MFIX

Multiphase Flow with Interphase eXchange

Cartesian Grid
User Guide

Jeff Dietiker

September 21st, 2009

Table of Contents

1. Introduction

A new capability, called Cartesian grid cut-cell technique has been implemented in MFIX, which allows the definition of curved or sloping boundaries, instead of the usual stair-step representation. Computational cells are truncated at the wall to conform to the shape of the boundaries. When a face is truncated, the velocity node is moved to the center of the face. The cell truncation introduces an additional face, called the cut face. Face surface areas and cell volumes are updated based on the shape of the cut cell. The contribution of the new cut face is added to the computation. The data can be saved in a vtk file for post-processing purpose.

The implementation of the Cartesian grid cut-cell technique is based on the work of Kirkpatrick and Armfield, and details about the cut cell treatment can be found in Reference [1]. Modifications have been implemented for the Eulerian/Eulerian approach, i.e., it is not available for Discrete Element Model. The cut-cell technique is a complement to existing boundary conditions. The usual specification of boundary conditions is still available.

This document describes how to use the Cartesian grid cut-cell capability. The new keywords introduced in mfix.dat are described and some examples illustrate their utilization. A series of tutorial files are provided with the MFIX distribution to help users get familiar with the new technique.

It is assumed that the reader is familiar with the general operation of MFIX. This document only describes the utilization of the Cartesian cut-cell technique. Details about MFIX can be found in Refs. [2-4].

2. Geometry definition

The Cartesian grid cut cell capability is activated by setting the keyword `CARTESIAN_GRID =` `.TRUE.`. Once this option is activated, the boundary geometry must be specified using one of the following methods:

a) **Quadric surface(s)**: The boundaries are defined using one or several quadric surfaces that can be translated, rotated, and combined. This option is activated by specifying a positive number of quadric surfaces (`N_QUADRIC` ≥ 1), and is valid for two and three-dimensional geometries.

b) **Polygons**: The boundaries are defined using one or several polygons. The geometry information is read from the data file poly.dat, which must be generated prior to running MFIX, and must be located in the run directory. This option is activated by setting the keyword `USE_POLYGON = .TRUE.`, and is limited to two-dimensional geometry only.

c) **User-defined function**: The boundaries are specified using a user-defined function. The geometry is defined in the subroutine `eval_usr_fct.f` prior to running MFIX. The code must be compiled every time this subroutine is modified. This option is activated by setting `N_USR_DEF = 1`, and is valid for two and three-dimensional geometries.

Note: In the current version, only one method can be used at a time.

2.1. Quadric surface

2.1.1. Normal form

Quadric surface parameters are defined as one-dimensional arrays, which index corresponds to the quadric being defined. In this document, the index is referred to as the quadric ID (`QID`). Several quadrics can be defined. The total number of active quadrics is defined by the keyword `N_QUADRIC`.

Quadric surfaces are written into one of their normal form:

$$f(x, y, z) = \lambda_x x^2 + \lambda_y y^2 + \lambda_z z^2 + d = 0 \qquad (1)$$

Where $\lambda_x, \lambda_y, \lambda_z$ and d are real scalars. The boundary is located where the function $f(x, y, z)$ is zero (within some tolerance defined by the parameter `TOL_F`). Regions where $f(x, y, z)$ is positive are excluded from the computational domain (blocked cells), and regions where $f(x, y, z)$ is negative are part of the computational domain (fluid cells). This option is activated by the keyword `QUADRIC_FORM(QID) = 'NORMAL'`, and is the default value. By default, all quadrics are centered about the origin (x = 0, y = 0, z = 0), and any quadric can be translated and rotated (see section 2.1.3).

Examples:

The following two examples illustrate the description of simple quadric surfaces (cylinder and cone). The idea is to express the shape's surface in the form of Equation (1) such that the quadric's parameters can be identified. The same procedure can be applied to other quadric surfaces, such as ellipsoids, hyperboloids, elliptic cylinders, etc. Note that there is an alternate way of defining planes, cylinders and cones, using pre-defined input (see section 2.1.2).

Circular cylinder
To model the flow over a circular cylinder of radius 2 in the (xy) plane (Figure 1a), the equation defining the boundary must be rearranged to fit the form of Equation (1), i.e.,
$x^2 + y^2 = 4$ becomes $f(x, y, z) = -x^2 - y^2 + 4 = 0$, since we want values of $f(x, y, z)$ to be negative outside the cylinder. Therefore, the following parameters are specified as:

$$\lambda_x = -1.0 \ ; \ \lambda_y = -1.0 \ ; \ \lambda_z = 0.0 \ ; \ d = 4.0 \tag{2}$$

In mfix.dat, this will be written as:

```
CARTESIAN_GRID = .TRUE.
N_QUADRIC = 1
lambda_x(1) = -1.0
lambda_y(1) = -1.0
lambda_z(1) =  0.0
dquadric(1) =  4.0
```

Note that it would be equivalent to use:

$$\lambda_x = -0.25 \ ; \ \lambda_y = -0.25 \ ; \ \lambda_z = 0.0 \ ; \ d = 1.0 \tag{3}$$

i.e., multiplying all coefficients by the same positive constant does not have any effect. However, changing the sign of all coefficients reverses blocked and fluid cells. Using the following parameters

$$\lambda_x = 1.0 \ ; \ \lambda_y = 1.0 \ ; \ \lambda_z = 0.0 \ ; \ d = -4.0 \tag{4}$$

will mesh the interior of the cylinder (Figure 1b).
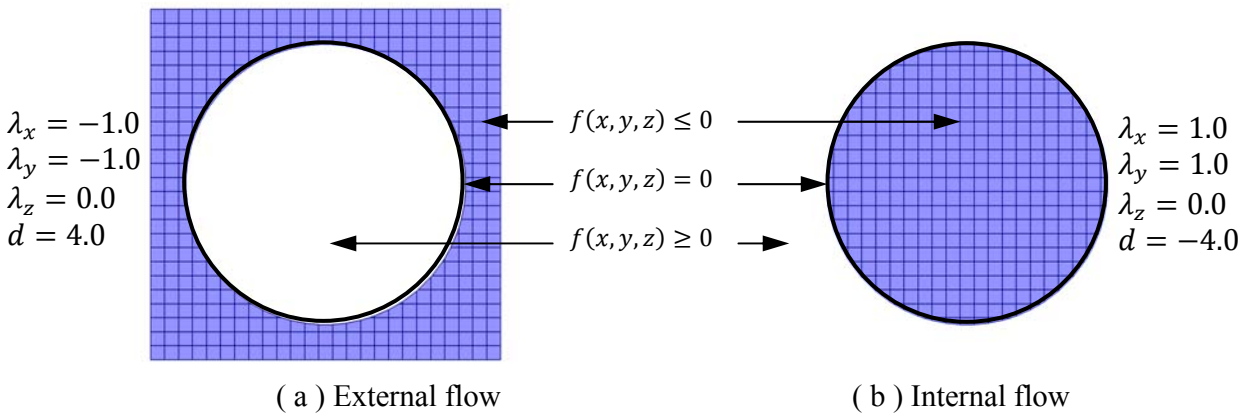
(a) External flow   (b) Internal flow
Figure 1. Meshing the exterior or interior region of a cylinder.

Cone

Assume the cone's axis of revolution is aligned with the y-axis, it has a cylindrical cross section along any y-plane, and the cone's half-angle is β. Since the radius varies linearly with height, we can write $r = \sqrt{x^2 + z^2} = y \tan\beta$, which can be re-arranged into

$$\frac{x^2}{(\tan\beta)^2} - y^2 + \frac{z^2}{(\tan\beta)^2} = 0 \tag{5}$$

i.e.,

$$\lambda_x = \frac{1}{(\tan\beta)^2} \;;\; \lambda_y = -1.0 \;;\; \lambda_z = \frac{1}{(\tan\beta)^2} \;;\; d = 0.0 \tag{6}$$

Notes:
1) The cone is by default centered around the origin. It will typically need to be translated, and probably combined with other quadrics (for example to model a spouted bed).
2) In 2D, we could leave $\lambda_z = 0$ since there is no z-dependence.

For example, to define a cone with half-angle of β =30 degrees, the following input can be entered in mfix.dat (note that $\tan(30\ deg) = 1/\sqrt{3}$):

```
CARTESIAN_GRID = .TRUE.

N_QUADRIC = 1

  lambda_x(1) =   3.0D0
  lambda_y(1) =  -1.0D0
  lambda_z(1) =   3.0D0
  dquadric(1) =   0.0D0

  t_x(1) = 2.5          ! Translation in x direction
  t_y(1) = 0.0          ! Translation in y direction
  t_z(1) = 2.5          ! Translation in z direction
```
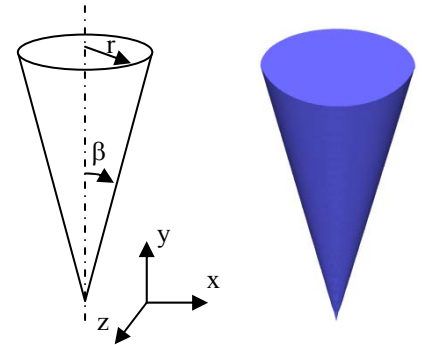

2.1.2. Pre-defined quadrics

It is anticipated that a few common quadrics, such as planes, cylinders, and cones will be used on a regular basis. Their definition is facilitated through user-friendly input that bypasses the specification of quadric parameters. The type of pre-defined quadric is set by the keyword QUADRIC_FORM. Current pre-defined quadrics include planes, cylinders and cones.

Plane

Single planes are defined by specifying the plane normal vector, and a point belonging to the plane. This option is activated by the keyword QUADRIC_FORM(QID) = 'PLANE'. The following notation is used:

$\vec{n} = (n_x, n_y, n_z)$ is the normal vector to the plane, pointing away from fluid cells. The normal vector does not need to be normalized. $\vec{P} = (x, y, z)$ is any point in space. $\vec{t} = (t_x, t_y, t_z)$ is a point belonging to the plane.

The equation of the plane is given by $\vec{n} \cdot (\vec{P} - \vec{t}) = 0$

or

$$f(x, y, z) = n_x x + n_y y + n_z z + d = 0 \tag{7a}$$

where

$$d = -(n_x t_x + n_y t_y + n_z t_z) \tag{7b}$$

The user needs to specify values of $\vec{n} = (n_x, n_y, n_z)$ and $\vec{t} = (t_x, t_y, t_z)$. The coefficient d can be left undefined as it will be automatically computed based on Equation (7b). From a practical point of view, the plane's orientation is defined by the vector $\vec{n}$, and translated by the amount $\vec{t}$.

Notes: 1) The vector $\vec{n}$ points away from the fluid cells.
        2) Planes are useful to clip other quadrics or groups of quadrics (see section 2.1.9).

<u>Example:</u>

The following parameters define a plane perpendicular to the (xy) plane, passing through (x=3, y=2), and rotated 20 degrees with respect to the y-axis (Figure 2):

```
CARTESIAN_GRID = .TRUE.
N_QUADRIC = 1
QUADRIC_FORM(1)  = 'PLANE'
n_x(1) =    0.940
n_y(1) =    0.342
t_x(1) =    3.0
t_y(1) =    2.0
```


$\vec{n} = (0.940, 0.342)$

$(3, 2)$

y

x
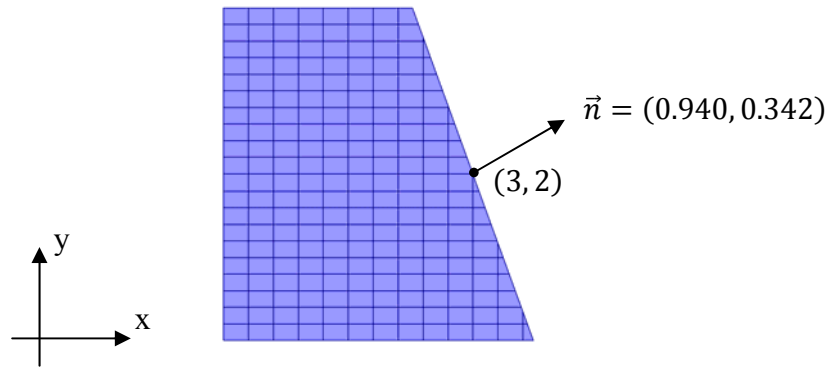
Figure 2. Example of plane definition.

Cylinder

A circular cylinder is defined by specifying its radius, initial orientation, and whether we want to model internal or external flow. The keyword QUADRIC_FORM(QID) contains information regarding the initial orientation and type of flow. For example, setting QUADRIC_FORM(QID) = 'Y_CYL_INT' will define a cylinder which axis of revolution is along the y-direction, for internal flow computation. Using QUADRIC_FORM(QID) = 'Z_CYL_EXT' will define a cylinder which axis of revolution is along the z-direction, for external flow computation. The cylinder radius is defined by setting a positive value to RADIUS(QID).

For example, to model the flow over a cylinder of radius 2, in the (xy) plane (the axis of revolution points in the z-direction), the following parameters are defined

```
CARTESIAN_GRID = .TRUE.
N_QUADRIC = 1
QUADRIC_FORM(1) = 'Z_CYL_EXT'
RADIUS(1) = 2.0
```

The initial orientation is limited to x, y, or z axis, but the cylinder can be rotated by any arbitrary angle, and translated in any direction (see section 2.1.3).

Cone

A cone is defined by setting its half-angle and initial orientation. A value of QUADRIC_FORM(QID) = 'Y_CONE' will define a cone which axis of revolution is aligned with the y-axis. The keyword HALF_ANGLE(QID) defined the cone's half-angle (in degrees).
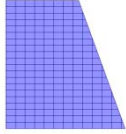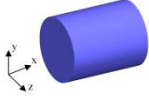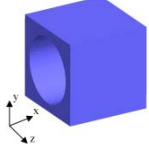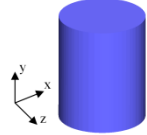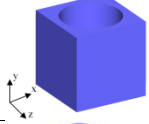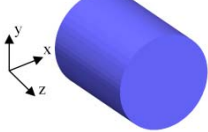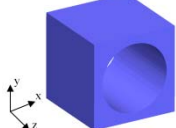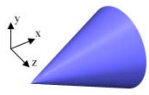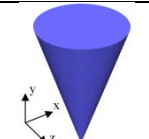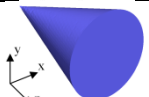
For example, to define a cone in the y-direction with half-angle of 30 degrees:

```
CARTESIAN_GRID = .TRUE.
N_QUADRIC = 1
QUADRIC_FORM(1) = 'Y_CONE'
HALF_ANGLE(1) = 30.0
```

By default, the cone's apex is located at the origin. The cone can be translated and rotated as any other quadric (see section 2.1.3). Currently, only internal flows can be modeled with the pre-defined cone.

Table 1 summarizes current available pre-defined quadrics. Other pre-defined quadrics can be included by adding them to the list in the SELECT CASE(TRIM(QUADRIC_FORM(Q))) construct in subroutine check_data_cartesian.f. The code must be recompiled for the modification to take effect.

Pre-defined quadric	Snapshot	Value of QUADRIC_FORM(QID)	Other input required
Plane		'PLANE'	n_x(QID) n_y(QID) n_z(QID)
Cylinder aligned with x-axis Internal flow		'X_CYL_INT'	RADIUS(QID)
Cylinder aligned with x-axis External flow		'X_CYL_EXT'	RADIUS(QID)
Cylinder aligned with y-axis Internal flow		'Y_CYL_INT'	RADIUS(QID)
Cylinder aligned with y-axis External flow		'Y_CYL_EXT'	RADIUS(QID)
Cylinder aligned with z-axis Internal flow		'Z_CYL_INT'	RADIUS(QID)
Cylinder aligned with z-axis External flow		'Z_CYL_EXT'	RADIUS(QID)
Cone aligned with x-axis Internal flow		'X_CONE'	HALF_ANGLE(QID)
Cone aligned with y-axis Internal flow		'Y_CONE'	HALF_ANGLE(QID)
Cone aligned with z-axis Internal flow		'Z_CONE'	HALF_ANGLE(QID)

Table 1. Available pre-defined quadrics.

2.1.3. Translating and rotating quadrics

Any quadric surface defined in its normal form can be translated by the amount $\vec{t} = (t_x, t_y, t_z)$, and rotated around the x, y, and z axes, by the amount $\theta_x, \theta_y, and \, \theta_z$ respectively (angles expressed in degrees). The order of rotation is currently fixed and is performed first around the x-axis, then the y-axis, and finally the z-axis.

Example:

To translate quadric 1 by the amount $\vec{t} = (1,2)$, use:

```
t_x(1)      =   1.0
t_y(1)      =   2.0
```

To rotate quadric 5 by θ_y =20 degrees about the y-axis, use:

```
Theta_y(5) = 20.0
```

Quadric surfaces defined in their degenerate forms (planes) are not allowed to be translated or rotated since all information required to define the plane is already contained in the normal vector $\vec{\lambda}$ and point $\vec{t}$ belonging to the plane.

2.1.4. Clipping limits

By default, the function $f(x, y, z)$ is defined over the entire original computational domain, i.e., $0 \leq x \leq XLENGTH, 0 \leq y \leq YLENGTH, and \, 0 \leq z \leq ZLENGTH$. It is possible to limit the definition of a given quadric surface to a rectangular region:
$clip\_xmin \leq x \leq clip\_xmax, \; clip\_ymin \leq y \leq clip\_ymax, and \, clip\_zmin \leq z \leq clip\_zmax$.
This has the effect of truncating the quadric. The region where the quadric is not defined (clipped region) can be considered either as part of the computational domain (fluid cells) or excluded from the computation (blocked cells). The flag FLUID_IN_CLIPPED_REGION is used to switch from one option to another. The default value is .TRUE.. Figure 3 illustrates the use of clipping limits for an internal flow. The quadric is limited to the region $5.0 \leq x \leq 20.0$. When FLUID_IN_CLIPPED_REGION = .TRUE., the cells that are outside the clipping limits are retained in the computational domain, whereas when FLUID_IN_CLIPPED_REGION = .FALSE., the cells that are outside the clipping limits are removed from the computational domain.
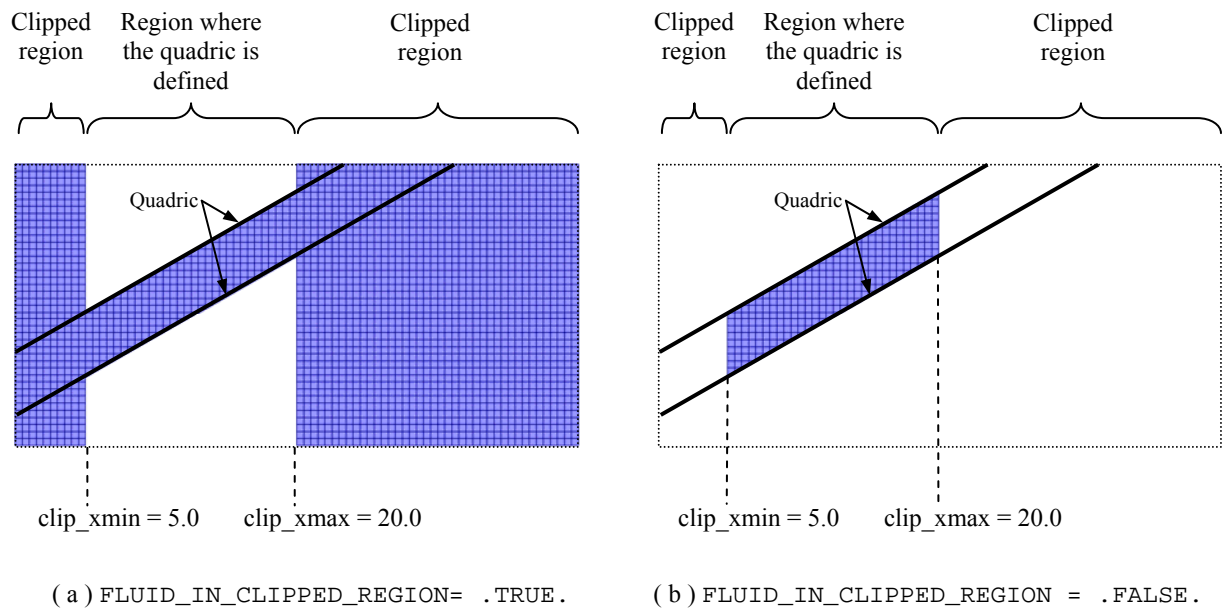
Figure 3. Illustration of clipping limits.

Notes:
1) The use of clipping limits is mostly used while defining piecewise functions (section 2.1.8).
2) Quadrics can be clipped along arbitrary planes using group relations (sections 2.1.5 to 2.1.9).

2.1.5. Combining quadrics

All quadric surfaces must belong to a group. The number of group(s) must be defined (default value is N_GROUP = 1), and the size of each group determines the number of quadric in the groups (default value is GROUP_SIZE(1) = 1). For each group, a list of quadric IDs is specified to populate the groups, using the matrix GROUP_Q(I,J), which store the quadric ID of Jth quadric assigned to group I.

Example:

Assuming N_QUADRIC = 5, the following would form two groups: group 1 combining quadrics 1, 2 and 4, and group 2 combining quadrics 3 and 5.

```
N_GROUP = 2
GROUP_SIZE(1) = 3
GROUP_Q(1,1)   = 1
GROUP_Q(1,2)   = 2
GROUP_Q(1,3)   = 4
GROUP_RELATION(1) = 'OR'

GROUP_SIZE(1) = 2
GROUP_Q(2,1)   = 3
GROUP_Q(2,2)   = 5
GROUP_RELATION(2) = 'AND'
```

`RELATION_WITH_PREVIOUS(2) = 'AND'`

Quadrics belonging to the same group can be combined by setting the keyword `GROUP_RELATION` for that group. Available options are `'OR'`, `'AND'` (logical grouping), and `'PIECEWISE'`.

2.1.6. Logical grouping within a group

Several quadrics belonging to a common group can be combined using the `'OR'` or `'AND'` attribute. In this case, the actual intersection points between quadrics do not need to be known in advance. For a given group `GID`, if `GROUP_RELATION(GID) = 'OR'`, a point belongs to the computational domain if at least one value of $f(x, y, z)$ among all quadrics is negative. If `GROUP_RELATION(GID) = 'AND'`, a point belongs to the computational domain if all values of $f(x, y, z)$ among all quadrics are negative. Each group is assigned an f-value, based on the group relation. With an `'OR'` relation, the f-value is the minimum value of $f(x, y, z)$ for all quadrics in the group. With an `'AND'` relation, the f-value is the maximum value of $f(x, y, z)$ for all quadrics in the group.

In the previous example, quadrics in group 1 are combined with the `'OR'` relation, and quadrics in group 2 are combined with the `'AND'` relation.

Note that this attribute will behave differently for internal and external flows (Figure 4).

Type of flow	Quadric signs	OR	AND
Internal			
External			

Figure 4. Differences in the effect of `GROUP_RELATION` for internal and external flows.

2.1.7. Logical grouping among groups

When quadrics are combined within a group, they generate more complex shapes than the individual quadrics. The resulting shapes from a group can be combined with the geometry obtained from all previous groups, using the keyword RELATION_WITH_PREVIOUS. Available options are 'OR', and 'AND', and they work similar to logical grouping of quadrics, except they apply to the entire groups, not individual quadric. A values of 'OR' means a point belongs to the computational domain if the f-value for the current group **or** the f-value for the combination of previous groups is negative. A value of 'AND' means a point belongs to the computational domain if the f-value for the current group **and** the f-value for the combination of previous groups is negative.

2.1.8. Piecewise grouping

When quadrics intersect along planes that are perpendicular to either the x, y, or z-axis, quadrics can be smoothly combined in a piecewise manner. This option is particularly suited to the combination of two or more surfaces of revolution that share the same axis of revolution, typically involving cylinders and cones. In this case, quadrics intersects along planes perpendicular to the axis of revolution, and can be calculated in advance.

To group quadrics in a piecewise fashion, set the group relation to 'PIECEWISE', and define clipping limits for each quadric, corresponding to the intersection planes between quadrics.

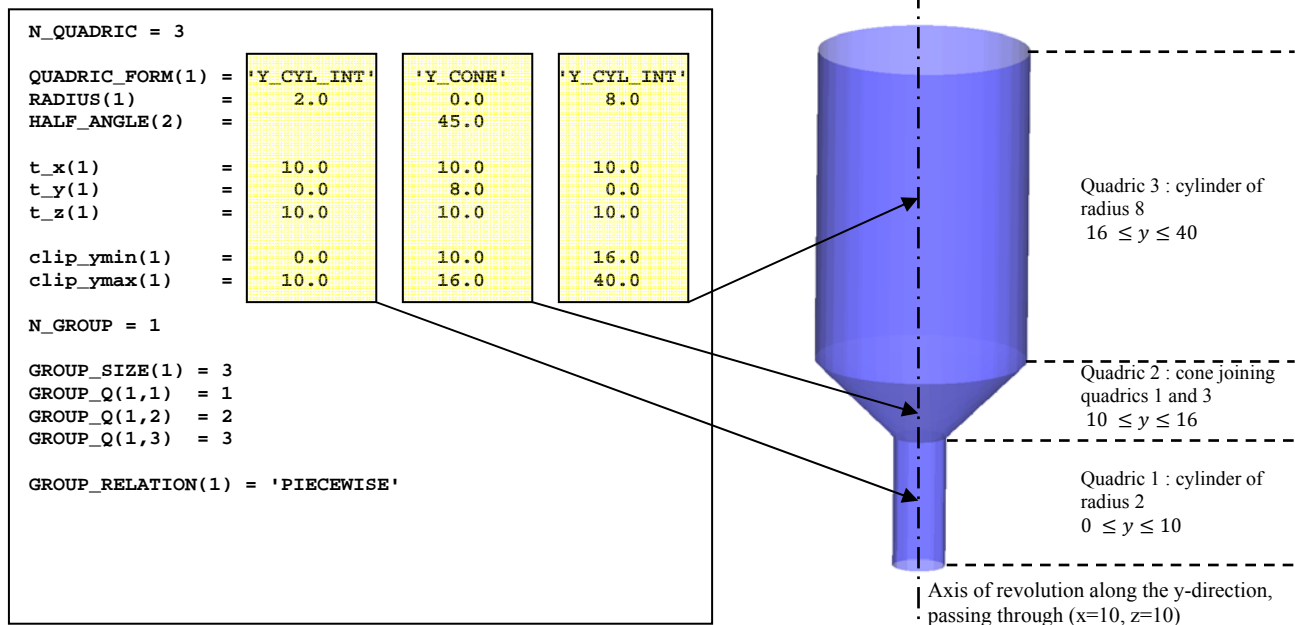Figure 5 illustrates the piecewise grouping of three quadrics.

Figure 5. Example of a piecewise grouping.

2.1.9. Example of quadric combination

This example illustrates the use of several groups to combine quadrics. The corresponding mfix.dat is located in the tutorial folder spoutedbed2. The geometry consists of a spouted bed and a stabilizer. The entire geometry can be described with 7 quadrics, shown in Figure 6a. Quadrics 1, 3, 4, and 5 represent pairs of parallel planes. Quadric 2 represents a pair of non-parallel planes. Quadrics 6 and 7 are two single planes (degenerate form). The geometry is built using the following steps:

Step 1: Define all quadric parameters (see mfix.dat in tutorial folder).
Step 2: Build group 1: Piecewise combination of quadric 4 and 5 (Figure 6b). This creates the stabilizer shape, running through the entire width of the spouted bed.

```
N_GROUP = 3

GROUP_SIZE(1) = 2
GROUP_Q(1,1) = 4
GROUP_Q(1,2) = 5
GROUP_RELATION(1) = 'PIECEWISE'
```

Step 3: Build group 2: Combine quadrics 6 and 7 with `OR` relation (Figure 6c). This creates a mask that will be used to trim group 1.

```
GROUP_SIZE(2) = 2
GROUP_Q(2,1) = 6
GROUP_Q(2,2) = 7
GROUP_RELATION(2) = 'OR'
```

Step 4: Combine groups 1 and 2 with `OR` relation (Figure 6d). The mask (group 2) is applied to group 1.

```
RELATION_WITH_PREVIOUS(2) = 'OR'
```

Step 5: Build group 3: Piecewise combination of quadric 1, 2, and 3 (Figure 6e). This group represents the walls of the spouted bed.

```
GROUP_SIZE(3) = 3
GROUP_Q(3,1) = 1
GROUP_Q(3,2) = 2
GROUP_Q(3,3) = 3
GROUP_RELATION(3) = 'PIECEWISE'
```

Step 6: Combine group 3 with groups 1 and 2, using `AND` relation (Figure 6f) to create the final geometry.

```
RELATION_WITH_PREVIOUS(3) = 'AND'
```
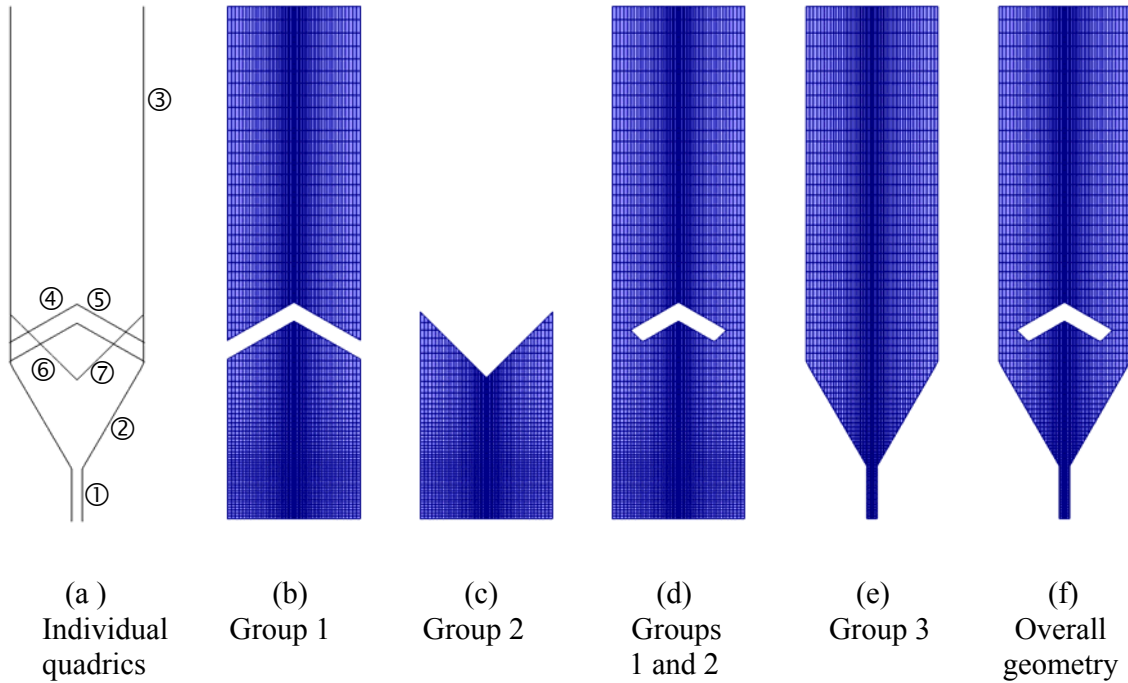
Figure 6. Steps involved in geometry definition of a spouted bed with stabilizer.

2.2. Polygons

For two-dimensional geometries, the boundaries can be described by one or a series of polygons. The polygons can be concave or convex. An ordered list of coordinates for each vertex is given by the user. The number of vertices is the same as the number of edges, since the polygon is closed by connecting the last vertex to the first vertex. Open shapes can be created by locating vertices outside of the computational domain defined by (XLENGTH, YLENGTH). The option to use polygon(s) is activated by setting the keyword USE_POLYGON = .TRUE., and the data is read from the file poly.dat. The file structure is as follows:
Line 1-13: file header. User input starts at line 14
Line 14: Number of polygon(s)
Line 15: Number of vertices defining first polygon, followed by the polygon sign (either 1.0 or -1.0). A value of 1.0 means the interior region is removed from computation (blocked cells) and a value of -1.0 means the interior region is part of the computational domain (fluid cells).
Lines 16 and below: For each vertex, provide x and y coordinate, followed by the boundary condition ID of the corresponding edge. The same sequence (number of vertices, polygon sign, list of coordinates, and boundary condition ID) is repeated as needed for each additional polygon.

<u>Example:</u>

An example is given below to define a square region using polygon data (Figure 7). Note that in this case, this could also be done using the regular boundary conditions in mfix.dat. The four vertices are located at the corners of the pressure cells. The four corresponding edges are assigned the same

boundary condition ID (2). To define the boundary as a no-slip wall, the boundary condition type `BC_TYPE(2) = 'CG_NSW'` must be defined in mfix.dat (see section 3).
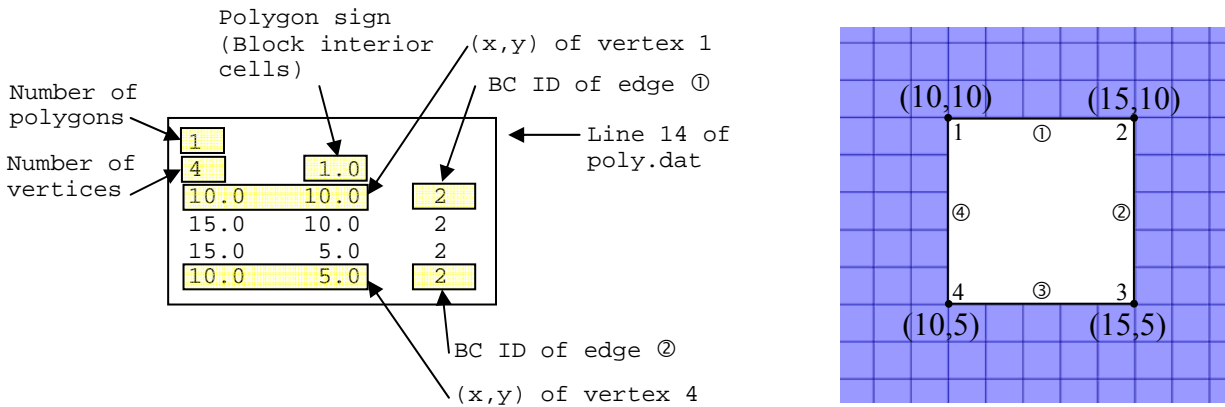
Figure 7. Example of polygon data entry.

2.3. User-defined function

To describe the geometry through a user-defined function, the subroutine `eval_usr_fct.f` must be modified, and MFIX must be compiled. The same convention applies, that is regions where $f(x, y, z)$ is positive are excluded from the computational domain (blocked cells), and regions where $f(x, y, z)$ is negative are part of the computational domain (fluid cells). The boundaries are located where $f(x, y, z)$ is zero (within the tolerance `TOL_F`). This option is activated by setting `N_USR_DEF = 1`.

The subroutine must assign a value to the following variables:

`f_usr` : value of $f(x, y, z)$ (Real), and

`BCID` : boundary condition ID of the cut cell associated with the boundary (Integer)

The subroutine `eval_usr_fct.f` is located in `mfix/model/cartesian_grid` folder.
Before modifying the file, it is recommended to create the sub-folder `cartesian_grid` in the current run directory, and copy the file `eval_usr_fct.f` into this directory. Once the file is modified, MFIX must be compiled for the modifications to take effect. The makefile will look into the current run directory structure and use the file located in the `cartesian_grid` folder. For example, if the current run directory is `wavy`, the subroutine `eval_usr_fct.f` should be placed in `wavy/cartesian_grid`.

3. Boundary condition specification

Recall that the boundaries can be specified using one of the following three methods: quadric surface(s), polygon(s) or user-defined function.

Each cut cell is assigned a boundary condition identification number (BC_ID). For a given quadric surface, the BC_ID is stored in the variable `BC_ID_Q`. For polygon data, the BC_ID is defined for each polygon edge, in the file `poly.dat` (see section 2.2). For a user-defined function, the BC_ID must be defined in the file `eval_usr_fct.f` (see section 2.3).

The BC_ID is linked to a type of boundary condition, similar to what is done for standard cells. Current available boundary conditions types for cut cells include No-slip wall (`'CG_NSW'`) and Free-slip wall (`'CG_FSW'`). The boundary condition type applies to both gas and solids phases.

In the example below, BC_ID 12 is assigned to quadric 1. Any cut cell related to Quadric 1, will be treated as a no-slip wall.

```
BC_ID_Q(1)  = 12
BC_TYPE(12) = 'CG_NSW'
```

A partial slip boundary condition is being implemented, but is not fully functional. Only the two extreme cases that are equivalent to no-slip and free-slip wall are currently available. This is useful to define different boundary condition for the gas and solid phase. The input is similar to the partial-slip wall for regular cells. Specifying a boundary conditions type of 'CG_PSW' requires the input of the coefficient h_w for gas and solid phase. For free-slip, set $h_w = 0$, and for no-slip, leave h_w undefined ($h_w = \infty$). The wall velocity v_w is zero and cannot currently be modified.

For example, BC_ID 10, linked to quadric ID 1 will specify no-slip wall for the gas phase and free-slip wall for the solids phase:

```
BC_ID_Q(1)  = 10
BC_TYPE(10) = 'CG_PSW'

BC_HW_s(10,1) = 0.0
```

Note: The BC_ID assigned to cut cells can be visualized from the vtk file, by including the flag 101 into the list of vtk variables VTK_VAR (see section 6).

4.   Removal of small pressure cells

The intersection of quadrics, polygons, or user-defined function with the background grid is likely to generate small cells, which can increase the stiffness of the system of equations. Small cells can be removed from the computational domain by slightly altering the geometry. When the boundary intersects the background grid near a corner point, the intersection point is moved to the corresponding corner point (i.e., the intersection point is "snapped" onto the grid). The tolerance parameter TOL_SNAP defines the sensitivity of the snapping procedure. It is expressed in term of a fraction of the current cell edge along which the intersection point is located. For example, specifying TOL_SNAP = 0.01 will snap an intersection points to a corner point if they are separated by less than 1 % of the cell edge. A value of TOL_SNAP = 0.0 turns off this option (default value). Figure 8 illustrates this procedure with an exaggerated value of TOL_SNAP = 0.10 for clarity. Typical values are in the range 0.01 to 0.05.

(a) TOL_SNAP = 0.00   (b) TOL_SNAP = 0.10
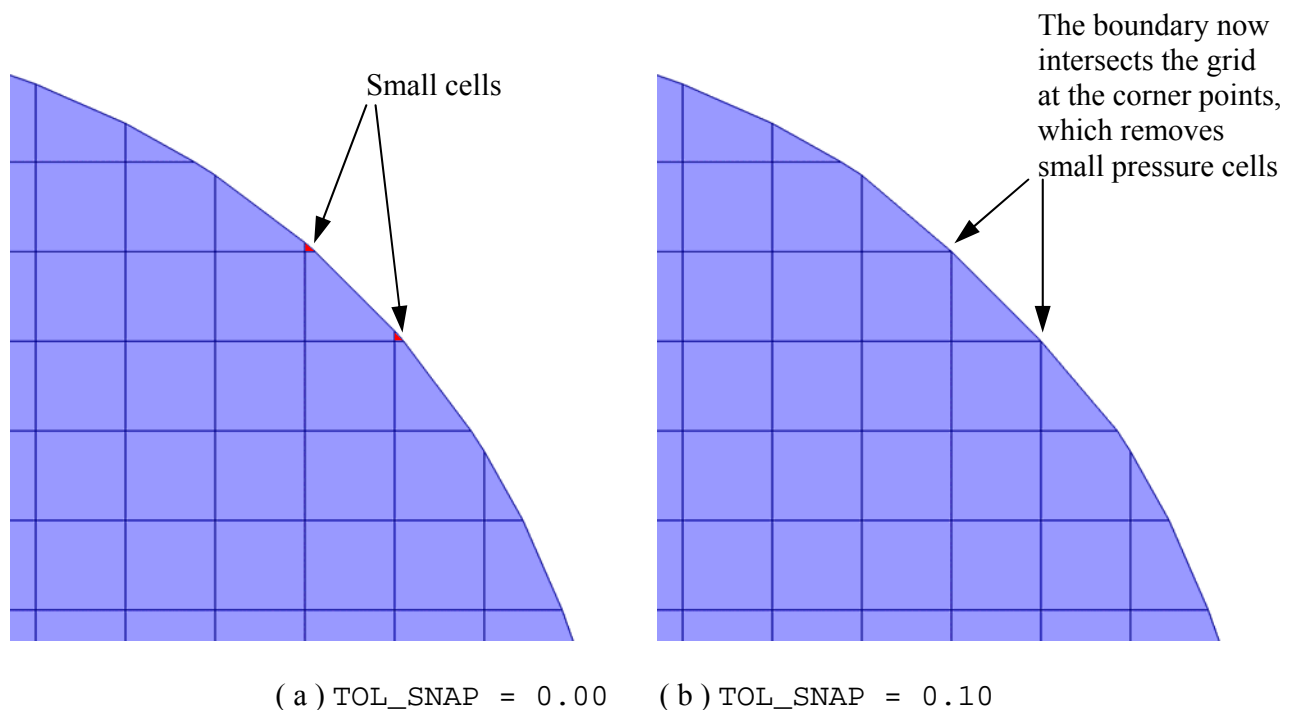
Figure 8. Effect of TOL_SNAP on small cells.

The definition of a small cell is arbitrary. One way to characterize a cell as small is to compare the volume of the cut cell with the volume of the original (uncut) cell. If the ratio of the volume of a cut cell over the volume of the original cell is less than the tolerance TOL_SMALL_CELL, a cut cell is flagged as small. A value of TOL_SMALL_CELL = 0.01 would flag cell IJK as small if VOL(IJK) < 0.01 DX(I)*DY(J)*DZ(K), where VOL(IJK) is the volume of the cut cell, and DX(I), DX(J), DZ(K) are the original cell dimensions. Currently, any small pressure cell remaining after the snapping procedure will be treated the same as other cut cells (i.e., they are not merged with adjacent cells). For a given value of TOL_SMALL_CELL, the number of small cells can be reduced by increasing TOL_SNAP.

5. Utilities

5.1. Progress bar

A progress bar can be displayed to visualize the progress of the major pre-processing steps step. This option is activated with the keyword `PRINT_PROGRESS_BAR=.TRUE.`. The appearance of the progress bar can be controlled by changing its length (keyword `BAR_WIDTH`) and the character used to create the bar (`BAR_CHAR`). By default, the progress bar is updated by increments of 5%. The update frequency is controlled by the keyword `BAR_RESOLUTION`. The progress is also printed at the center of the bar. Figure 9 shows the appearance of the progress bar. Showing the progress bar is mostly useful for fine grids and three dimensional grids to monitor the progress of the pre-processing stage.

```
INTERSECTING GEOMETRY WITH SCALAR CELLS...
|=====================  75.0 % ========                 |
```

Figure 9. Appearance of the progress bar.

5.2. Dashboard

While MFIX is running, progress in the simulation can be followed on the screen. The screen output can be redirected to a file. For example, issuing the command `./mfix | tee run.log` will launch MFIX, display the output on the screen as well as in the file `run.log`.

A summary of the simulation progress can also be written in the file `DASHBOARD.TXT`, which will be referred to as the dashboard. This option is activated by setting the keyword `WRITE_DASHBOARD = .TRUE.` (default value is `.FALSE.`). When this option is activated, the dashboard is updated at every time step. The frequency can be increased by setting a value of `F_DASHBOARD` larger than one.

Figure 10 illustrates the dashboard. The top portion contains descriptions of the simulation (provided in mfix.dat), the run status, elapsed CPU time and estimated CPU time left, the name of the latest vtk file written, and the type of run (serial or parallel).

A table summarizes values of some variables. The current, minimum and maximum values are displayed, and a progress bar corresponding to the numerical value is displayed on the right-hand-side. The sign beside the variable name DT indicates whether the last time step was increased (+) or decreased (-) during the computation. Units for Time and DT are seconds. Sm is the solid inventory in the domain (expressed in grams for cgs unit system and kilograms for SI unit system). NIT is the number of iterations for the current time step. The name of the variable yielding the maximum residual is displayed in the bottom row. A time stamps is printed at the bottom of the dashboard.

The script `show_dashboard` located in `mfix/tools/Dashboard` can be used to display and automatically refresh the dashboard. For example, issuing the command `./mfix.exe > run.log &` will run MFIX in the background, and invoking `show_dashboard 2` will display the dashboard and update it every 2 seconds.

```
|--------------------------------------------------------------------------|
|                            MFIX DASHBOARD                                |
|--------------------------------------------------------------------------|
|                                                                          |
| RUN_NAME             = HG                                                |
| Description          = 2D HOURGLASS - NO SLIP WALL                       |
| Run Status           = In Progress...                                    |
| CPU time elapsed     =    1.7 min                                        |
| CPU time left        =    8.4 min                                        |
| Latest vtk file      = HG_22.vtk                                         |
| Serial run                                                               |
|                                                                          |
|--------------------------------------------------------------------------|
|        |         |         |         |          |                       |
|  Name  |  Value  |   Min   |   Max   | % of max |0%      Progress    100%|
|_____|_____|_____|_____|_____|_____|
|        |         |         |         |          |                       |
| Time   | 0.22E+00| 0.00E+00| 0.10E+01|  22.0 %  |=====                  |
| DT (+) | 0.86E-03| 0.77E-03| 0.20E-02|  43.0 %  |===========            |
| Sm     | 0.73E+02| 0.73E+02| 0.73E+02| 100.0 %  |======================|
| NIT    |       11|       11|       39|  28.2 %  |=======                |
| Max res|    P1   |         |         |          |                       |
|_____|_____|_____|_____|_____|_____|
| Last updated at: 11:25:19 on: 07/02/2009                                 |
```

Figure 10. Appearance of the dashboard.

6. Post-processing

When the Cartesian grid option is activated, the data can be saved into vtk files [5], which can be visualized with the post-processing tool Paraview [6] or Visit [7]. Visualizing the data using the .RES file will not show any cut cell. To save output files in vtk format, set the keyword WRITE_VTK_FILES = .TRUE.. This option is only valid when CARTESIAN_GRID = .TRUE.. When the Cartesian grid capability is turned-off, the .RES file can be used for data visualization, since no cut cells are generated.
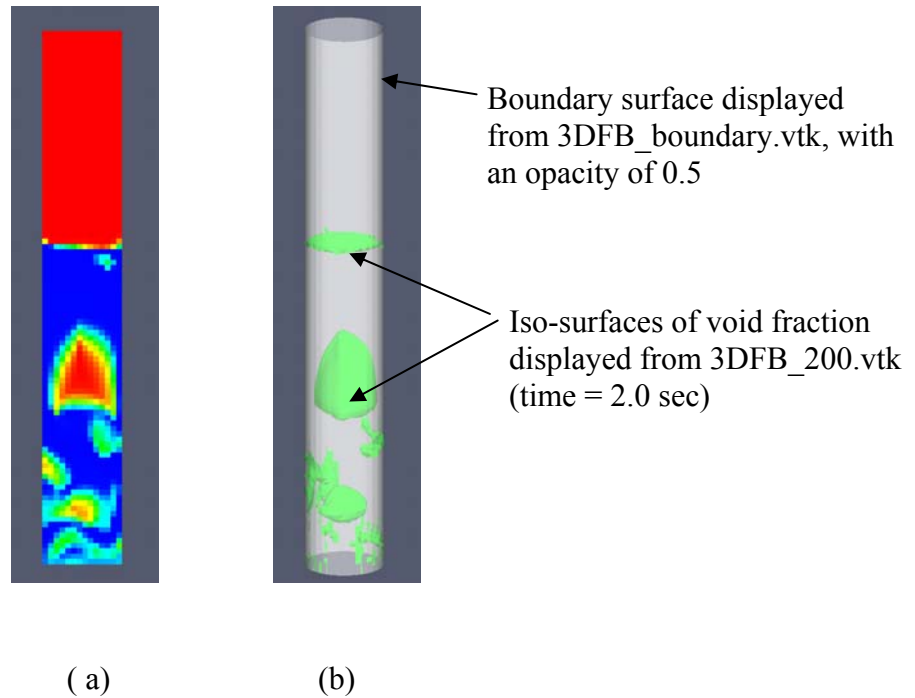
The name of the vtk file is based on RUN_NAME, defined in mfix.dat. For transient computations, a sequential integer is appended to the filename, starting at 0, to create a series of files (use TIME_DEPENDENT_FILENAME = .TRUE.). The file(s) extension is always .vtk. The keyword FRAME will offset the starting integer. For example, assume we have the following files after a successful run: VAR_0.vtk, VAR_1.vtk, ……, VAR_10.vtk. We can restart the simulation (with RUN_TYPE = 'RESTART_1'), and set FRAME = 10. The simulation will restart and the vtk files will not be overwritten, i.e., the next vtk file will be VAR_11.vtk.

The list of variables stored into the vtk file is controlled by a list of integers stored in VTK_VAR. Current available flags for VTK_VAR are:

1:      Void fraction (EP_g)
2:      Gas pressure, solids pressure (P_g, P_star)
3:      Gas velocity (U_g, V_g, W_g)
4:      Solids velocity (U_s, V_s, W_s)
5:      Solids density (ROP_s)
6:      Gas and solids temperature (T_g, T_s1, T_s2)
7:      Gas and solids mass fractions (X_g, X_s)
8:      Granular temperature (G)
11:     Turbulence quantities (k and epsilon)
12:     Gas vorticity magnitude and Lambda_2 (VORTICITY, LAMBDA_2)
100:    Processor assigned to scalar cell (Partition)
101:    Boundary condition flag for scalar cell (BC_ID)

For example, setting VTK_VAR = 1 3 4 will store the void fraction, gas velocity and solids velocity into each vtk file.

A boundary file, starting with the value of RUN_NAME, and ending with _boundary.vtk is written during the preprocessing stage. It can be used to easily visualize the boundary. It is mostly useful for three-dimensional geometries. If it is desired to look through the boundary, the boundary must be displayed with a low opacity value. Figure 11 shows data obtained from the 3dfb tutorial (three-dimensional fluidized bed), with a cylindrical boundary. Figure 11a show void fraction contour along a vertical slice, whereas Figure 11b shows the boundary, and isosurfaces of the void fraction.

Boundary surface displayed from 3DFB_boundary.vtk, with an opacity of 0.5

Iso-surfaces of void fraction displayed from 3DFB_200.vtk (time = 2.0 sec)

(a)   (b)

Figure 11. Instantaneous data visualization (a) void fraction contour along a vertical slice and (b) boundary and isosurfaces of void fraction.

The file CUT_CELL.LOG contains statistics about the grid, with minimum and maximum values of geometrical quantities, interpolation factors, and non-orthogonality correction terms.

For two-dimensional simulations, the file ani_cutcell.vtk will be generated at the beginning of the run. It will be used by animate_mfix to display cut cells. Cut cells will not be displayed by animate_mfix for three-dimensional computations. The only option to visualize cut cells in 3D is to use the VTK files. Note that animate_mfix uses the SPX_DT values between consecutive frames, not the VTK_DT values.
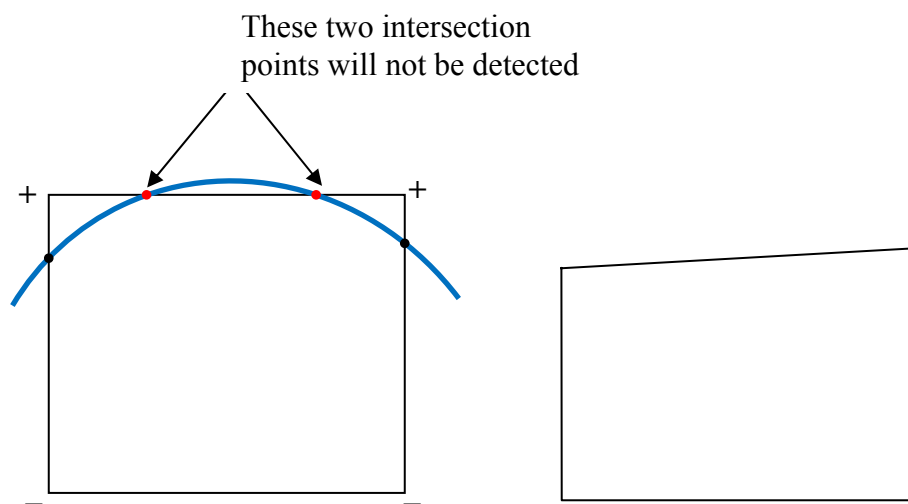
7. Code modifications

7.1. Preprocessing

The procedure to identify cut cells is as follows. An intersection search is performed along each edge of the cell (the search is performed only if the function $f(x, y, z)$ has opposite signs at the edge extremities). The location of the intersection points may be altered depending on the value of TOL_SNAP (see section 4). Once all intersections points are determined within a cell, the connectivity is established for each cut-cell and its faces, and re-ordered if necessary. Faces are described as convex polygons, and the face areas are computed similar to convex polygon areas. To compute cut-cell volumes, cut-cells are split into pyramids, and volumes of pyramids are computed and added to form the cell volume.

Velocity nodes are placed at the center of pressure cell faces. All interpolation correction terms and non-orthogonality correction terms are computed from the new velocity node locations. An example of a computational grid, with the location of velocity nodes is shown in Figure 12, where the thick solid line is the domain boundary.
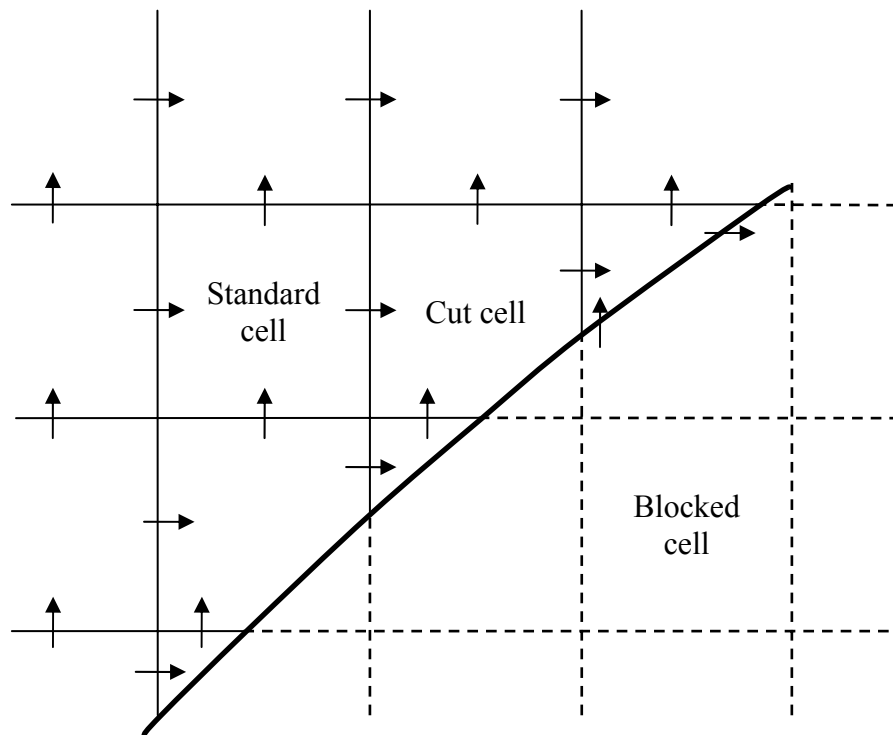
Figure 12. Example of scalar cut-cells with location of velocity nodes.

Note: It is assumed that the geometry (surface quadrics, polygon data, or user-defined function) intersects an edge no more than one time. If there are actually two intersections along the same edge, they will not be detected since the function $f(x, y, z)$ will have the same sign at the edge extremities (see Figure 13). The sign of $f(x, y, z)$ is indicated at the corners.

These two intersection
points will not be detected

(a) Original cell and quadric surface   (b) shape of corresponding cut cell

Figure 13. Example of misrepresentation of boundary.

7.2. No-slip wall

Due to the relocation of velocity nodes at the face centers, the velocity u_e may not be located at the center of the east face (Figure 14). Instead, the velocity at the face center u_{ec} is computed assuming zero velocity at the wall using the ratio of normal distances to the wall, as given in Equation (8). Figure 14 shows the notation used to define the interpolation correction factor α_e. In the code, this correction factor is named `alpha_ue_c`. The same types of correction terms are defined for other velocity components. Their names start with the same root `alpha_`, followed by two letters, representing the velocity component (u,v, or w), and the face (e for East , n for North, t for Top), respectively. For example, in the u-momentum cell, the following three correction terms will be computed (in 3D): `alpha_ue_c`, `alpha_un_c`, `alpha_ut_c`. Wall distances are named `DELH` in the code.

$$u_{ec} = \frac{\Delta h_{ec}}{\Delta h_e} u_e = \alpha_e u_e \tag{8}$$
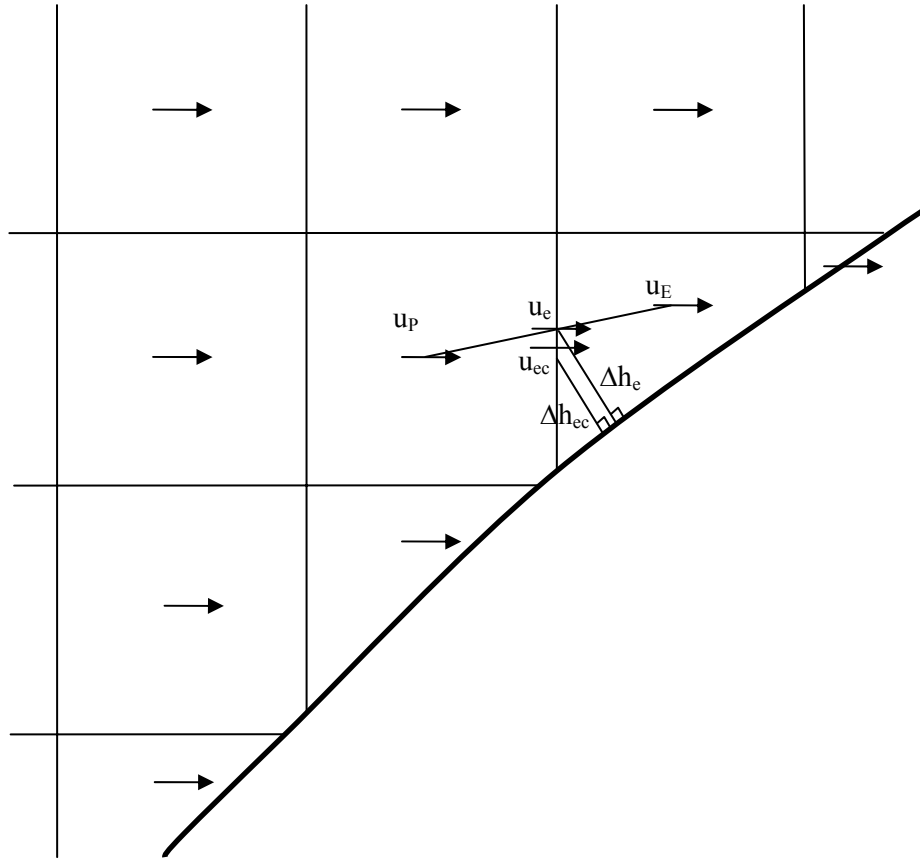
Figure 14. Example of x-velocity cut-cells, with illustration of the interpolation correction for u_{ec} at east face.

In the u-momentum cell, other velocity components are interpolated at the face center based on the location of adjacent velocity nodes (Figure 15). For example, the v-component of velocity along the north face would be interpolated from

$$v_n = \theta_{ne} v_{ne} + \theta_{nw} v_{nw} \tag{9}$$

where

$$\theta_{ne} = \frac{\Delta x_w}{\Delta x_{we}}, \theta_{nw} = \frac{\Delta x_e}{\Delta x_{we}}, \text{and } \Delta x_{we} = \Delta x_w + \Delta x_e \tag{10}$$

In the code, θ_{ne} in the u-momentum cell is named theta_u_ne. Other interpolation correction terms are computed for other velocity components, and named in a similar fashion.
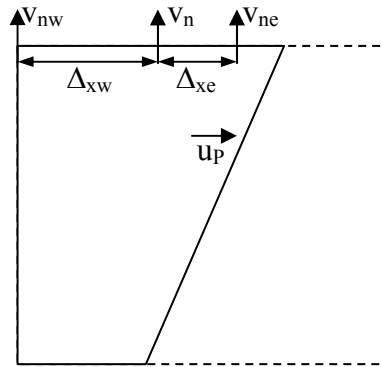
Figure 15. Interpolation of v_n at north face of u-velocity cell.

Due to the relocation of velocity nodes at the face centers, vector $\vec{s}$ joining points P and E, may not be perpendicular to the east face (Figure 16). Therefore, the derivative $\partial u / \partial x$ is approximated by equations (11) using the fact that velocity is zero at the wall. The vector $\vec{N}$ is perpendicular to the wall and passes through point e.

$$\frac{\partial u}{\partial x} \approx \frac{u_E - u_P}{|S_x|} - \frac{1}{S_x}\left(S_y \frac{\partial u}{\partial y} + S_z \frac{\partial u}{\partial z}\right) \tag{11}$$
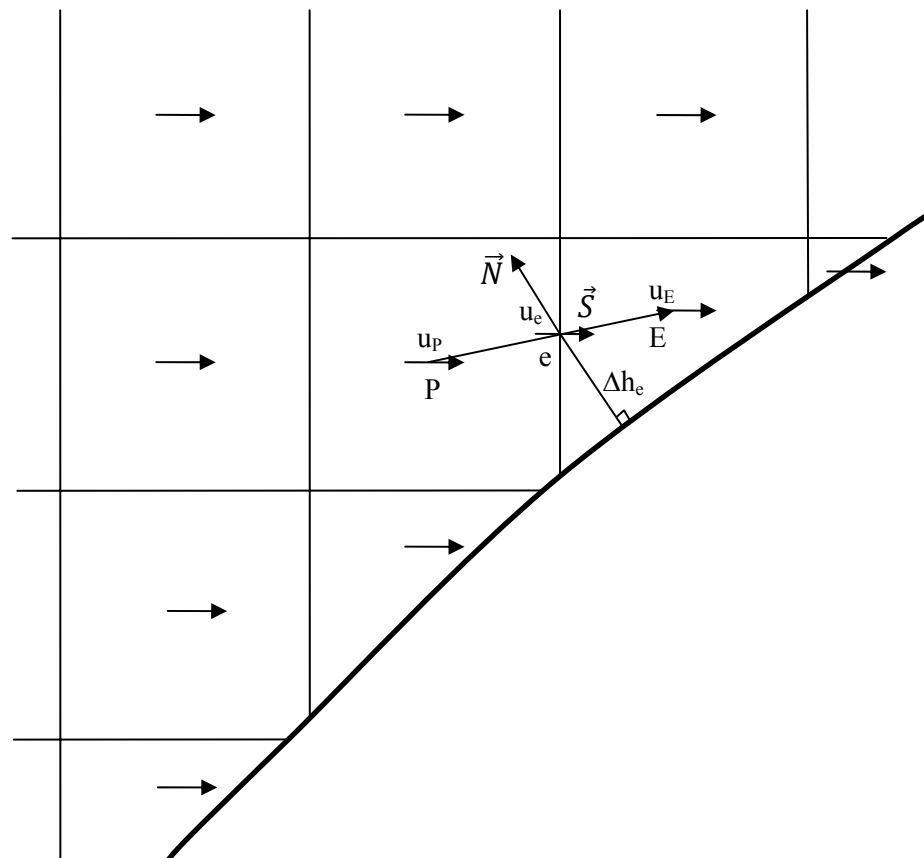
where

$$\frac{\partial u}{\partial y} \approx \frac{N_y u_e}{\Delta h_e} \quad \text{and} \quad \frac{\partial u}{\partial z} \approx \frac{N_z u_e}{\Delta h_e} \tag{12}$$

Therefore,

$$\frac{\partial u}{\partial x} \approx \frac{u_E - u_P}{|S_x|} - \frac{S_y N_y + S_z N_z}{S_x \Delta h_e} u_e = \frac{u_E - u_P}{|S_x|} - \text{NOC } u_e \tag{13}$$

The non-orthogonality correction term $\text{NOC} = \frac{S_y N_y + S_z N_z}{S_x \Delta h_e}$ can be computed during preprocessing. In the code, it is named NOC_U_E. Other non-orthogonality correction terms are computed and named similarly.


Figure 16. Example of x-velocity cut cells, with illustration of the non-orthogonality correction for the evaluation of the derivative $\partial u / \partial x$ at east face.

It is assumed that in the cut cell, the velocity is tangential to the wall. The shear force acting on the fluid due to the wall shear stress is parallel to the velocity vector (Figure 17), and can be written as

$$\vec{F} \approx -\mu \Delta A_{cut} \frac{\partial \vec{u}}{\partial n} \approx -\mu \Delta A_{cut} \left(\frac{u}{\Delta h} \vec{\imath} + \frac{v}{\Delta h} \vec{\jmath} + \frac{w}{\Delta h} \vec{k} \right) = F_x \vec{\imath} + F_y \vec{\jmath} + F_z \vec{k} \qquad (14)$$

Where ΔA_{cut} is the surface area of the cut face. Therefore, the contribution of the cut face can be applied implicitly along each direction (e.g., $F_x = -\mu \Delta A_{cut} \frac{u}{\Delta h}$ in the x-direction).
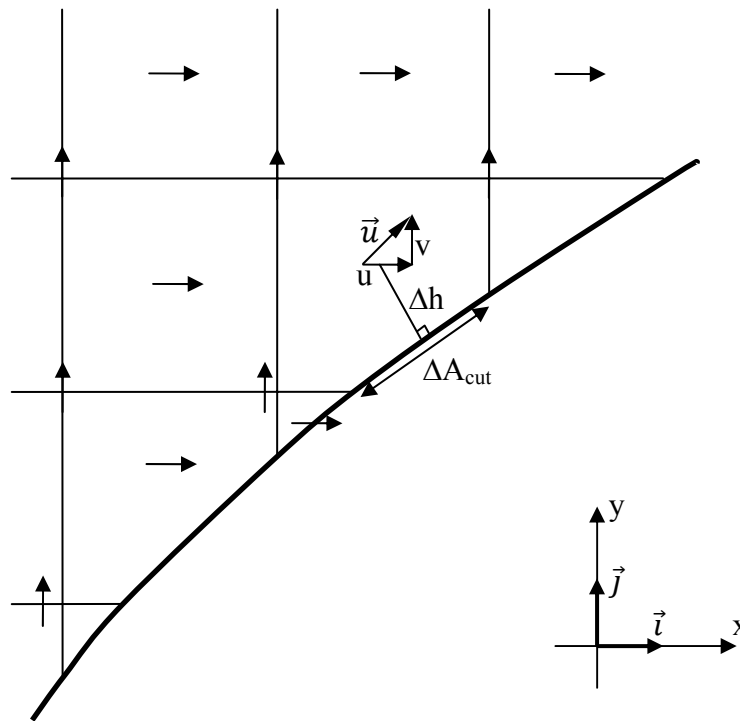
**Figure 17.** Additional wall shear stress arising from the cut-face.

7.3. Free-Slip wall

For free-slip walls, the assumption of zero wall velocity is not valid. Instead, the velocity gradient normal to the wall is assumed to be zero. Therefore, the correction term α_e is set to one, and the non-orthogonality correction terms are set to zero. No additional shear force term is added.

7.4. Problematic cells

A wide variety of cut cells are typically generated, and cut cells vary in shape and size. Figure 18 illustrates various shapes of cut cells. The generation of small pressure cells leads to stability issues. Section 4 describes a method to reduce the number of small pressure cells, by slightly altering the geometry. Due to the staggered grid formulation, velocity cells require pairs of pressure nodes (East/West, North/South and Top/Bottom) to compute pressure gradients. However, when some pressure cells are removed (blocked cells), the corresponding velocity cells will only have one pressure node and the pressure gradient cannot be computed in that direction. When this situation occurs, the velocity cell is flagged as a "wall cell", and the momentum equation is not solved for this cell. A velocity is specified, which depends on the type of boundary condition assigned to the cell. For no-slip wall (BC_TYPE = 'CG_NSW'), a zero velocity is assigned. For Free-slip wall (BC_TYPE = 'CG_FSW'), the velocity is the same as the velocity of an adjacent cell. The adjacent cell is called the "master" cell of the "wall cell".

The treatment of single-pressure velocity cells effectively limits their size to half the size of a standard cell, in any direction. This criterion automatically avoids problems linked to small velocity cells.
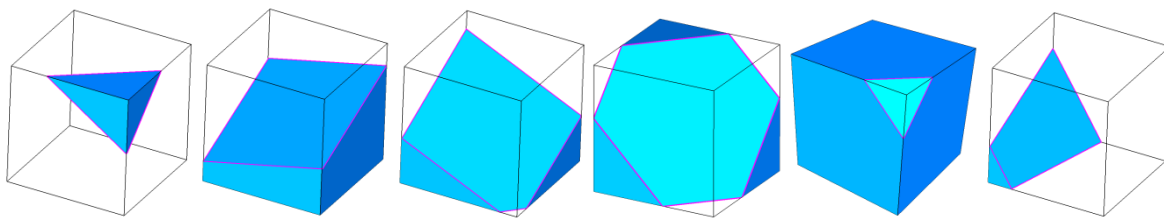
Figure 18. Examples of cut cell shapes.

8. Limitations

The Cartesian grid cut-cell technique is currently available for solving conservation of mass and momentum for gas and solids phases.
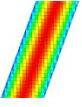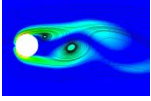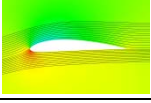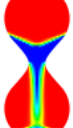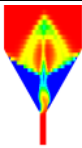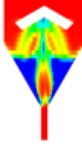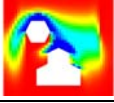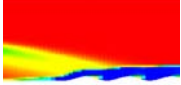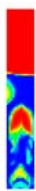
Currently, the method is not applicable to:

- Energy equation
- Granular temperature equation
- k-epsilon equation
- Discrete Element Method

The only boundary conditions available for cut cells are No-slip wall and Free-slip wall.

9. Tutorial files

The following tutorial files are located in `/mfix/cartesian_grid_tutorials`.

Snapshot	Folder	Method to describe geometry	Dimension	Phase (G=Gas, S=Solid)	Description
	channel	One quadric surface	2D	G	Flow in a skewed channel
	cylinder	One quadric surface	2D	G	Flow over a cylinder
	naca4412	Polygon data	2D	G	Flow over an airfoil
	hourglass	Three quadric surfaces	2D	G/S	Solids falling in an hourglass-shaped container
	spoutedbed1	Three quadric surfaces	2D	G/S	Spouted bed
	spoutedbed2	7 quadrics, 3 groups	2D	G/S	Spouted bed with stabilizer
	polygons	Polygon data	2D	G/S	Pack of solids falling on polygons
	wavy	User-defined function	2D	G/S	Solid jet impinging on a wavy surface
	3dfb	One quadric surface	3D	G/S	3D fluidized bed
	cyclone	12 quadrics, 6 groups	3D	G/S	Flow of particles through a cyclone

10. Cartesian Grid keywords

keyword(dimension)	Type	Description
CARTESIAN_GRID [.FALSE.] .TRUE.	L	Activate Cartesian grid cut cell technique. Do not use Cartesian grid cut cell technique. Use Cartesian grid cut cell technique. One of the following methods must be used to define the geometry: Quadric surfaces (N_QUADRIC >=1), polygons (USE_POLYGON = .TRUE.), or a user-defined function (N_USR_DEF = 1).
N_QUADRIC [0]	I	Number of quadric surfaces defining the boundaries (<=100).
USE_POLYGON [.FALSE.] .TRUE.	L	Use polygons to describe geometry. Do not use polygons. Read polygon data (for 2D geometry only) from poly.dat.
N_USR_DEF [0]	I	Number of user-defined functions (currently limited to 0 or 1). If set to 1, the geometry is defined in the user subroutine eval_usr_fct.f.
QUADRIC_FORM ['NORMAL']	C	Form of the quadric surface equation. Use normal form, as defined in Equation (1). Regions where f(x,y,z) < 0 are part of the computational domain. Regions where f(x,y,z) > 0 are excluded from the computational domain. See section 2.1.2 for pre-defined quadric surfaces (planes, cylinder and cones)
LAMBDA_X() [ZERO]	DP	Coefficient lambda_x in Equation (1) ('NORMAL' form) or x-component of normal vector defining plane in Equation (5) ('DEGENERATE' form).
LAMBDA_Y() [ZERO]	DP	Coefficient lambda_y in Equation (1) ('NORMAL' form) or y-component of normal vector defining plane in Equation (5) ('DEGENERATE' form).
LAMBDA_Z() [ZERO]	DP	Coefficient lambda_z in Equation (1) ('NORMAL' form) or z-component of normal vector defining plane in

		Equation (5) (`'DEGENERATE'` form).
DQUADRIC() [ZERO]	DP	Coefficient d in Equation (1).
N_X() [ZERO]	DP	x-component of normal vector defining the plane (used when QUADRIC_FORM = 'PLANE')
N_Y() [ZERO]	DP	y-component of normal vector defining the plane (used when QUADRIC_FORM = 'PLANE')
N_Z() [ZERO]	DP	z-component of normal vector defining the plane (used when QUADRIC_FORM = 'PLANE')
RADIUS() [ZERO]	DP	Cylinder radius (used when QUADRIC_FORM = '*_CYL_***')
HALF_ANGLE() [ZERO]	DP	Cone half angle, expressed in degrees (used when QUADRIC_FORM = '*_CONE')
THETA_X() [ZERO]	DP	Rotation angle with respect to x-axis (Degrees).
THETA_Y() [ZERO]	DP	Rotation angle with respect to y-axis (Degrees).
THETA_Z() [ZERO]	DP	Rotation angle with respect to z-axis (Degrees).
T_X() [ZERO]	DP	Translation in x-direction.
T_Y() [ZERO]	DP	Translation in y-direction.
T_Z() [ZERO]	DP	Translation in z-direction.
CLIP_XMIN() [- LARGE_NUMBER]	DP	Lower x-limit where the quadric is defined.
CLIP_XMAX() [LARGE_NUMBER]	DP	Upper x-limit where the quadric is defined.
CLIP_YMIN() [- LARGE_NUMBER]	DP	Lower y-limit where the quadric is defined.
CLIP_YMAX() [LARGE_NUMBER]	DP	Upper y-limit where the quadric is defined.
CLIP_ZMIN() [- LARGE_NUMBER]	DP	Lower z-limit where the quadric is defined.
CLIP_ZMAX() [LARGE_NUMBER]	DP	Upper z-limit where the quadric is defined.

`FLUID_IN_CLIPPED_REGION`	L	Flag defining the type of cells that are outside of the zone defined by [`CLIP_XMIN;CLIP_XMAX`], [`CLIP_YMIN;CLIP_YMAX`],[`CLIP_ZMIN;CLIP_ZMAX`].
[`.TRUE.`]		Treat cells as fluid cells.
`.FALSE.`		Remove cells from computational domain.
`BC_ID_Q()` [`UNDEFINED_I`]	I	Boundary condition flag
`N_GROUP` [`1`]	I	Number of group(s) of quadrics (<=50).
`GROUP_SIZE()` [`GROUP_SIZE(1)=1`]	I	Number of quadrics in the group.
`GROUP_Q(I,J)` [`Q(1,1) = 1`]	I	Quadric ID of Jth quadric assigned to group I.
`GROUP_RELATION()`	C	Relation among quadrics of a same group.
[`'OR'`]		A point belongs to the computational domain if at least one of f(x,y,z) among all quadrics is negative
`'AND'`		A point belongs to the computational domain if all of f(x,y,z) among all quadrics are negative
`RELATION_WITH_PREVIOUS`	C	Relation between current group and combination of all previous groups.
[`'OR'`]		A point belongs to the computational domain if f-value for the current group **or** f-value for the combination of previous groups is negative
`'AND'`		A point belongs to the computational domain if f-value for the current group **and** f-value for the combination of previous groups is negative
`TOL_SNAP` [`0.0`]	DP	Tolerance used to snap an intersection point onto an existing cell corner (expressed as a fraction of edge length, between 0.0 and 0.5).
`TOL_DELH` [`0.0`]	DP	Tolerance used to limit acceptable values of normal distance to the wall (expressed as a fraction of cell diagonal, between 0.0 and 1.0).
`TOL_SMALL_CELL` [`0.01`]	DP	Tolerance used to detect small cells (expressed as a fraction of cell volume, between 0.0 and 1.0).
`TOL_SMALL_AREA` [`0.01`]	DP	Tolerance used to detect small faces (expressed as a fraction of original face area, between 0.0 and 1.0).

ALPHA_MAX [ONE]	DP	Maximum acceptable value of interpolation correction factor.
TOL_F [1.0D-9]	DP	Tolerance used to find intersection of quadric surfaces or user-defined function with background grid.
TOL_POLY [1.0D-9]	DP	Tolerance used to find intersection of polygon with background grid.
ITERMAX_INT [10000]	I	Maximum number of iterations used to find intersection points.
SET_CORNER_CELLS [.FALSE.]	L	Flag to detect and treat corner cells the same way as in the original MFIX version (i.e. without cut cells). If set to .TRUE., some cut cells may be treated as corner cells.
FAC_DIM_MAX_CUT_CELL [0.25]	DP	Factor used to allocate some cut cell arrays (expressed as a fraction of DIMENSION_3G)
WRITE_VTK_FILES	L	Write vtk files at regular intervals.
[.FALSE.]		Do not write vtk files. If there are cut cells, they will not be displayed from the usual .RES file
.TRUE.		Valid only if CARTESIAN_GRID = .TRUE.
TIME_DEPENDENT_FILENAME	L	Use time-dependent vtk file names
[.TRUE.]		A sequential integer is appended to the vtk filenames as they are written to create a series of files (recommended for transient computation).
.FALSE.		The vtk file overwrites the previous file (recommended for steady-state computation).
VTK_DT [UNDEFINED]	DP	Interval (expressed in seconds of simulation time) at which vtk files are written.
VTK_VAR() [1 2 3 4]	I	List of variables written in vtk files: 1 : Void fraction (EP_g) 2 : Gas pressure, solids pressure (P_g, P_star) 3 : Gas velocity (U_g, V_g, W_g) 4 : Solids velocity (U_s, V_s, W_s) 5 : Solids density (ROP_s) 6 : Gas and solids temperature (T_g, T_s1, T_s2) 7 : Gas and solids mass fractions (X_g, X_s) 8 : Granular temperature (G) 9 : NOT AVAILABLE (User defined scalars) 10 : NOT AVAILABLE (Reaction Rates) 11 : Turbulence quantities (k and ε) 12 : Gas vorticity magnitude and Lambda_2 (VORTICITY, LAMBDA_2)

		100 : Processor assigned to scalar cell (Partition) 101 : Boundary condition flag for scalar cell (BC_ID)
FRAME [-1]	I	Initial frame counter (suffix of vtk file)
PG_OPTION	I	Option for pressure gradient computation in cut cells.
[0]		Use East, North and Top areas of pressure cell (same as standard cells).
1		Use maximum of (East/West), (North/South), and (Top/Bottom) pairs of velocity cells.
2		Use both (East/West), (North/South), and (Top/Bottom) areas of velocity cells.
CG_SAFE_MODE	L	Run code in safe mode.
[.FALSE.]		Runs the code with modified subroutines for cut cell treatment.
.TRUE.		Performs initial preprocessing but use all original MFIX subroutines during flow solution (using only cell volumes and areas of cut cells).
PRINT_WARNINGS [.FALSE.]	L	Prints any warning message encountered during pre-processing on the screen.
CG_UR_FAC() [1.0]	DP	Under-Relaxation factor used in cut cells (only CG_UR_FAC(2) is used).
PRINT_PROGRESS_BAR [.FALSE.]	L	Print a progress bar during each major step of pre-processing stage.
BAR_WIDTH [50]	I	Width of the progress bar (complete status), expressed in number of characters (between 10 and 80).
BAR_CHAR ['=']	C	Character used to create the progress bar.
BAR_RESOLUTION [5.0]	DP	Update frequency of progress bar, expressed in percent of total length (between 1.0 and 100.0).
WRITE_DASHBOARD [.FALSE.]	L	Writes the file DASHBOARD.TXT at regular intervals. The file shows a summary of the simulation progress.
F_DASHBOARD [1]	I	Frequency, expressed in terms of iterations, at which the dashboard is updated.

11. Quick reference

This section provides a summary of basic options to use the Cartesian grid cut-cell technique, for each boundary definition method. It can be used as a checklist before running MFIX.

Quadric surface(s):

☐ Activate Cartesian grid capability: `CARTESIAN_GRID = .TRUE.`
☐ Define the number of quadrics `N_QUADRIC`
☐ For each quadric surface, define quadric parameters, rotation angles, translation, and clipping limits, as necessary.
☐ Assign a boundary condition ID to each quadric (`BC_ID_Q`)
☐ Use groups to combine quadrics if necessary.
☐ Define a boundary condition type for each value of `BC_ID_Q` (`'CG_NSW'` or `'CG_FSW'`)
☐ Set `WRITE_VTK_FILES = .TRUE.` and set list of output variables `VTK_VAR`.

Polygon(s):

☐ Activate Cartesian grid capability: `CARTESIAN_GRID = .TRUE.`
☐ Define polygon data in `poly.dat`. Place `poly.dat` in the run directory.
☐ Set the keyword `USE_POLYGON = .TRUE.`.
☐ Define a boundary condition type for each Boundary condition ID associated with the polygon edges. (`'CG_NSW'` or `'CG_FSW'`)
☐ Set `WRITE_VTK_FILES = .TRUE.` and set list of output variables `VTK_VAR`.

User-defined function:

☐ Place `eval_usr_fct.f` in the sub-directory `cartesian_grid/` in the run directory.
☐ Modify the subroutine `eval_usr_fct.f` to define the function $f(x,y,z)$ (`f_usr`) and boundary condition ID (`BCID`).
☐ Compile MFIX from run directory.
☐ Activate Cartesian grid capability: `CARTESIAN_GRID = .TRUE.`
☐ Set the keyword `USE_USR_DEF = 1`.
☐ For each value of `BCID`, set a boundary condition type (`'CG_NSW'` or `'CG_FSW'`)
☐ Set `WRITE_VTK_FILES = .TRUE.` and set list of output variables `VTK_VAR`.

12. Trouble shooting

It is recommended to go though the tutorials first to get familiar with the various options described in this document. To start a new simulation, copy an existing mfix.dat file and make incremental modifications. When using polygon data, copy `poly.dat` from one of the tutorials (e.g. naca4412) and modify the file. When using user-defined function, copy the subroutine `eval_usr_fct.f` from the tutorial wavy, and modify it.

When several quadrics are combined, define and visualize individual quadrics before combining them, to make sure they are defined properly.

Small cells can be removed (to some extent) by increasing the value of `TOL_SNAP`.

The most common causes of pre-processing failure include (beside incorrect input data):

- Unable to find an intersection point. Try to increase `TOL_F` or `ITERMAX_INT`.

- Too many intersections found in one cell. This occurs when the geometry's local radius of curvature is very small, or the combination of quadrics is not well detected. Refining the grid usually helps solving this problem. This can also occurs when polygon data defines sharp angles. Slightly moving one or more vertices usually helps solving this problem.

- Clipping limits are not well defined when using the piecewise grouping option. The clipping limits must correspond to the plane where quadrics intersect.

When the code seems to be unstable, even with an optimized grid, running the code in safe mode (`CG_SAFE_MODE = .TRUE.`) can help determine if the problem comes from the flow condition. In safe mode, the flow solution proceeds without using any modification introduced by the cut cell technique. The only modified variables are the cell volumes and face areas. If MFIX still fails in safe mode, it is likely due to improper initial conditions or flow properties.

13. References

[1]   M.P. Kirkpatrick, S.W. Armfield, J.H. Kent, "A representation of curved boundaries for the solution of the Navier–Stokes equations on a staggered three-dimensional Cartesian grid," *Journal of Computational Physics*, 184 (2003) 1–36.

[2]   https://mfix.netl.doe.gov/documentation/Readme.pdf

[3]   Syamlal, M.; Rogers, W., O'Brien, T. J. MFIX Documentation: Theory Guide; DOE/METC-94/1004 (DE94000087); U.S. Department of Energy: Morgantown, WV, 1993. https://mfix.netl.doe.gov/documentation/Theory.pdf.

[4]   S. Benyahia, M. Syamlal, T.J. O'Brien, "Summary of MFIX Equations 2005-4", From URL http://www.mfix.org/documentation/MfixEquations2005-4-3.pdf, July 2007.

[5]   http://www.vtk.org

[6]   http://www. paraview.org

[7]   https://wci.llnl.gov/codes/visit