and material boundaries
e.g. *polygon.tec*	Tecplot graphics file containing "TextGeom" geometries for polygons (if used) to define materials
e.g. *Mesh2d.vts*	VTK graphics file suitable for VisIt and Paraview plotting software
e.g. *Mesh2d.gnu*	Gnuplot graphics file

Table 2-2. File format for specifying the $x(i)$ or $x(i,j)$ grid coordinates.

Line	**Format**
Line 0	iFlag scale x|nPts|x_1, x_2, …, x_N
Line 0.1 … 0.nPts	J x (if iFlag=5) … (nPts instances)
Line I, I=1→M	iFlag nx dir scheme skew scale dx|x|nPts|x_1, x_2, …, x_N (one line per each of M grid zones in the x-direction)
Line I.1 … I.npts	J dx|x (if iFlag=4|5) … (nPts instances)
Input	**Description**
iFlag	Mode flag: Line 0: iFlag =1, 5 or 6 Line I: iFlag =0, 1, 4, 5 or 6 Modes: 0 = Fixed thickness of mesh zone (dx) 1 = Fixed end position of mesh zone (x) 4 = Variable thickness by selected zone indices (J,dx) $x(I,J\leq J_1) = x(I-1,J\leq J_1) + dx_1$ $x(I,J_2) = x(I-1,J_2) + dx_2$ … $x(I,J\geq J_{nPts}) = x(I-1,J\geq J_{nPts}) + dx_{nPts}$ 5 = Variable end position by selected zone indices (J,x) $x(I,J\leq J_1) = x_1$ $x(I,J_2) = x_2$ … $x(I,J\geq J_{nPts}) = x_{nPts}$ 6 = Variable end position by all indices (N from yMesh.dat) $x(I,1) = x_1$ … $x(I,N) = x_N$
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.
dx|x	Line 0: Starting position of left boundary (x) (real) Line I > 0: The thickness (dx) or ending position (x) of grid zone I (real, $dx\|x_i-x_{i-1} > 0.0$)

Table 2-3. File format for specifying the $y(i,j)$ grid coordinates.

Line	**Format**
Line 0	iFlag scale y|nPts|y_1, y_2, …, y_M
Line 0.1 … 0.nPts	x|I y (if iFlag = 3|5) … (nPts instances)
Line J, J=1→N	iFlag ny dir scheme skew scale dy|y|nPts|y_1, y_2, …, y_M (one line per each of N grid zones in the y-direction)
Line J.1 … J.nPts	x|I dy|y (if iFlag = 2|3|4|5) … (nPts instances)
Input	**Description**
iFlag	Mode flag: Line 0: iFlag =1, 3, 5 or 6 Line I: iFlag =0, 1, 2, 3, 4, 5 or 6 Modes: 0 = Fixed thickness of mesh zone (dy) 1 = Fixed end position of mesh zone (y) 2 = Variable thickness interpolated from table of (x_k,dy_k) values $y(I,J) = y(I,J-1) + dy$ @ $\min[\max(x(I,J), x_{k=1}), x_{k=nPts}]$ 3 = Variable end position interpolated from table of (x,y) values $y(I,J) = y$ @ $\min[\max(x(I,J), x_{k=1}), x_{k=nPts}]$ 4 = Variable thickness by selected indices (I,dy) $y(I \le I_1,J) = y(I \le I_1,J-1) + dy_1$ $y(I_2,J) = y(I_2,J-1) + dy_2$ … $y(I \ge I_{nPts},J) = y(I \ge I_{nPts},J-1) + dy_{nPts}$ 5 = Variable end position by selected indices (I,y) $y(I \le I_1,J) = y_1$ $y(I_2,J) = y_2$ … $y(I \ge I_{nPts},J) = y_{nPts}$ 6 = Variable end position by all indices (M from xMesh.dat) $y(1,J) = y_1$ … $y(M,J) = y_M$
ny	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 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 (iFlag = 2|3).
dy|y	Line 0: Starting position(s) of bottom boundary (y) (real) Line J > 0: The thickness (dy) or ending position (y) of grid zone J (real, $dy \| y_i - y_{i-1} > 0.0$)

Table 2-4. File format for specifying the $mtyp(i,j)$ material assignments.

Line	**Format**
Line K	iFlag im|xm ip|xp jm|ymm jp|ypm mZone
Line K.2 if iFlag = 2	ymp ypp
Line K.2 if iFlag = 3	polygon (filename)
repeat as needed	
Input	**Description**
iFlag	Mode flag (integer = 0 to 3): 0 = Material zone specified using grid zone indices (I,J) 1 = Material zone specified using grid element indices (i,j) 2 = Material zone specified by trapezoid with vertical sides using (x,y) points 3 = Material zone specified by general polygon using (x,y) points
im|xm	iFlag = 0, 1: starting zone (I) or element (i) index in x-direction (integer) iFlag = 2: left or x- coordinate of trapezoid (real) iFlag = 3: read but ignored
ip|xp	iFlag = 0, 1: ending zone (I) or element (i) index in x-direction (integer) iFlag = 2: right or x+ coordinate of trapezoid (real) iFlag = 3: read but ignored
jm|ymm	iFlag = 0, 1: starting zone (J) or element (j) index in y-direction (integer) iFlag = 2: lower left y coordinate of trapezoid (real) iFlag = 3: read but ignored
jp|ypm	iFlag = 0, 1: ending zone (J) or element (j) index in y-direction (integer) iFlag = 2: lower right y coordinate of trapezoid (real) iFlag = 3: read but ignored
mZone	Material type identification number (arbitary integer)
ymp	iFlag = 2: upper left y coordinate of trapezoid (real)
ypp	iFlag = 2: upper right y coordinate of trapezoid (real)
polygon	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., (x_1,y_1) (x_2,y_2) … (x_1,y_1)

3.0 Example Grids

The example grids described in this section illustrate how to use the various grid generation options offered by Mesh2d. Example 1 invokes Modes 0 through 3 available in the preceding version of Mesh2d. Example 2 illustrates new Modes 4 through 6.

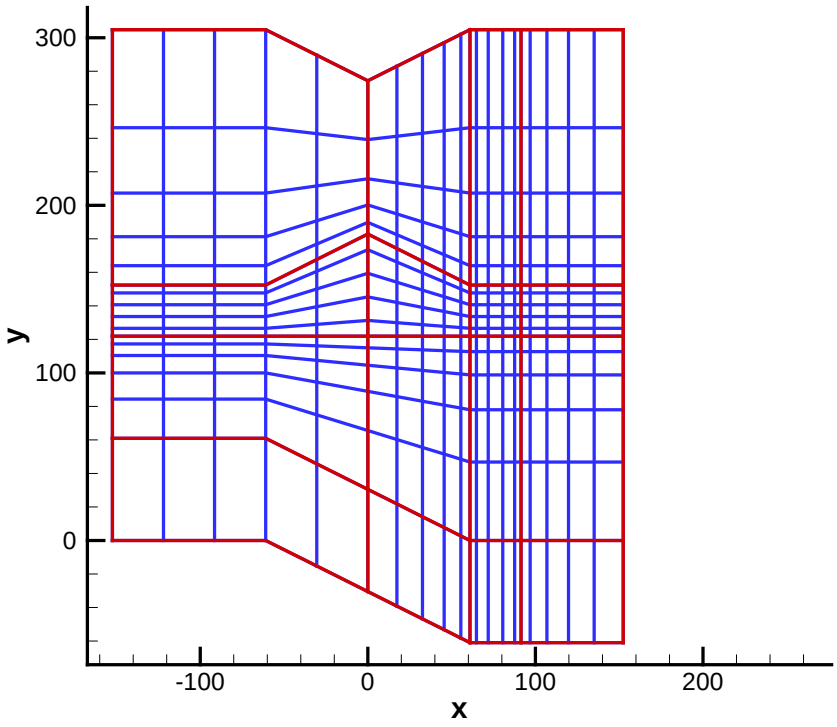
3.1 Example 1

Figure 3-1 illustrates an example two-dimensional grid and material type assignment that utilizes many of the grid specification options described in Section 2.0. Table 3-1 through Table 3-5 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 3-1).

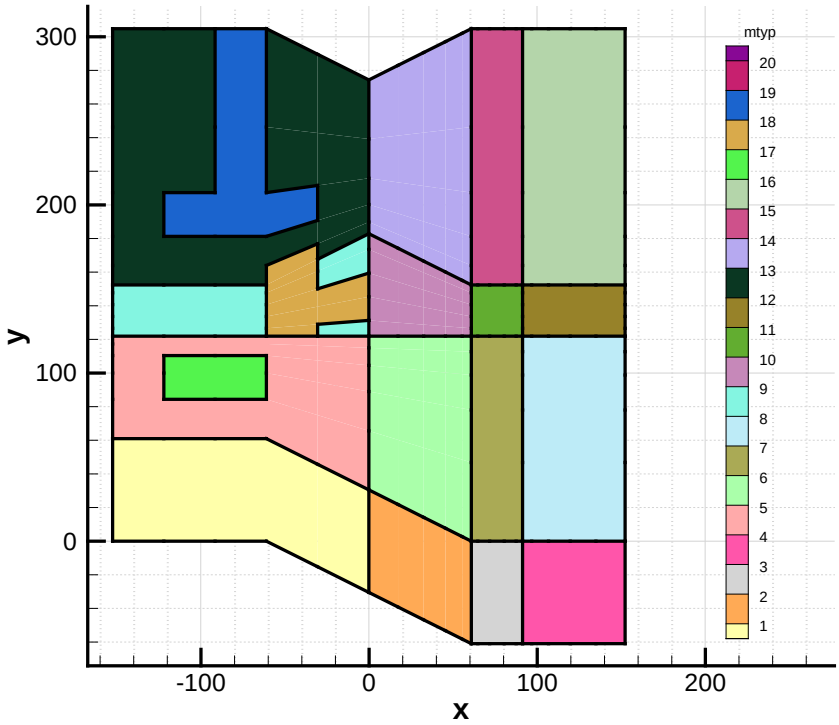
In the x-direction the grid is composed of 4 zones, each subdivided into 5 elements or cells (Table 3-2). The uniform gridding is accomplished in the first zone by disabling skewing (*dir = d*). The same effect can be achieved by setting the skewing parameter to 1.0. Non-uniform gridding is applied to 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 3-3). 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 3-4). 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 3-5. In Figure 3-1, the trapezoid is depicted in blue and the polygon in green.



(a)



(b)

Figure 3-1. Two-dimensional grid generated by Mesh2d, Example 1: (a) zones and refined grid, (b) materials.

Table 3-1. *superfile* for two-dimensional mesh Example 1.

```
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 3-2. *xMesh.dat* file for two-dimensional mesh Example 1.

```
!iFlag key:
!     0=constant interval
!     1=constant position
!dir key:
!     p=positive
!     n=negative
!     c=center
!     d=disabled
!scheme key:
!     geom=geometric growth factor
!     poly=polynominal transformation
!     prev=previous polynomial transformation
!
!iFlag      nx    dir   scheme      skew  scale dx|x
1                          30.48 -5    !I=0: origin at x=-5'*30.48cm/ft
1     5     d     prev  1.5   30.48 0     !I=1: step to x=0'*30.48cm/ft
0     5     p     poly  1.5   30.48 2     !I=2: add dx=2'*30.48cm/ft
0     5     c     poly  1.5   2.54  12    !I=3: add dx=12"*2.54cm/in
0     5     n     poly  1.5   30.48 2     !I=4: add dx=2'*30.48cm/ft
```

Table 3-3. ***yMesh.dat*** **file for two-dimensional mesh Example 1.**

```
!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=polynominal transformation
!     prev=previous polynomial transformation
!
!iFlag      nx    dir   scheme      skew  scale dy|y|nPts   (x,y)
3                            30.48 2                 !J=0: variable y0*30.48cm/ft
                                        -2    0     !     (x1,y1)
                                        +2    -2    !     (x2,y2)
0     1     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)
```

Table 3-4. *mtypMesh.dat* file for two-dimensional mesh Example 1.

```
!iFlag key:
! 0=zone indices from x and y mesh input
! 1=element indices
! 2=(x,y) trapezoidal zone with vertical sides
! 3=polygon
!
!iFlag im|xm  ip|xp  jm|ymm jp|ypm mZone
!                      ymp     ypp
0      1      99     1      99     1      !selection of entire grid by zone indices
0      2      2      1      1      2      !selection of one zone by zone indices
0      3      3      1      1      3      !selection of one zone by zone indices
0      4      4      1      1      4      !selection of one zone by zone indices
!
0      1      1      2      2      5      !selection of one zone by zone indices
0      2      2      2      2      6      !selection of one zone by zone indices
0      3      3      2      2      7      !selection of one zone by zone indices
0      4      4      2      2      8      !selection of one zone by zone indices
!
0      1      1      3      3      9      !selection of one zone by zone indices
0      2      2      3      3      10     !selection of one zone by zone indices
0      3      3      3      3      11     !selection of one zone by zone indices
0      4      4      3      3      12     !selection of one zone by zone indices
!
0      1      1      4      4      13     !selection of one zone by zone indices
0      2      2      4      4      14     !selection of one zone by zone indices
0      3      3      4      4      15     !selection of one zone by zone indices
0      4      4      4      4      16     !selection of one zone by zone indices
!
1      2      3      3      4      17     !selection of four cells by cell indices
2      -50    -10    120    130    18     !selection of cells by trapezoid
                     170    150
3      -99    -99    -99    -99    19     !selection of cells by polygon
example.ply
```

Table 3-5. *example.ply* file referenced in *mtypMesh.dat* for two-dimensional mesh Example 1.

```
Polygon file
-120  180
-30   180
-75   280
-120  180
```

3.2 Example 2

The second example illustrates the new features of Mesh2d: Modes 4, 5 and 6. Figure 3-2(a) displays the grid and Figure 3-2(b) displays the underlying grid zones and material assignments. The contents of the key input files are shown in Table 3-6 through Table 3-8. The *superfile* is like Example 1 and not listed herein.

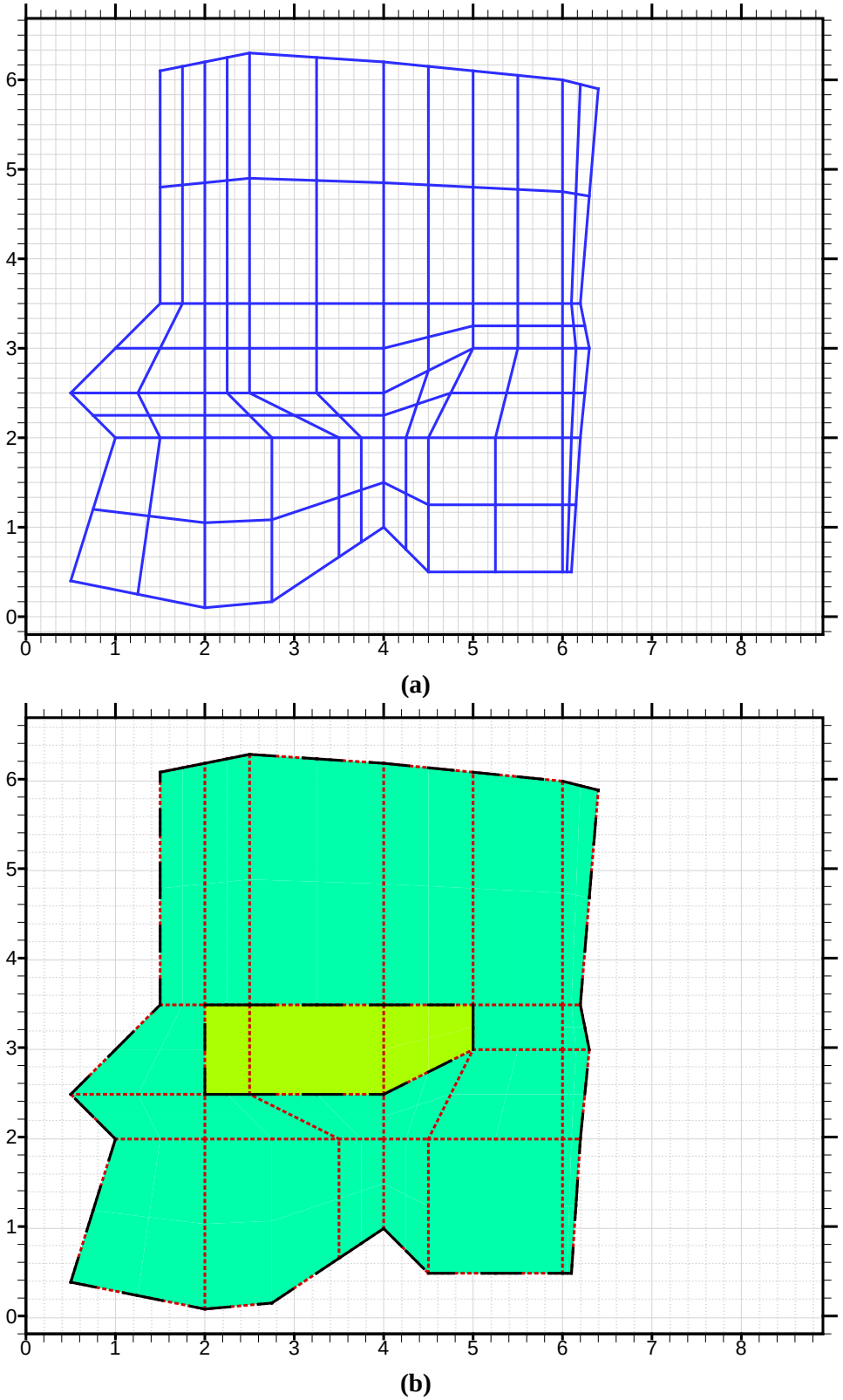


Figure 3-2. Two-dimensional grid generated by Mesh2d, Example 2: (a) refined grid, (b) zones and materials.

Table 3-6. *xMesh.dat* file for two-dimensional mesh Example 2.

```
!iFlag key:
!     0=constant interval
!     1=constant position
!     4=variable interval by zonal indices
!     5=variable position by zonal indices
!     6=direct specification
!dir key:
!     p=positive
!     n=negative
!     c=center
!     d=disabled
!scheme key:
!     geom=geometric growth factor
!     poly=polynominal transformation
!     prev=previous polynomial transformation
!
!iFlag      nx    dir   scheme      skew  scale dx|x|nPts   (j,x)
5                              1     3
                                           1     1.0
                                           2     0.5
                                           3     1.5
1     2     d     geom  1.5   1     2
4     2     d     geom  1.5   1     2
                                           1     1.5
                                           2     0.5
1     2     d     geom  1.5   1     4
5     2     d     geom  1.5   1     2
                                           1     4.5
                                           2     5
1     2     d     geom  1.5   1     6
6     2     d     geom  1.5   1     6.1  6.2  6.3  6.2  6.4
```

Table 3-7. *yMesh.dat* file for two-dimensional mesh Example 2.

```
!iFlag key:
!     0=constant interval
!     1=constant position
!     2=variable interval by coordinates
!     3=variable position by coordinates
!     4=variable interval by zonal indices
!     5=variable position by zonal indices
!     6=direct specification
!dir key:
!     p=positive
!     n=negative
!     c=center
!     d=disabled
!scheme key:
!     geom=geometric growth factor
!     poly=polynominal transformation
!     prev=previous polynomial transformation
!
!iFlag      nx    dir   scheme      skew  scale dy|y|nPts   (x,y)
3                              1     4
                                           0     0.5
                                           2.5   0
                                           4     1.0
                                           4.5   0.5
5     2     d     geom  1.5   1     2
                                           1     2
                                           2     2
3     2     d     geom  1.5   1     2
                                           4     2.5
                                           5     3.0
4     2     d     geom  1.5   1     2
                                           3     1
                                           4     0.5
6     2     d     geom  1.5   1     6.1  6.2  6.3  6.2  6.1  6.0  5.9
```

Table 3-8. *mtypMesh.dat* file for two-dimensional mesh Example 2.

```
!iFlag key:
!     0=zone indices from x and y mesh input
!     1=element indices
!     2=(x,y) trapezoidal zone with vertical sides
!     3=polygon
!
!iFlag      im|xm ip|xp jm|ymm      jp|ypm      mZone
!                    ymp     ypp
0     1     99    1     99    1     !selection of entire grid by zone indices
0     2     4     3     3     2     !second material
```

The position of the left boundary (I=0) is explicitly specified at three zone indices J=1, 2 and 3 (Table 3-6):

x(0,1) = 1.0
x(0,2) = 0.5
x(0,3) = 1.5

The position of J=0 is implied to be the same as J=1; similarly, the position of J=4 is implicitly the same as J=3:

x(0,0) = 1.0
x(0,4) = 1.5

The next zonal grid line (I=1) is fixed at x=2. The next grid line (I=2) is specified by adding a thickness of 1.5 at nodal elevations J=0 (implied) and 1 (explicit) and 0.5 for J=2 and above (implied). The next grid (I=3) is fixed at x=4. The grid line I=4 is constructed by moving to specified positions x=4.5 and x=5 at nodal elevations J=0,1 and J=2,3,4 respectively. The next grid line (I=5) is fixed at x=6. The x-coordinates of the final grid line (I=6) are explicitly defined at every J node. All six I zones (Figure 3-2(b)) are subdivided into two grid cells (Figure 3-2(a)).

The position of the bottom boundary (zone index J=0) is specified through interpolation (Mode 3) using the (x, y) table values (Table 3-7)

(0, 0.5)
(2.5, 0)
(4, 1.0)
(4.5, 0.5)

Grid line J=2 presents another example of Mode 3. The remaining y-grid specifications use Modes 4 – 6 in a similar manner as the x-grid.

mtypMesh.dat (Table 3-8) defines just two materials as shown by the colored shading in Figure 3-2(b). Output file *Geometry.tec* contains two Tecplot "TextGeom" groups. Red lines delineate the grid zones which are subsequently subdivided to create the computational grid (Figure 3-2(a)). Black lines delineate distinct materials, which may or may not align with zone blocks.

4.0 References

Flach, G. P. and F. G. Smith III. Mesh2d Grid Generator Design and Use. SRNL-STI-2012-00005, Rev. 0. January 2012.

Appendix A. Fortran90 source code

```
program Mesh2d
!Mesh2d, Version 2.0, 02Oct2017
!  -- Modified by Frank Smith to support non-vertical walls in Saltstone Disposal Unit
6
!  -- Further modified by Greg Flach to be backward compatible with Version 1.01,
!     specifically, all y-grid lines fall on any prescribed dy|y=f(x) functions
!Mesh2d, Version 1.01, 15Sep2015:
!  --added code name, version, and date information to log file
!Mesh2d, Version 1.0, 2012:
!  --initial release

!Copyright © 2017, 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. The United States Government retains, for itself and others acting on
!its behalf, a paid-up, nonexclusive, irrevocable worldwide license to
!reproduce, to distribute copies to the public, to prepare derivative works,
!to perform publicly and to display publicly, and to permit others to do so for
!such 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 (only for y coordinates)
!   3   variable elevation (only for y coordinates)
!   4   variable thickness specifying zone vertices
!   5   variable elevation specifying zone vertices
!   6   all zone vertices specified

! iFlag for material type:
!   0   zone indices from x and y mesh input
!   1   element indices
!   2   (x,y) trapezoidal zone
!   3   selection by polygon

! direction key for mesh skewing:
!   p   positive
!   n   negative
!   c   centered
!   d   disabled
```

```
use global

implicit none

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

integer, parameter :: isup=10, &
                      iinp=11, &
                      jinp=12, &
                      minp=13, &
                      ilog=21, &
                      imsh=22, &
                      icor=23, &
                      ityp=24, &
                      itec=31, &
                      ite2=32, &
                      ite3=33, &
                      ipar=34, &
                      ignu=35, &
                      iply=41, &
                     icoor=51, &
                     ielem=52

integer :: iunt, inp, nargs, ipos
integer :: npx, npy, nnx, nny
integer :: nex, ney, nex2, ney2
integer :: i,j

integer, dimension(:)  , allocatable :: iz , jz
integer, dimension(:)  , allocatable :: izb, jzb, izb3
integer, dimension(:,:), allocatable :: mtyp

real(LONG), dimension(:,:), allocatable ::  x,  x2, x3
real(LONG), dimension(:,:), allocatable ::  y,  y2, y3
real(LONG), dimension(:,:), allocatable :: tx, tx2, tx3
real(LONG), dimension(:,:), allocatable :: ty, ty2, ty3

character(len=FilenameLength) :: Filename
character(len=PathLength) :: Path
character(len=FullLength) :: Fullpath
character(len=3) :: xyz
character(len=80) :: version

logical :: idebug

data idebug/.true./
data version/"Mesh2d, Version 2.0, 02Oct2017"/


!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,'(a)') "!"//trim(version)
write(ilog,'("!number of command lines arguments: ",i2)') nargs

!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

!READ INITIAL X MESH DATA
nex = 0; nex2 = 0
call read_init (iinp, "iinp", nex, nex2)
nnx = nex + 1
npx = nex + 2

!READ INITIAL Y MESH DATA
ney = 0; ney2 = 0
call read_init (jinp, "jinp", ney, ney2)
nny = ney + 1
npy = ney + 2

!ALLOCATE ARRAYS AND SET INITIAL VALUES
allocate ( x(0:nex,0:ney)); allocate ( x2(0:nex2,0:ney2))
allocate (tx(0:ney,0:nex)); allocate (tx2(0:ney2,0:nex2))

allocate ( y(0:nex,0:ney)); allocate ( y2(0:nex2,0:ney2))
allocate (ty(0:ney,0:nex)); allocate (ty2(0:ney2,0:nex2))

allocate (iz(0:nex))
allocate (jz(0:ney))
allocate (mtyp(0:nex,0:ney))

allocate (izb(0:nex2))
allocate (jzb(0:ney2))

 x = 0;  x2 = 0
```

```
 y = 0;  y2 = 0
tx = 0; tx2 = 0
ty = 0; ty2 = 0

iz = -999; jz = -999
mtyp = -999

izb = 0; jzb = 0

!WRITE LOG FILE
write(ilog,'(a,3i4)') "!nex, nnx, npx: ", nex, nnx, npx
write(ilog,'(a,3i4)') "!nex2: ", nex2
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 X ZONES
call make_zone (iinp, "iinp", x2, y2, nex2, ney2, izb)
write(ilog,'(/a)') "X Zones Created"

!CREATE Y ZONES
tx2 = transpose(x2)
call make_zone (jinp, "jinp", ty2, tx2, ney2, nex2, jzb)
y2 = transpose(ty2)
write(ilog,'(/a)') "Y Zones Created"

if (idebug) call debug_out (ilog, "Zones", x2, y2, nex2, ney2)

!CREATE X MESH ON ZONE BOUNDARIES
call mesh_zone (iinp, "iinp", x, y, x2, y2, jzb, iz, &
                nex, ney, nex2, ney2)
write(ilog,'(/a)') "X Zone Mesh Created"

!CREATE Y MESH ON ZONE BOUNDARIES
tx = transpose(y); tx2 = transpose(y2)
ty = transpose(x); ty2 = transpose(x2)

call mesh_zone (jinp, "jinp", tx, ty, tx2, ty2, izb, jz, &
                ney, nex, ney2, nex2)
write(ilog,'(/a)') "Y Zone Mesh Created"

x = transpose(ty)!; x2 = transpose(ty2)
y = transpose(tx)!; y2 = transpose(tx2)

if (idebug) call debug_out (ilog, "Zone Mesh", x, y, nex, ney)

if (.true.) then !added by Greg Flach for backward compatibility
  !FIRST SHARPEN Y ELEVATIONS
  ! The "sharpen" step ensures that all y-grid lines, not just y-zone lines,
  ! fall on any prescribed dy|y=f(x) functions

  allocate (x3(0:nex,0:ney2)); allocate (tx3(0:ney2,0:nex))
  allocate (y3(0:nex,0:ney2)); allocate (ty3(0:ney2,0:nex))

  do i=0,nex
    do j=0,ney2
      x3(i,j) = x(i,jzb(j))
      y3(i,j) = -999
    end do
  end do
```

```
  call sharpen (jinp, "jinp", x2, y2, x3, y3, nex, nex2, ney2)
  write(ilog,'(/a)') "Y Zones Sharpened"

  if (idebug) call debug_out (ilog, "Zones", x3, y3, nex, ney2)

  do i=0,nex
    do j=0,ney2
      y(i,jzb(j)) = y3(i,j)
    end do
  end do

  !THEN RE-CREATE Y MESH ON ZONE BOUNDARIES ASSUMING X-GRID LINES ARE X-ZONE LINES
  allocate (izb3(0:nex))
  do i=0,nex
    izb3(i) = i
  end do

  do i=0,nex
    do j=0,ney2
      x3(i,j) = x(i,jzb(j))
      y3(i,j) = y(i,jzb(j))
    end do
  end do

  tx = transpose(y); tx3 = transpose(y3)
  ty = transpose(x); ty3 = transpose(x3)

  call mesh_zone (jinp, "jinp", tx, ty, tx3, ty3, izb3, jz, &
                  ney, nex, ney2, nex)
  write(ilog,'(/a)') "Y Zone Mesh Re-created"

  x = transpose(ty)!; x2 = transpose(ty2)
  y = transpose(tx)!; y2 = transpose(tx2)

  if (idebug) call debug_out (ilog, "Zone Mesh", x, y, nex, ney)

  !GRID IS COMPLETE BECAUSE NO INTERNAL NODES LEFT (no need for make_grid)

else !as last coded by Frank Smith
  !CREATE GRID
  !Grid points internal to zone blocks are the intersection of straight lines
  !connecting zone boundary points. Not backward compatible with Mesh2d 1.01.
  !
  call make_grid (x, y, izb, jzb, &
                  nex, ney, nex2, ney2)
  write(ilog,'(/a)') "X-Y GRID Created"

  if (idebug) call debug_out (ilog, "Array", x, y, nex, ney)
end if

!CREATE MATERIAL TYPES
call set_materials (minp, "minp", mtyp, x, y, nex, ney, &
                    iz, jz, ite3, iply)
write(ilog,'(/a)') "Material types assigned"

!WRITE OUTPUT FILES
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, nny, &
                   x2, y2, nex2, ney2, izb, jzb)
```

```
call WriteParaview (ipar, x, y, mtyp, nex, ney)

call WriteGnuplot (ignu, x, y, mtyp, nex, ney)

write(ilog,'(/a)') "Output files written"

end program Mesh2d

!////////////////////////////////////////////////////////////////////
subroutine debug_out (ilog, label, x, y, nex, ney)

  use global

  implicit none

  integer :: ilog, nex, ney, i, j
  real(LONG) :: x(0:nex,0:ney), y(0:nex,0:ney)
  character(len=*) :: label

  write(ilog,'(/a,a)') "X ", label
  do j=0,ney
    write(ilog,*) (x(i,j), i=0,nex)
  end do
  write(ilog,'(/a,a)') "Y ", label
  do i=0,nex
    write(ilog,*) (y(i,j), j=0,ney)
  end do

end subroutine debug_out

!////////////////////////////////////////////////////////////////////
subroutine read_line (iunt, name, Line, LineLength, exit_flag)

  use global
  use stuff

  implicit none

  character(len=LineLength) :: Line
  character(len=4) :: name

  integer :: iostat_flag, iunt, LineLength
  logical :: exit_flag

  exit_flag = .false.

  do
    read (iunt,'(a)', iostat=iostat_flag) Line
    call IostatCheck (iostat_flag, name, exit_flag)
    if (exit_flag) exit  !end-of-file
    if (Line(1:1) .eq. '!' .or. len(trim(Line)) .eq. 0) cycle
    exit
  end do

end subroutine read_line

!////////////////////////////////////////////////////////////////////
subroutine read_init (iunt, name, nex, nex2)

  use global

  implicit none
```

```
  integer, parameter :: LineLength=80

  character(len=LineLength) :: Line
  character(len=4) :: name, scheme
  character(len=1) :: dir

  integer :: iFlag, iunt
  integer :: nex, nex2, nx, k

  real(LONG) :: skew, scale, tmp1

  logical exit_flag

  call read_line (iunt, name, Line, LineLength, exit_flag)
  read (Line,*) iFlag, scale, tmp1
  if (iFlag .gt. 1 .and. iFlag .lt. 6) then
    do k=1,int(tmp1)
      call read_line (iunt, name, Line, LineLength, exit_flag)
    end do
  end if

  do
    call read_line (iunt, name, Line, LineLength, exit_flag)
    if (exit_flag) exit  !end-of-file
    read (Line,*) iFlag, nx, dir, scheme, skew, scale, tmp1
    nex = nex + nx; nex2 = nex2 + 1

    if (iFlag .gt. 1 .and. iFlag .lt. 6) then
      do k=1,int(tmp1)
        call read_line (iunt, name, Line, LineLength, exit_flag)
      end do
    end if
  end do

  rewind (iunt)

end subroutine read_init

!////////////////////////////////////////////////////////////////
subroutine make_zone (iunt, name, x2, y2, nex2, ney2, izb)

  use global
  use Bspline

  implicit none

  integer, parameter :: LineLength=80

  character(len=LineLength) :: Line
  character(len=4) :: name, scheme
  character(len=1) :: dir

  integer :: iunt, nPts, i, k, iPt
  integer :: iFlag, nx, nex2, ney2, i2, j2
  integer :: izb(0:nex2)

  logical exit_flag

  real(LONG) :: skew, scale, tmp1, fx
  real(LONG) :: x2(0:nex2,0:ney2)
  real(LONG) :: y2(0:nex2,0:ney2)
```

```
real(LONG), dimension(:), allocatable :: tmpx, xPts, yPts
allocate(tmpx(0:ney2))

call read_line (iunt, name, Line, LineLength, exit_flag)
read (Line,*) iFlag, scale, tmp1

!The first line of input defines the location of the mesh border
if (iFlag .eq. 1) then
  tmpx = tmp1
else if (iFlag .eq. 3 .and. name .eq. "jinp") then
  nPts = int(tmp1)
  allocate(xPts(nPts))
  allocate(yPts(nPts))
  do k=1,nPts
    call read_line (iunt, name, Line, LineLength, exit_flag)
    read (Line,*) xPts(k), yPts(k)
    xPts(k) = scale*xPts(k)
    yPts(k) = scale*yPts(k)
  end do
  do j2=0,ney2
    call Interp(nPts,xPts,yPts,y2(0,j2),tmpx(j2))
  end do
  deallocate(xPts)
  deallocate(yPts)
  scale = 1
else if (iFlag .eq. 5) then
  nPts = int(tmp1)
  do k=1,nPts
    call read_line (iunt, name, Line, LineLength, exit_flag)
    read (Line,*) iPt, tmp1
    if (k .eq. 1) then
      tmpx(0:iPt) = tmp1
    else if (k .lt. nPts) then
      tmpx(iPt) = tmp1
    else
      tmpx(iPt:ney2) = tmp1
    end if
  end do
else if (iFlag .eq. 6) then
  read (Line,*) iFlag, scale, (tmpx(j2), j2=0,ney2)
else
  stop "invalid iFlag value"
end if

x2(0,0:ney2) = scale*tmpx(0:ney2)

i = 0; i2 = 0
do
  call read_line (iunt, name, Line, LineLength, exit_flag)
  if (exit_flag) exit  !end-of-file
  read (Line,*) iFlag, nx, dir, scheme, skew, scale, tmp1
  if (iFlag .eq. 0 .or. iFlag .eq. 1) then
      fx = 1 - iFlag
    tmpx = tmp1
  else if ((iFlag .eq. 2 .or. iFlag .eq. 3) .and. name .eq. "jinp") then
      fx = 3 - iFlag
    nPts = int(tmp1)
    allocate(xPts(nPts))
    allocate(yPts(nPts))
    do k=1,nPts
      call read_line (iunt, name, Line, LineLength, exit_flag)
      read (Line,*) xPts(k), yPts(k)
      xPts(k) = scale*xPts(k)
```

```
        yPts(k) = scale*yPts(k)
      end do
      do j2=0,ney2
        call Interp(nPts,xPts,yPts,y2(i2+1,j2),tmpx(j2))
      end do
      deallocate(xPts)
      deallocate(yPts)
      scale = 1
    else if (iFlag .eq. 4 .or. iFlag .eq. 5) then
        fx = 5 - iFlag
      nPts = int(tmp1)
      do k=1,nPts
        call read_line (iunt, name, Line, LineLength, exit_flag)
        read (Line,*) iPt, tmp1
        if (k .eq. 1) then
          tmpx(0:iPt) = tmp1
        else if (k .lt. nPts) then
          tmpx(iPt) = tmp1
        else
          tmpx(iPt:ney2) = tmp1
        end if
      end do
    else if (iFlag .eq. 6) then
      fx = 0
      read (Line,*) iFlag, nx, dir, scheme, skew, scale, (tmpx(j2), j2=0,ney2)
    else
    stop "invalid iFlag value"
    end if

    i = i + nx; i2 = i2 + 1
    izb(i2) = i
    x2(i2,0:ney2) = fx*x2(i2-1,0:ney2) + scale*tmpx(0:ney2)

  end do

  deallocate(tmpx)

  rewind (iunt)

end subroutine make_zone

!////////////////////////////////////////////////////////////////
subroutine sharpen (iunt, name, x2, y2, x3, y3, nex, nex2, ney2)

  use global
  use Bspline

  implicit none

  integer, parameter :: LineLength=80

  character(len=LineLength) :: Line
  character(len=4) :: name, scheme
  character(len=1) :: dir

  integer :: iunt, nPts, i, k, iPt
  integer :: iFlag, nx, nex, nex2, ney2, j2

  logical exit_flag

  real(LONG) :: skew, scale, tmp1
  real(LONG) :: x3(0:nex,0:ney2)
  real(LONG) :: y3(0:nex,0:ney2)
```

```
real(LONG) :: x2(0:nex2,0:ney2)
real(LONG) :: y2(0:nex2,0:ney2)

real(LONG), dimension(:), allocatable :: tmpy, xPts, yPts
allocate(tmpy(0:nex))

if (name .ne. "jinp") stop 'name .ne. "jinp"'

!J=0
j2 = 0
call read_line (iunt, name, Line, LineLength, exit_flag)
read (Line,*) iFlag, scale, tmp1

if (iFlag .eq. 3) then
  nPts = int(tmp1)
  allocate(xPts(nPts))
  allocate(yPts(nPts))
  do k=1,nPts
    call read_line (iunt, name, Line, LineLength, exit_flag)
    read (Line,*) xPts(k), yPts(k)
    xPts(k) = scale*xPts(k)
    yPts(k) = scale*yPts(k)
  end do

else if (iFlag .eq. 1  .or.  iFlag .eq. 5  .or.  iFlag .eq. 6) then
  nPts = nex2 + 1
  allocate(xPts(nPts))
  allocate(yPts(nPts))
  do i=0,nex2
    xPts(i+1) = x2(i,j2)
    yPts(i+1) = y2(i,j2)
  end do

  if (iFlag .eq. 5) then
    do k=1,nPts
      call read_line (iunt, name, Line, LineLength, exit_flag)
    end do
  end if

else
  stop "invalid iFlag value"
end if

do i=0,nex
  call Interp(nPts,xPts,yPts,x3(i,j2),tmpy(i))
end do

deallocate(xPts)
deallocate(yPts)

y3(0:nex,j2) = tmpy(0:nex)


!J>0
j2 = 1
do
  call read_line (iunt, name, Line, LineLength, exit_flag)
  if (exit_flag) exit  !end-of-file
  read (Line,*) iFlag, nx, dir, scheme, skew, scale, tmp1

  if (iFlag .eq. 2  .or.  iFlag .eq. 3) then
    nPts = int(tmp1)
    allocate(xPts(nPts))
```

```
      allocate(yPts(nPts))
      do k=1,nPts
        call read_line (iunt, name, Line, LineLength, exit_flag)
        read (Line,*) xPts(k), yPts(k)
        xPts(k) = scale*xPts(k)
        yPts(k) = scale*yPts(k)
      end do

    else if (iFlag .eq. 1  .or.  iFlag .eq. 5  .or.  iFlag .eq. 6) then
      if (iFlag .eq. 5) then
        nPts = int(tmp1)
        do k=1,nPts
          call read_line (iunt, name, Line, LineLength, exit_flag)
        end do
      end if

      nPts = nex2 + 1
      allocate(xPts(nPts))
      allocate(yPts(nPts))
      do i=0,nex2
        xPts(i+1) = x2(i,j2)
        yPts(i+1) = y2(i,j2)
      end do

    else if (iFlag .eq. 0) then
      nPts = 1
      allocate(xPts(nPts))
      allocate(yPts(nPts))
      xPts(1) = 0
      yPts(1) = scale*tmp1

    else if (iFlag .eq. 4) then
      nPts = int(tmp1)
      allocate(xPts(nPts))
      allocate(yPts(nPts))
      do k=1,nPts
        call read_line (iunt, name, Line, LineLength, exit_flag)
        read (Line,*) iPt, tmp1
        xPts(k) = x2(iPt,j2)
        yPts(k) = scale*tmp1
      end do

    else
      stop "invalid iFlag value"
    end if

    do i=0,nex
      call Interp(nPts,xPts,yPts,x3(i,j2),tmpy(i))
    end do

    deallocate(xPts)
    deallocate(yPts)

    if (iFlag .eq. 0  .or. iFlag .eq. 2  .or.  iFlag .eq. 4) then
      y3(0:nex,j2) = y3(0:nex,j2-1) + tmpy(0:nex)
    else
      y3(0:nex,j2) = tmpy(0:nex)
    end if

    j2 = j2 + 1

  end do
```

```
  deallocate(tmpy)

  rewind (iunt)

end subroutine sharpen

!////////////////////////////////////////////////////////////////
subroutine mesh_zone (iunt, name, x, y, x2, y2, jzb, iz, &
                      nex, ney, nex2, ney2)
  use global

  implicit none

  integer, parameter :: LineLength=80

  character(len=LineLength) :: Line
  character(len=4) :: name, scheme
  character(len=1) :: dir

  integer :: iFlag, iunt, nx
  integer :: nex, ney, nex2, ney2, iZone
  integer :: j, jj, i, ii, k
  integer :: iz(0:nex)
  integer :: jzb(0:ney2)

  logical exit_flag

  real(LONG) :: skew, scale, tmp1, dx, dy
  real(LONG) :: x(0:nex,0:ney), x2(0:nex2,0:ney2)
  real(LONG) :: y(0:nex,0:ney), y2(0:nex2,0:ney2)

  call read_line (iunt, name, Line, LineLength, exit_flag)
  read (Line,*) iFlag, scale, tmp1
  if (iFlag .gt. 1 .and. iFlag .lt. 6) then
    do k=1,int(tmp1)
      call read_line (iunt, name, Line, LineLength, exit_flag)
    end do
  end if

  i=0; iZone=1
  do
    call read_line (iunt, name, Line, LineLength, exit_flag)
    if (exit_flag) exit  !end-of-file
    read (Line,*) iFlag, nx, dir, scheme, skew, scale, tmp1
    if (iFlag .gt. 1 .and. iFlag .lt. 6) then
      do k=1,int(tmp1)
        call read_line (iunt, name, Line, LineLength, exit_flag)
      end do
    end if

    do jj=0,ney2
      j = jzb(jj)
      x(i,j) = x2(iZone-1,jj)
      y(i,j) = y2(iZone-1,jj)
      dx = x2(iZone,jj) - x(i,j)
      dy = y2(iZone,jj) - y(i,j)

      if (scheme .eq. "prev") then
        call prev (x(0:nex,j), nx, nex, iz, iZone, i, dx, dir, skew)

      else if (scheme .eq. "poly") then
        call poly (x(0:nex,j), nx, nex, iz, iZone, i, dx, dir, skew)
```

```
      else if (scheme .eq. "geom") then
        call geom (x(0:nex,j), nx, nex, iz, iZone, i, dx, dir, skew)

      else
        stop "invalid scheme"
      end if

      !LOCATE MESH POINTS ALONG ZONE BORDER
      do ii=i,i+nx
        y(ii,j) = y(i,j) + dy*(x(ii,j) - x(i,j))/dx
      end do

    end do

    i = i + nx
    iZone = iZone + 1
  end do

  rewind (iunt)

end subroutine mesh_zone

!////////////////////////////////////////////////////////////////
subroutine make_grid (x, y, izb, jzb, &
                      nex, ney, nex2, ney2)
  use global

  implicit none

  integer :: nex, ney, nex2, ney2
  integer :: izb(0:nex2), jzb(0:ney2)
  integer :: i, j, ii, jj
  integer :: i1, i2, j1, j2

  real(LONG) :: x(0:nex,0:ney)
  real(LONG) :: y(0:nex,0:ney)
  real(LONG) :: xl, yl, xr, yr
  real(LONG) :: xb, yb, xt, yt
  real(LONG) :: xint, yint

  i1 = 0; j1 = 0
  i2 = 0; j2 = 0
  !LOOP OVER X ZONES
  do i=1,nex2
    i2 = i1 + izb(i) - izb(i-1)
    if (i2 .le. i1) cycle
    !LOOP OVER Y ZONES
    do j=1,ney2
      j2 = j1 + jzb(j) - jzb(j-1)
      if (j2 .le. j1) cycle

      !LOOP OVER Y BORDER POINTS IN ZONE
      do jj=j1+1,j2-1
        xl = x(i1,jj); xr = x(i2,jj)
        yl = y(i1,jj); yr = y(i2,jj)
        !LOOP OVER X BORDER POINTS IN ZONE
        do ii=i1+1,i2-1
          xb = x(ii,j1); xt = x(ii,j2)
          yb = y(ii,j1); yt = y(ii,j2)
          call intersect (xl, yl, xr, yr, xb, yb, xt, yt, xint, yint)
          x(ii,jj) = xint
          y(ii,jj) = yint
        end do
```

```
      end do

      j1 = j2
    end do
    j1 = 0
    i1 = i2
  end do

end subroutine make_grid

!//////////////////////////////////////////////////////////////////
subroutine intersect (xl, yl, xr, yr, xb, yb, xt, yt, xint, yint)

    use global

    implicit none

    real(LONG) :: xl, yl, xr, yr
    real(LONG) :: xb, yb, xt, yt
    real(LONG) :: xint, yint
    real(LONG) :: slope1, slope2

    slope1 = 0; slope2 = 0

    if (yl .eq. yr) then
      yint = yl
      if (xt .eq. xb) then
        xint = xt
      else
        slope1 = (yt - yb)/(xt - xb)
        xint = xt + (yint - yt)/slope1
      end if
    else
      slope2 = (yr - yl)/(xr - xl)
      if (xt .eq. xb) then
        xint = xt
      else
        slope1 = (yt - yb)/(xt - xb)
        xint = (yr - slope2*xr - yt + slope1*xt)/(slope1 - slope2)
      end if
      yint = yr + slope2*(xint - xr)
    end if

end subroutine intersect

!//////////////////////////////////////////////////////////////////
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 (skew .lt. zero) stop "specify skew >=0"
        a = zero
        b =  3 - 2 * skew
        c = -6 * (1 - skew)
        d =  4 * (1 - skew)
        factor = a + (b + (c + d * xi) * xi) * xi
      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 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 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 set_materials (iunt, name, mtyp, x, y, nex, ney, iz, jz, &
                          ite3, iply)

  use global
  use stuff
```

```
use Polyg

implicit none

integer, parameter :: LineLength=80
integer, parameter :: FilenameLength=80

character(len=LineLength) :: Line
character(len=FilenameLength) :: Filename
character(len=4) name
character(len=6) :: color

integer :: iunt, ite3, iply
integer :: iFlag, nex, ney, mZone
integer :: i, j, im, ip, jm, jp
integer :: iIndex, jIndex

integer :: mtyp(0:nex,0:ney)
integer :: iz(0:nex)
integer :: jz(0:ney)

real(LONG) :: x(0:nex,0:ney), y(0:nex,0:ney)
real(LONG) :: tmp1, tmp2, tmp3, tmp4, xm, xp, ym, yp, ymm, ypm, ymp, ypp
real(LONG) :: xoff, yoff, sint, cost, srse, srsn, xCenter, yCenter

logical :: first, inplg, is3d, exit_flag

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

do
  call read_line (iunt, name, Line, LineLength, exit_flag)
  if (exit_flag) exit  !end-of-file
  read (Line,*) iFlag, tmp1, tmp2, tmp3, tmp4, mZone

  if (iFlag .le. 1) then
    im = int(tmp1)
    ip = int(tmp2)
    jm = int(tmp3)
    jp = int(tmp4)

  else if (iFlag .eq. 2) then
    xm  = tmp1
    xp  = tmp2
    ymm = tmp3
    ypm = tmp4
    call read_line (iunt, name, Line, LineLength, exit_flag)
    if (exit_flag) exit  !end-of-file
    read (Line,*) ymp, ypp

    !WRITE POLYGON PLOTTING FILE FOR TECPLOT (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
    call read_line (iunt, name, Line, LineLength, exit_flag)
    if (exit_flag) exit  !end-of-file
```

```
      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 = 0.25*(x(i,j  ) + x(i+1,j  ) + &
                          x(i,j+1) + x(i+1,j+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 - ymp)/(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 = 0.25*(x(i,j  ) + x(i+1,j  ) + &
                          x(i,j+1) + x(i+1,j+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; sint=0; cost=1
          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
  end do

end subroutine set_materials

!////////////////////////////////////////////////////////////////
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,0:ney), 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,j), 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), 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,0:ney)

    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,g14.6)') (i+1, tab, x(i,0), 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,izb,jzb)

    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)
```

```
integer :: izb(0:nex2)
integer :: jzb(0:ney2)

real(LONG)  x(0:nex ,0:ney ),  y(0:nex ,0:ney )
real(LONG) x2(0:nex2,0:ney2), y2(0:nex2,0:ney2)

logical writeline

!WRITE TECPLOT 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,j), y(i,j), mtyp(i,j), i=0,nex), j=0,ney)

!WRITE TEXTGEOM FILE FOR ZONAL BOUNDARIES
if (.false.) then
  !Assumes straight line connection between zone nodes
  do i2=0,nex2
    do j2=1,ney2
      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))') x2(i2,j2-1), y2(i2,j2-1)
      write(ite2,'(2(g14.6))') x2(i2,j2  ), 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=RED'
      write(ite2,'(i1)') 1
      write(ite2,'(i3)') 2
      write(ite2,'(2(g14.6),i7)') x2(i2-1,j2), y2(i2-1,j2)
      write(ite2,'(2(g14.6),i7)') x2(i2  ,j2), y2(i2  ,j2)
    end do
  end do

else
  !Allows piecewise linear connection between zone nodes
  do i2=0,nex2
    do j2=1,ney2
      write(ite2,'(a)') 'GEOMETRY X=0, Y=0, CS=GRID, T=LINE, C=RED'
      write(ite2,'(i1)') 1
      write(ite2,'(i3)') jzb(j2)-jzb(j2-1)+1
      do j=jzb(j2-1), jzb(j2)
        write(ite2,'(2(g14.6))') x(izb(i2),j), y(izb(i2),j)
      end do
    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=RED'
      write(ite2,'(i1)') 1
      write(ite2,'(i3)') izb(i2)-izb(i2-1)+1
      do i=izb(i2-1), izb(i2)
        write(ite2,'(2(g14.6),i7)') x(i,jzb(j2)), y(i,jzb(j2))
      end do
    end do
  end do
```

```
    end if

    !WRITE TEXTGEOM FILE FOR MATERIAL BOUNDARIES
    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=BLACK'
                write(ite2,'(i1)') 1
                write(ite2,'(i3)') 2
                write(ite2,'(2(g14.6),i7)') x(i,j-1), y(i,j-1)
                write(ite2,'(2(g14.6),i7)') x(i,j  ), 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=BLACK'
                write(ite2,'(i1)') 1
                write(ite2,'(i3)') 2
                write(ite2,'(2(g14.6),i7)') x(i-1,j), y(i-1,j)
                write(ite2,'(2(g14.6),i7)') x(i  ,j), 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,0:ney), 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)') &
        '    <Piece Extent="0 ', &
        nex, &
        ' 0 ', &
        ney, &
        ' 0 ', &
        1, &
        '">'

    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,j), y(i,j), zero, i=0,nex), j=0,ney)
    write(ipar,'(3(g14.6))') ((x(i,j), y(i,j),  one, i=0,nex), j=0,ney)

    write(ipar,'(a)') '        </DataArray>'
    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)') '        </DataArray>'
    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)') '        </DataArray>'
    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,0:ney), 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,j), y(i,j), mtyp(i,j), i=0,nex)
        write(ignu,'(a)') ""
    end do

end subroutine WriteGnuplot


module Bspline

!Copyright © 2017, 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. The United States Government retains, for itself and others acting on
!its behalf, a paid-up, nonexclusive, irrevocable worldwide license to
!reproduce, to distribute copies to the public, to prepare derivative works,
!to perform publicly and to display publicly, and to permit others to do so for
!such 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.

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

      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
  end do

  b0 = bs(0)

  return
  end function b0

end module Bspline


module polyg

!Copyright © 2017, 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. The United States Government retains, for itself and others acting on
!its behalf, a paid-up, nonexclusive, irrevocable worldwide license to
!reproduce, to distribute copies to the public, to prepare derivative works,
!to perform publicly and to display publicly, and to permit others to do so for
!such 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, &
                  nv,xv,yv, 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,ymax, xmodel,ymodel, xoff,yoff,sint,cost, x,y
real(LONG), dimension(nvmax) :: xv,yv

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
  xv(nv) = x
  yv(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


module stuff

!Copyright © 2017, 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. The United States Government retains, for itself and others acting on
!its behalf, a paid-up, nonexclusive, irrevocable worldwide license to
!reproduce, to distribute copies to the public, to prepare derivative works,
!to perform publicly and to display publicly, and to permit others to do so for
!such 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.

implicit none

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


module global

!Copyright © 2017, 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. The United States Government retains, for itself and others acting on
!its behalf, a paid-up, nonexclusive, irrevocable worldwide license to
!reproduce, to distribute copies to the public, to prepare derivative works,
!to perform publicly and to display publicly, and to permit others to do so for
!such 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
```

This page intentionally left blank