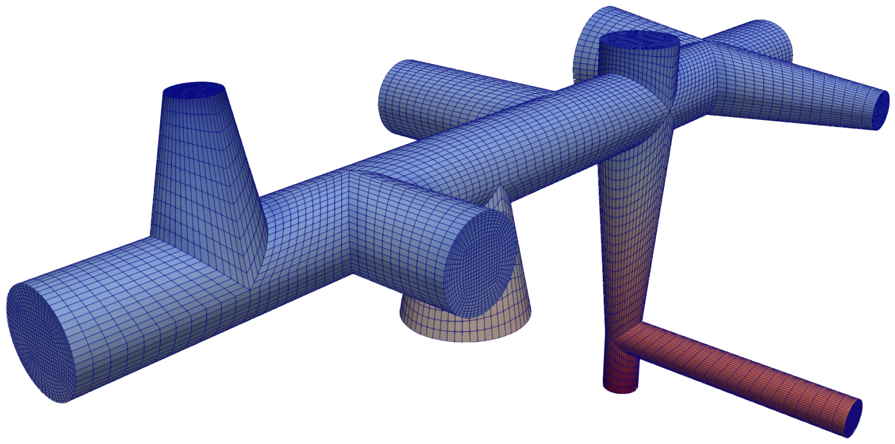
Pipeline mesh generator based on blockMesh version 2.5

Jiri Polansky
jiri.polansky@icloud.com

5.2.2021



Contents

List of Figures

1 Introduction

This report provides a description of an OpenFOAM script for constructing fully hexahedral meshes for a variety of different pipeline geometries. This script allows to change several length scales as well as some useful geometrical parameters concerning the refinement of the mesh, for example close to the walls to resolve boundary layers.

The aim of the mesh generator tool is to extent the blockMesh utility in such a way that it allows to create arbitrary geometries consisting of bends, blind-Ts, Venturi parts, and straight pipes. For the different parts, the following parameters can be adapted to the considered geometry:

- **Bend** diameter of the pipe, radius of curvature, orientation in space
- **Blind-T** diameter of inflow pipe, diameter of outflow pipe, length of blind end, orientation in space
- **Venturi part** inlet diameter, throat diameter, outlet diameter, convection angle, divergence angle, orientation in space
- **Straight pipe** diameter of pipe, orientation in space.

Furthermore, one can prescribe the number of nodes used in flow direction, in radial direction as well as within the boundary layer.

The present report includes the following parts:

- **Mesh generation in OpenFOAM** – Short introduction to mesh generation in OpenFOAM.
- **Utility blockMesh** – This part includes a brief introduction to the blockMesh utility and can be overpassed by advanced openFoam users.
- **MeshGenerator** – Main description, including the code structure, definition of variables, and a manual on how to use the code for pipeline mesh creation.
- **MeshGenerator code** – This part includes C++ code decomposed to main functions with short comments and descriptions. This part is not necessary for starting to use the generator, but it will be useful as inspiration for future extension or using #codeStream for another activity.
- **Examples** – blockMesh dict examples for inspiration and good practice.

2 Mesh generation in OpenFOAM

2.1 The OpenFOAM mesh format

The typical file structure is present on Figure 1. We will run utilities mainly form openFoam working directory. If not, it is necessary use parameter -case and write actual working directory.

```
cavity
    ├── 0
    |    └── ...
    ├── constant
    |    ├── ...
    |    └── polyMesh
    |         ├── points
    |         ├── faces
    |         ├── owner
    |         ├── neighbour
    |         └── boundary
    └── system
         └── ...
```

Figure 1: Structure of working directory

More details on the OpenFOAM mesh format can be found at:
`http://cfd.direct/openfoam/ user-guide/mesh-description/`.

2.2 Mesh generation

Similar to other software (e.g. Abaqus, ANSYS, ...), there are a number of methods available to create a mesh in OpenFOAM; they can be broadly categorised as: generation: creating a mesh using one of the native OpenFOAM meshing utilities e.g. blockMesh, snappyHexMesh, cfMesh, foamyHexMesh, extrudeMesh, etc. A full list of the mesh generation utilities can be found at:

```
$> ls $FOAM_UTILITIES/mesh/generation
```

2.2.1 Conversion

Creating the mesh using third-party software, such as ANSYS, PointWise, ICEM, Gambit, and then using the in-built OpenFOAM mesh conversion utilities to convert it to the OpenFOAM format e.g. fluentMeshToFoam, gmshToFoam, startToFoam, etc. A full list of the mesh conversion utilities can be found at:

```
$> ls $FOAM_UTILITIES/mesh/conversion
```

2.2.2 Manipulation

Once you have a mesh in OpenFOAM, either generated directly or converted, there are a number of OpenFOAM mesh manipulation utilities for checking and modifying the mesh, if necessary, such as checkMesh, rotateMesh, transformPoints, tetDecomposition, etc. A full list of the mesh manipulation utilities can be found at:

```
$> ls $FOAM_UTILITIES/mesh/manipulation
```

2.2.3 Boundary

Boundary include a list of so-called patches, where patches are distinct regions of the mesh boundary, so that distinct boundary conditions can be applied to separate regions of the boundary. Each boundary patch definition comprises: a type, the number of faces on the patch, and the index of the first face on the patch, example:

```
        movingWall {
        type patch;
        nFaces 20;
        startFace 760;
        } ...
```

Boundaries informations is located on directory `../constant/polyMeshboundary`.

3 Utility blockMesh

This section describes the mesh generation utility, blockMesh, supplied with OpenFOAM. The blockMesh utility creates parametric meshes with grading and curved edges.

The mesh is generated from a dictionary file named blockMeshDict located in the system (or constant/polyMesh) directory of a case. blockMesh reads this dictionary, generates the mesh and writes out the mesh data to points and faces, cells and boundary files in the same directory.

The principle behind blockMesh is to decompose the domain geometry into a set of 1 or more three dimensional, hexahedral blocks. Edges of the blocks can be straight lines, arcs or splines. The mesh is ostensibly specified as a number of cells in each direction of the block, sufficient information for blockMesh to generate the mesh data.

Each block of the geometry is defined by 8 vertices, one at each corner of a hexahedron. The vertices are written in a list so that each vertex can be accessed using its label, remembering that OpenFOAM always uses the C++ convention that the first element of the list has label '0'. An example block is shown in Figure 4 with each vertex numbered according to the list. The edge connecting vertices 1 and 5 is curved to remind the reader that curved edges can be specified in blockMesh.

It is possible to generate blocks with less than 8 vertices by collapsing one or more pairs of vertices on top of each other.

Each block has a local coordinate system (x1,x2,x3) that must be right-handed. A right-handed set of axes is defined such that to an observer looking down the Oz axis, with O nearest them, the arc from a point on the Ox axis to a point on the Oy axis is in a clockwise sense.

The local coordinate system is defined by the order in which the vertices are presented in the block definition according to:

- the axis origin is the first entry in the block definition, vertex 0 in our example;
- the x1 direction is described by moving from vertex 0 to vertex 1;
- the x2 direction is described by moving from vertex 1 to vertex 2;
- vertices 0, 1, 2, 3 define the plane x3 = 0;
- vertex 4 is found by moving from vertex 0 in the x3 direction;
- vertices 5,6 and 7 are similarly found by moving in the x3 direction from vertices 1,2 and 3 respectively.

Keyword	**Description**	**Example/selection**
convertToMeters	Scaling factor for the vertex coordinates	0.001 scales to mm
vertices	List of vertex coordinates	(0 0 0)
edges	Used to describe arc or spline edges	arc 1 4 (0.939 0.342 -0.5)
block	Ordered list of vertex labels	hex (0 1 2 3 4 5 6 7)
patches	List of patches	symmetryPlane
mergePatchPairs	List of patches to be merged	-

Table 1: Keywords used in blockMeshDict

3.1 Writing a blockMeshDict file

The blockMeshDict file is a dictionary using keywords described below.

- **convertToMeters** scaling factor for the vertex coordinates, e.g. 0.001 scales to mm.
- **vertices** list of vertex coordinates
- **edges** used to describe curved geometry
- **block** ordered list of vertex labels and mesh size
- **boundary** sub-dictionary of boundary patches
- **mergePatchPairs** list of patches to be merged

The **convertToMeters** keyword specifies a scaling factor by which all vertex coordinates in the mesh description are multiplied. For example,

```
        convertToMeters 0.001;
```

means that all coordinates are multiplied by 0.001, i.e. the values quoted in the blockMeshDict file are in mm. The scaling is provided after using blockMesh generator.

3.1.1 Vertices

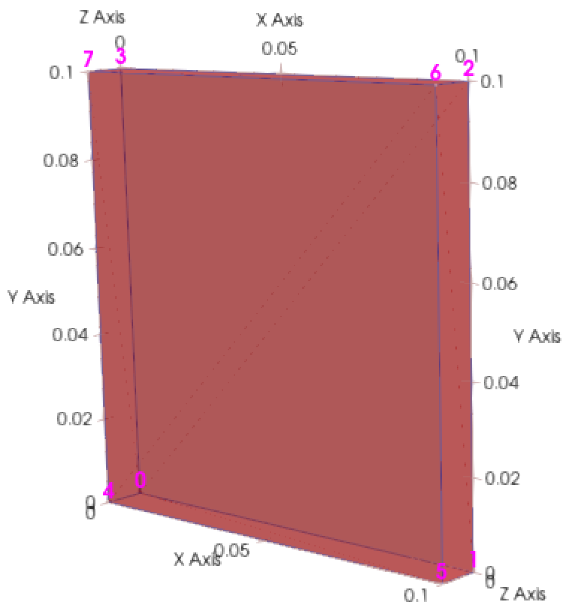


Figure 2: Block generated by blockMesh utility, example of vertices order

The vertices of the blocks of the mesh are given next as a standard list named vertices, e.g. for our example block in Figure 2, the vertices are:

```
vertices
(
( 0 0 0 )   // vertex number 0
( 1 0 0.1)   // vertex number 1
( 1.1 1 0.1)   // vertex number 2
( 0 1 0.1)   // vertex number 3
(-0.1 -0.1 1 )   // vertex number 4
( 1.3 0 1.2)   // vertex number 5
( 1.4 1.1 1.3)   // vertex number 6
( 0 1 1.1)   // vertex number 7
);
```

3.1.2 Edges

Each edge joining 2 vertex points is assumed to be straight by default. However any edge may be specified to be curved by entries in a list named edges. The list is optional; if the geometry contains no curved edges, it may be omitted.

Each entry for a curved edge begins with a keyword specifying the type of curve from those listed in Table.

Keyword selection	Description	Additional entries
arc	Circular arc	Single interpolation point
spline	Spline curve	List of interpolation points
polyLine	Set of lines	List of interpolation points
BSpline	B-spline curve	List of interpolation points
line	Straight line	-

Table 2: Edge types available in the blockMeshDict dictionary

The keyword is then followed by the labels of the 2 vertices that the edge connects. Following that, interpolation points must be specified through which the edge passes. For a arc, a single interpolation point is required, which the circular arc will intersect. For spline, polyLine and BSpline, a list of interpolation points is required. The line edge is directly equivalent to the option executed by default, and requires no interpolation points. Note that there is no need to use the line edge but it is included for completeness. For our example block in 3 we specify an arc edge connecting vertices 0 and 1 as follows through the interpolation point arc 0 1 (0.5 0.1 0.1):

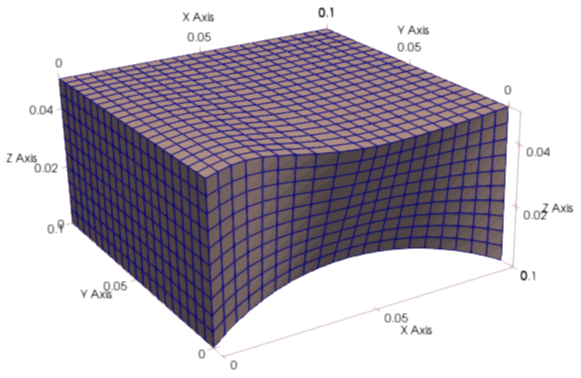


Figure 3: Example of arc and b-spline application on the box edges

```
edges
        (
        arc 0 1 (0.5 0.1 0.1)
        );
```

respectively use list of interpolation points for b-spline:

```
edges
        (
                bspline 1 5 ((1.1 0.0 0.5)(1.2 0.0 0.6)(1.3 0.0 0.7))
        );
```

3.1.3 Block

The block definitions are contained in a list named blocks. Each block definition is a compound entry consisting of a list of vertex labels whose order is described in previous subsections. A vector giving the number of cells required in each direction, the type and list of cell expansion ratio in each direction.

Then the blocks are defined as follows:

```
blocks
(
hex (0 1 2 3 4 5 6 7) // vertex numbers
(10 10 10) // numbers of cells in each direction
simpleGrading (1 2 3) // cell expansion ratios
);
```

The definition of each block is as follows:

Vertex numbering The first entry is the shape identifier of the block, as defined in the $FOAM_ETC-6cellModels file. The shape is always hex since the blocks are always hexahedral. There follows a list of vertex numbers, ordered in the manner described Figure 4.

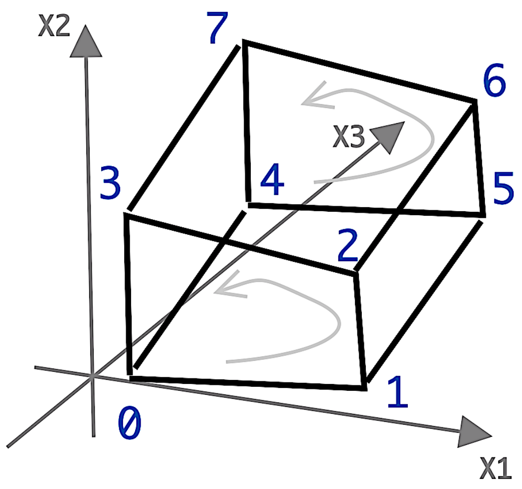


Figure 4: Hexahedral block, vertices ordering

Number of cells The second entry gives the number of cells in each of the x1 x2 and x3 directions for that block.

Cell expansion ratios The third entry gives the cell expansion ratios for each direction in the block. The expansion ratio enables the mesh to be graded, or refined, in specified directions. The ratio is that of the width of the end cell δ_e along one edge of a block to the width of the start cell δ_s along that edge, as shown in Figure 5. Each of the following keywords specify one of two types of grading specification available in blockMesh.

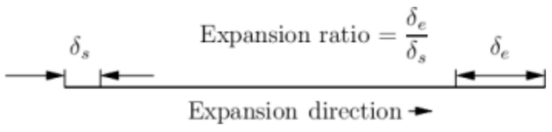


Figure 5: Mesh grading along a block edge (Openfoam user guide section 4.3)

simpleGrading The simple description specifies uniform expansions in the local x1, x2 and x3 directions respectively with only 3 expansion ratios, e.g. simpleGrading (1 2 3)

edgeGrading The full cell expansion description gives a ratio for each edge of the block, numbered according to the scheme shown in 6 with the arrows representing the direction from first cell to last cell e.g.
edgeGrading (1 1 1 1 2 2 2 2 3 3 3 3)
This means the ratio of cell widths along edges 0-3 is 1, along edges 4-7 is 2 and along 8-11 is 3 and is directly equivalent to the simpleGrading example given above.

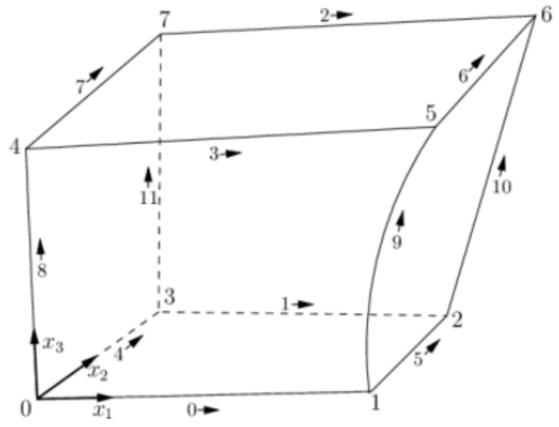


Figure 6: Single blockMesh box (Openfoam user guide section 4.3)

3.1.4 Multi-grading of a block

Using a single expansion ratio to describe mesh grading permits only ”one-way” grading within a mesh block. In some cases, it reduces complexity and effort to be able to control grading within separate divisions of a single block, rather than have to define several blocks with one grading per block. For example, to mesh a channel with two opposing walls and grade the mesh towards the walls requires three regions: two with grading to the wall with one in the middle without grading.

OpenFOAM v2.4+ includes multi-grading functionality that can divide a block in an given direction and apply different grading within each division. This multi-grading is specified by replacing any single value expansion ratio in the grading specification of the block, e.g. ”1”, ”2:, ”3” in

```
blocks
        (
                hex (0 1 2 3 4 5 6 7) (100 300 100)
                simpleGrading (1 2 3);
        );
```

We will present multi-grading for the following example:

split the block into 3 divisions in the y-direction, representing 20%, 60% and 20% of the block length; include 30% of the total cells in the y-direction (300) in each divisions 1 and 3 and the remaining 40% in division 2; apply 1:4 expansion in divisions 1 and 3, and zero expansion in division 2.

We can specify this by replacing the y-direction expansion ratio ”2” in the example above with the following:

```
blocks
        (
                hex (0 1 2 3 4 5 6 7) (100 300 100)
                simpleGrading
                (
                1  // x-direction expansion ratio
                (
                (0.2 0.3 4)  // 20% y-dir, 30% cells, expansion = 4
                (0.6 0.4 1)  // 60% y-dir, 40% cells, expansion = 1
                (0.2 0.3 0.25)  // 20% y-dir, 30% cells, expansion = 0.25
                )
                3  // z-direction expansion ratio
                )
        );
```

Both the fraction of the block and the fraction of the cells are normalized automatically. They can be specified as percentages, fractions, absolute lengths, etc. and do not need to sum to 100, 1, etc. The example above can be specified using percentages, e.g.

```
blocks
        (
                hex (0 1 2 3 4 5 6 7) (100 300 100)
                simpleGrading
                (
                        1
                (
                        (20 30 4) // 20%, 30%...
                        (60 40 1)
```

```
                    (20 30 0.25)
              )
                    3
              )
        );
```

3.1.5 Boundary

The boundary of the mesh is given in a list named boundary. The boundary is broken into patches (regions), where each patch in the list has its name as the keyword, which is the choice of the user, although we recommend something that conveniently identifies the patch, e.g. inlet; the name is used as an identifier for setting boundary conditions in the field data files. The patch information is then contained in sub-dictionary with:

- **type** the patch type, either a generic patch on which some boundary conditions are applied or a particular geometric condition, as listed bellow.
- **faces** a list of block faces that make up the patch and whose name is the choice of the user, although we recommend something that conveniently identifies the patch, e.g. inlet; the name is used as an identifier for setting boundary conditions in the field data files.

List of boundary conditions:

- basic types
  - fixedValue
  - fixedGradient
  - mixed
- geometrical constraints
  - symmetry
  - wedge
  - empty
  - cyclic
- derived, specialised conditions
  - fixedProfile: to specify a profile of a variable
  - swirlFlowRateInletVelocity: to specify velocity inlet for a swirling flow providing flow rate
  - inletOutlet: outlet condition with handling of reverse flow
  - codedFixedValue: fixed value set by user coding

blockMesh collects faces from any boundary patch that is omitted from the boundary list and assigns them to a default patch named defaultFaces of type empty. This means that for a 2 dimensional geometry, the user has the option to omit block faces lying in the 2D plane, knowing that they will be collected into an empty patch as required.

Returning to the example block in 6, if it has an inlet on the left face, an output on the right face and the four other faces are walls then the patches could be defined as follows:

```
boundary // keyword
        (
                inlet // patch name
                {
                        type patch; // patch type for patch 0
                        faces
                        (
                        (0 4 7 3) // block face in this patch
                        );
                } // end of 0th patch definition
                outlet // patch name
                {
                        type patch; // patch type for patch 1
                        faces
                        (
                        (1 2 6 5)
                        );
                }
                walls
                {
                type wall;
                faces
                (
                        (0 1 5 4)
                        (0 3 2 1)
                        (3 7 6 2)
                        (4 5 6 7)
                );
                }
        );
```

Each block face is defined by a list of 4 vertex numbers. The order in which the vertices are given must be such that, looking from inside the block and starting with any vertex, the face must be traversed in a clockwise direction to define the other vertices.

When specifying a cyclic patch in blockMesh, the user must specify the name of the related cyclic patch through the neighbourPatch keyword. For example, a pair of cyclic patches might be specified as follows:

```
        left
        {
                type cyclic;
                neighbourPatch right;
                faces ((0 4 7 3));
}
right
{
        type cyclic;
        neighbourPatch left;
        faces ((1 5 6 2));
}
```

3.1.6 Multiple blocks

A mesh can be created using more than 1 block. In such circumstances, the mesh is created as has been described in the preceding text; the only additional issue is the connection between blocks, in which there are two distinct possibilities:

- **face matching** the set of faces that comprise a patch from one block are formed from the same set of vertices as a set of faces patch that comprise a patch from another block;
- **face merging** a group of faces from a patch from one block are connected to another group of faces from a patch from another block, to create a new set of internal faces connecting the two blocks.

To connect two blocks with face matching, the two patches that form the connection should simply be ignored from the patches list. blockMesh then identifies that the faces do not form an external boundary and combines each collocated pair into a single internal faces that connects cells from the two blocks.

The alternative, face merging, requires that the block patches to be merged are first defined in the patches list. Each pair of patches whose faces are to be merged must then be included in an optional list named mergePatchPairs. The format of mergePatchPairs is:

```
        mergePatchPairs
        (
                ( <masterPatch> <slavePatch> ) // merge patch pair 0
                ( <masterPatch> <slavePatch> ) // merge patch pair 1
        ...
)
```

The pairs of patches are interpreted such that the first patch becomes the master and the second becomes the slave. The rules for merging are as follows:

- the faces of the master patch remain as originally defined, with all vertices in their original location;
- the faces of the slave patch are projected onto the master patch where there is some separation between slave and master patch;
- the location of any vertex of a slave face might be adjusted by blockMesh to eliminate any face edge that is shorter than a minimum tolerance;
- if patches overlap as shown in 7, each face that does not merge remains as an external face of the original patch, on which boundary conditions must then be applied;
- if all the faces of a patch are merged, then the patch itself will contain no faces and is removed.

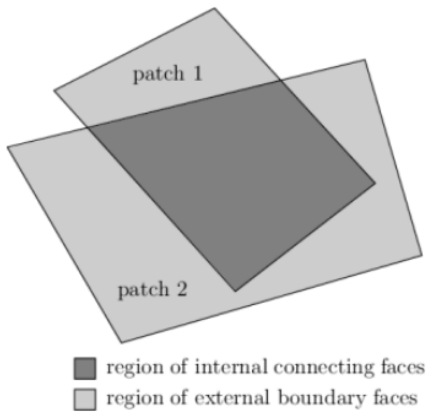

patch 1
patch 2
region of internal connecting faces
region of external boundary faces

Figure 7: Merging overlapping patches (Openfoam user guide section 4.3)

The consequence is that the original geometry of the slave patch will not necessarily be completely preserved during merging. Therefore in a case, say, where a cylindrical block is being connected to a larger block, it would be wise to the assign the master patch to the cylinder, so that its cylindrical shape is correctly preserved. There are some additional recommendations to ensure successful merge procedures:

- in 2 dimensional geometries, the size of the cells in the third dimension, i.e. out of the 2D plane, should be similar to the width/height of cells in the 2D plane;
- it is inadvisable to merge a patch twice, i.e. include it twice in mergePatchPairs;
- where a patch to be merged shares a common edge with another patch to be merged, both should be declared as a master patch.

3.2 Running blockMesh

Utility blockMesh can be run with specific options:

- **-blockTopology** Write block edges and centres as obj files and exit
- **-case directory** Specify case directory to use (instead of the cwd)
- **-dict file** Alternative dictionary for the blockMesh description
- **-noClean** Do not remove any existing polyMesh/ directory or files
- **-noFunctionObjects** Do not execute function objects
- **-region name** Specify alternative mesh region
- **-sets** Write cellZones as cellSets too (for processing purposes)
- **-time time** specify a time to write mesh to
- **-doc** Display documentation in browser
- **-doc-source** Display source code in browser
- **-help** Display short help and exit
- **-help-full** Display full help and exit

4 Mesh generator

4.1 CodeStream

There are two further directives that enable calculations from within input files: #calc, for simple calculations; #codeStream, for more complex calculations.

Present meshGenerator is based on codeStream openFoam class. This class is member of namespace Foam::functionEntries. Dictionary entry that contains C++ OpenFOAM code that is compiled to generate the entry itself. This techniques is available for blockeMesh, boundary and initial condition, controll postprocessing and so one.

CodeStream reads three entries: "code", "codeInclude", which is optional, "codeOptions" and uses those to generate library sources inside codeStream. These get compiled using 'wmake libso' the resulting library is loaded in executed with as arguments: `(const dictionary\& dict, Ostream\& os)` , where the dictionary is the current dictionary. The code has to write into Ostream which is then used to construct the actual dictionary entry.

In our case, we are using codeStream mainly for blockMesh generation. Another application is to set the internal field, see following example:

```
internalField #codeStream
{
        code
        #{
        const IOdictionary& d = static_cast<const IOdictionary&>(dict);
        const fvMesh& mesh = refCast<const fvMesh>(d.db());
        scalarField fld(mesh.nCells(), 12.34);
        fld.writeEntry("", os);
        #};

        codeInclude
        #{
                #include "fvCFD.H"
                #};

        //! Optional:
        codeOptions
        #{
                -I$(LIB_SRC)/finiteVolume/lnInclude
                #};
}
```

Note, the `#{ ... #}` syntax is a 'verbatim' input mode that allows inputting strings with embedded newlines.

There are some important limitations for coding inside codeStream: ∼ symbol not allowed inside the code sections and probably some other limitations (uses string::expand which expands $ and ∼ sequences). The code to be compiled is stored under the local codeStream directory with a subdirectory name corresponding to the SHA1 of the contents. The corresponding library code is located under the local

`codeStream/platforms/$WM_OPTIONS/lib directory`

in a library libcodeStream_SHA1.so. Source files are codeStream.H and codeStream.C. We can find Definition at line 116 of file codeStream.H.

`https://www.openfoam.com/documentation/guides/latest/api/codeStream_8H_source.html#l00116`

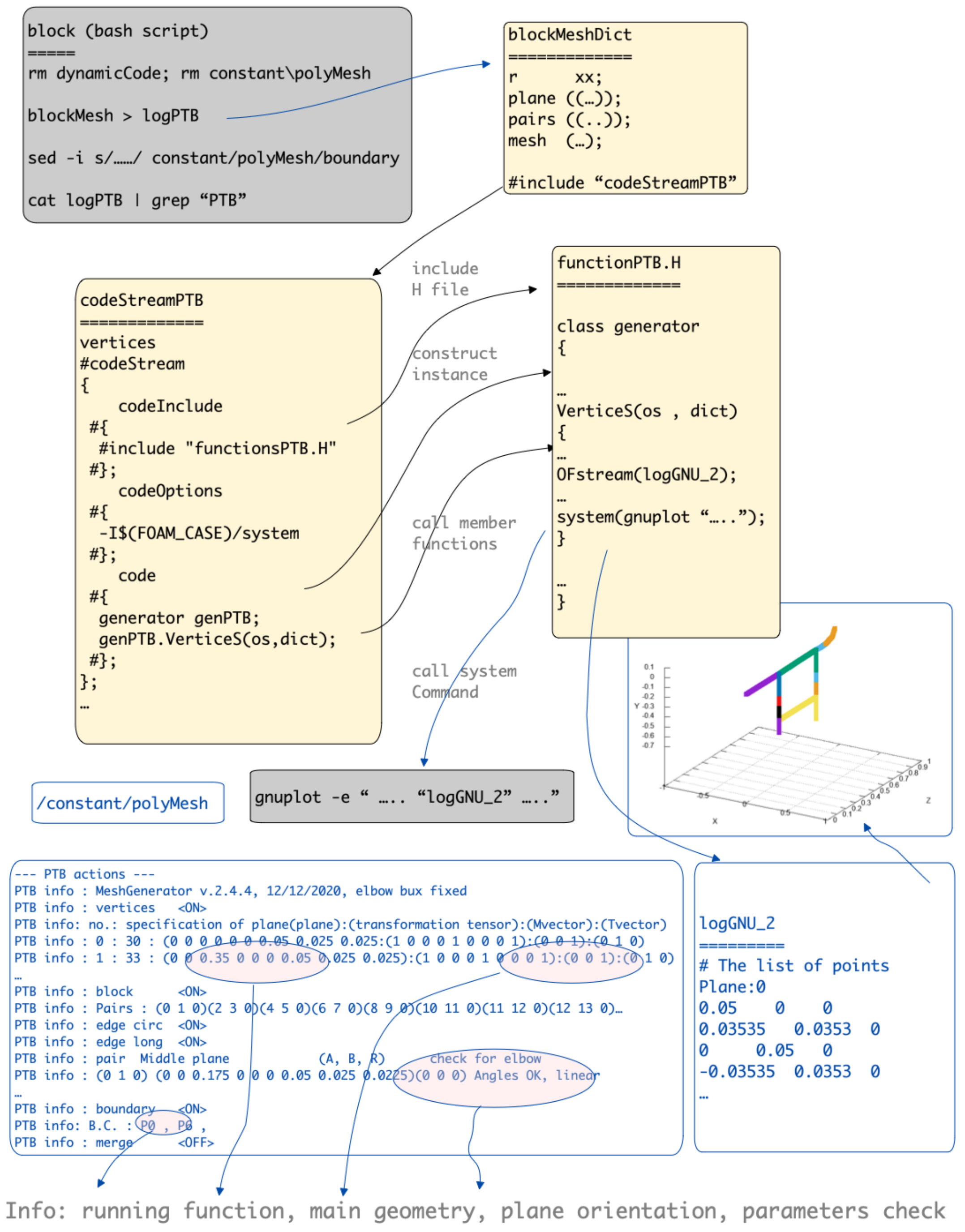
4.2 Workflow

The typical workflow of Mesh generation is apparent on Figure 8

The workflow is consist of following steps:

- **run alias**, optional step, help to filter log-files .
- **run blockMesh**, call directly on working directory, or use parameter -case, or through the script of alias.

- **include codeStreamPTB**, the file included by blockMeshDict, create the requested blockMeshDict structure.
- **call suitable member functions**, write on file functionPTB.H, called through the file codeStreamPTB.



include
H file
construct
instance
call member
functions
call system
Command
Y
X
Z
Info: running function, main geometry, plane orientation, parameters check

```
block (bash script)
=====
rm dynamicCode; rm constant\polyMesh

blockMesh > logPTB

sed -i s/……/ constant/polyMesh/boundary

cat logPTB | grep “PTB”
```

```
blockMeshDict
=============
r      xx;
plane ((…));
pairs ((..));
mesh  (…);

#include “codeStreamPTB”
```

```
codeStreamPTB
=============
vertices
#codeStream
{
    codeInclude
 #{
  #include "functionsPTB.H"
 #};
    codeOptions
 #{
  -I$(FOAM_CASE)/system
 #};
    code
 #{
  generator genPTB;
  genPTB.VerticeS(os,dict);
 #};
};
…
```

```
functionPTB.H
=============

class generator
{

…
VerticeS(os , dict)
{
…
OFstream(logGNU_2);
…
system(gnuplot “…..”);
}

…
}
```

```
/constant/polyMesh
```

```
gnuplot -e “ ….. “logGNU_2” …..”
```

```
--- PTB actions ---
PTB info : MeshGenerator v.2.4.4, 12/12/2020, elbow bux fixed
PTB info : vertices   <ON>
PTB info: no.: specification of plane(plane):(transformation tensor):(Mvector):(Tvector)
PTB info : 0 : 30 : (0 0 0 0 0 0 0.05 0.025 0.025:(1 0 0 0 1 0 0 0 1):(0 0 1):(0 1 0)
PTB info : 1 : 33 : (0 0 0.35 0 0 0 0.05 0.025 0.025):(1 0 0 0 1 0 0 0 1):(0 0 1):(0 1 0)
…
PTB info : block      <ON>
PTB info : Pairs : (0 1 0)(2 3 0)(4 5 0)(6 7 0)(8 9 0)(10 11 0)(11 12 0)(12 13 0)…
PTB info : edge circ  <ON>
PTB info : edge long  <ON>
PTB info : pair  Middle plane            (A, B, R)      check for elbow
PTB info : (0 1 0) (0 0 0.175 0 0 0 0.05 0.025 0.025)(0 0 0) Angles OK, linear
…
PTB info : boundary   <ON>
PTB info: B.C. : P0 , P6 ,
PTB info : merge      <OFF>
```

```
logGNU_2
=========
# The list of points
Plane:0
0.05      0      0
0.03535   0.0353  0
0       0.05    0
-0.03535  0.0353  0
…
```

Figure 8: Workflow example

It is mentioned three important files: blockMeshDict, codeStreamPTB and functionPTB.H, see Figure 9, located on system directory of working openFoam directory. **All the code are decompose to those three file, blockMeshDict include just user define variables and MeshGenerator variables, and code itself written to files codeStreamPTB and functionPTB.H, forbidden modify by user.** Files relationship is apparent from scheme on 9.

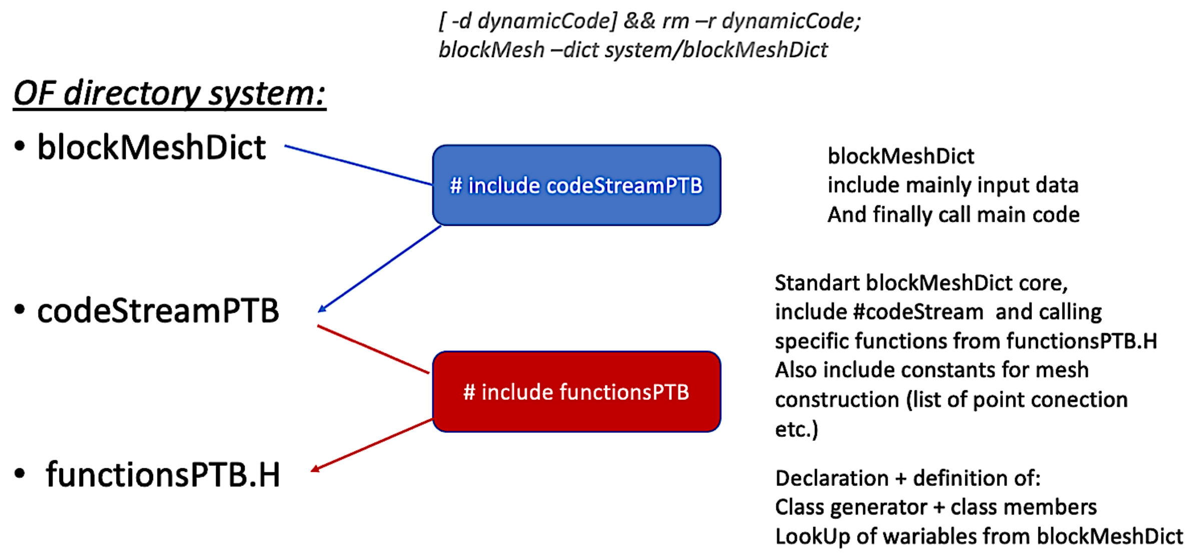

[-d dynamicCode] && rm –r dynamicCode;
blockMesh –dict system/blockMeshDict
OF directory system:
• blockMeshDict
• codeStreamPTB
• functionsPTB.H
# include codeStreamPTB
# include functionsPTB
blockMeshDict
include mainly input data
And finally call main code
Standart blockMeshDict core,
include #codeStream and calling
specific functions from functionsPTB.H
Also include constants for mesh
construction (list of point conection
etc.)
Declaration + definition of:
Class generator + class members
LookUp of wariables from blockMeshDict

Figure 9: MeshGenerator files relationship

The file **blockMeshDict** include user variables and MeshGenerator variables for mesh generation. This file is edit by user. The file **codeStreamPTB** include the code for creating right blockMeshDict structure and it is forbidden to modify by user. The file **functionsPTB** include the code of generator class and it is also forbidden to modify by user.

4.3 Key variables

To control the mesh and geometry creation, we need list of input parameters to describe required pipeline system. The list of all parameters including types:

- **scale**, scalar, OpenFoam optional variable, scale of model. It is used after generator.
- **radiuS**, scalar, default pipeline radius. Obligatory.
- **planes**, tensorField, include transformation of specific planes related to base circle on origin and plane x-y.
- **defaultSystem**, scalar, optional, default is 0, means relative coordinate system.

- **planeSpec**, 2D vectorField, obligatory even if it is empty , specify local non-planar planes, specially for T-junction
- **pairs**, vectorField, obligatory, include information about connection of specific pairs of planes
- **freePlanes**, scalarField, obligatory even if it is empty, include the list of planes for BC specification.
- **mesh**, 4 position scalarField, obligatory, include mesh size
- **switcher**, 6 position scalarField, optional, switcher for blockMeshDict blocks. Allow switch of some some part for code testing.

Example of using all parameters, see follow blockMeshdict:

```
//(OF header)
...
FoamFile
\{
version 2.0;
format ascii;
class dictionary;
object blockMeshDict;
\}
// * * * * * * * * * * * * * * * * * * * * * * * * ** * * //

// --------- User define variables ------

PI 3.14159265358979323846;
scale 1;
radiuS 1;

planes  (
        ( 0 0 0 0 0 0 1 0.5 0.45 ) // PL-00
        ( 0 0 1 0 0 0 1.5 0.5 0.45 ) // PL-01
               );

defaultSystem 0;

planeSpec (
                      (1 0)
                      (2 0)
                      );

pairs (
                (0 1 0 )
                );

freePlanes (0 1);

mesh (5 5 20 1);

// ====== switch =====

switcher ( 1 // vortices
1 // blocks
1 // edges circumferential
1 // edges longitudial
1 // boundary
0 // mergePatchPairs
); // default (1 1 1 1 1 1)

// ---- codeStream for pipeline Mesh generator, v.2.5, 31.12.2020 ---
\#include "codeStreamPTB";
```

The code generate simple pipe with begin diameter 1 and final diameter 1.5, see previous definition on planes variable and Figure 10.

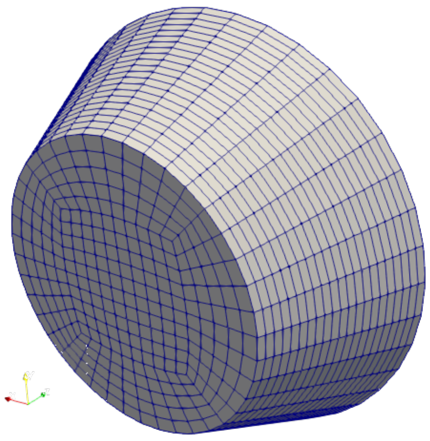


Figure 10: Pile construction

4.4 Procedure recommendation

The first step fur user is decomposition of complex geometry several parts: linear pipe, pipe with variable diameter, such as Venturi pipe, elbow and T-junction. Next step is definition of characteristic planes (variable planes) an finally define pairs of plane which form a hole pipeline system. For last step is also necessary to define plane specification (variable planeSpec), especially for non-planar planes on T-junctions.

4.4.1 Linear pipe

For linear pipeline , see Figure 11, is just necessary to define specific planes and pairs, see follow related code:

```
...
radiuS 1;

planes (
        ( 0 0 0 0 0 0 1 0.5 0.45 ) // PL-00
        ( 0 0 1 0 0 0 1.5 0.5 0.45 ) // PL-01
        ( 0 0 2 0 0 0 0.9 0.5 0.45 ) // PL-02
        ( 0 0 1.5 0 0 0 0.9 0.5 0.45 ) // PL-03
);

planeSpec ( );

pairs (
(0 1 0 )
(1 2 0 )
(2 3 0 )
);
...
```

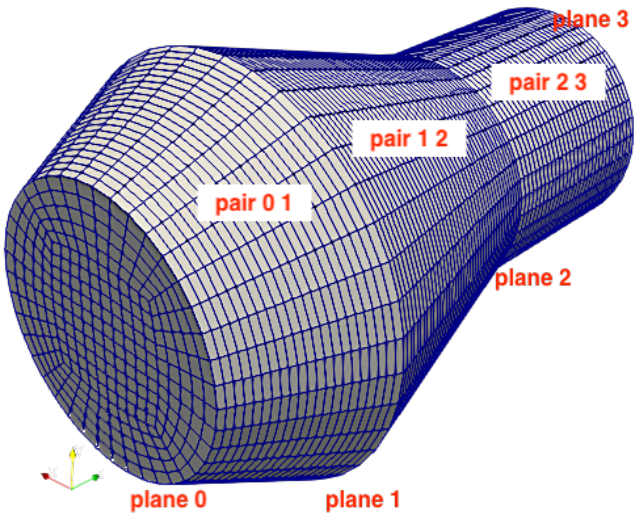


Figure 11: Linear pile construction

4.4.2 Elbow

Elbow part is created, similarly as linear part, between two specific planes. The system calculate angles between normal of planes $N[n]$ and vector of $M[m]$, see Figure 12, and if both angles are non zero, system calculate middle plane (internal variable planesC) and create arc, see Figure 12.

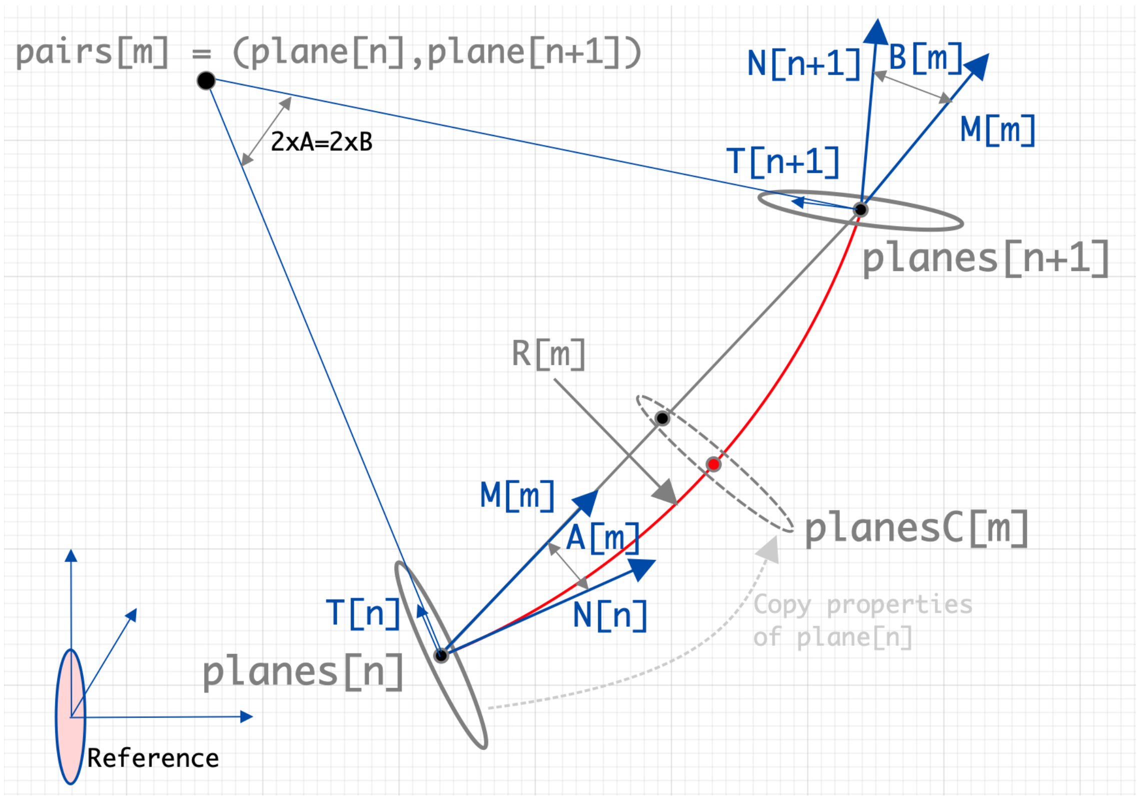


Figure 12: Elbow construction

In the case of non zero angles, the system calculate a middle plane of arc, see example on Figure 13, and use command blockMesh ”edge” for specific pair. Note: on the last version v.2.5 from 31.12.2020, both angles must be non-zeros.

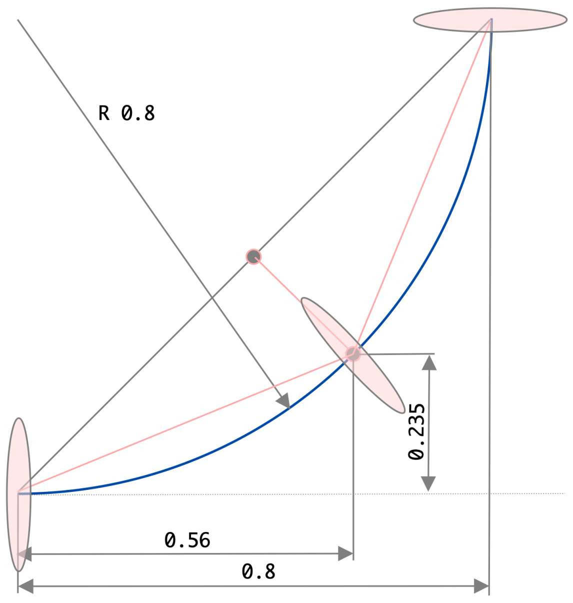


Figure 13: Middle point is used for elbow construction

To correctly define orientation of planes is necessary to understand coordinates system and related transformation. Example of vector orientation is obvious from Figure 14. There are two different results for second elbow if you are using rotation vector (1 0 0) of (0 0 0) for final plane.

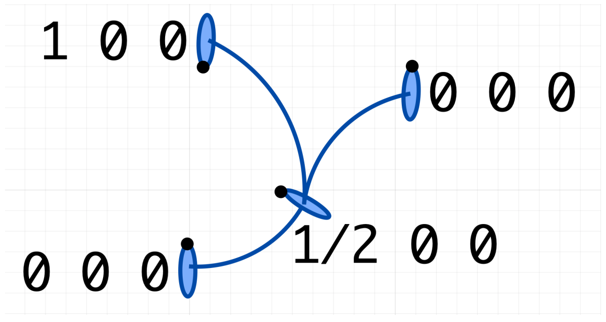
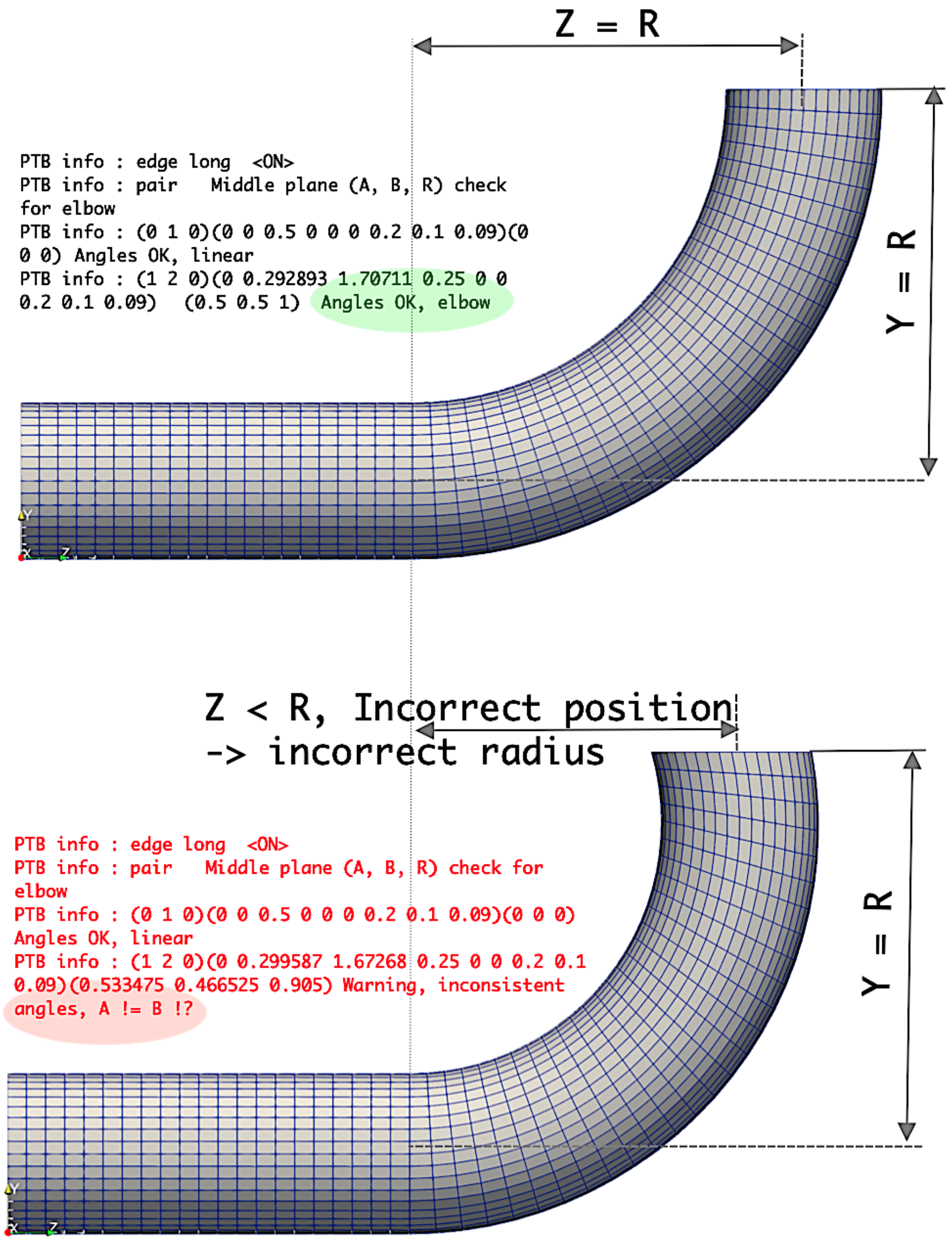


Figure 14: Elbow, rotation vector example

The final plane of elbow bust be correctly calculated by user. Incorrect position definition allow elbow creation, but geometry is not right, see Figure 15. The log file also include calculated angles and warning of inconsistent angles for specific plane pair. Regardless angle inconsistency, the system try to create arc between planes. The normal vector of first plane is correct and final plane normal is incorrect, please compare both results on figure.



```
PTB info : edge long  <ON>
PTB info : pair   Middle plane (A, B, R) check
for elbow
PTB info : (0 1 0)(0 0 0.5 0 0 0 0.2 0.1 0.09)(0
0 0) Angles OK, linear
PTB info : (1 2 0)(0 0.292893 1.70711 0.25 0 0
0.2 0.1 0.09)  (0.5 0.5 1)  Angles OK, elbow
```

```
PTB info : edge long  <ON>
PTB info : pair   Middle plane (A, B, R) check for
elbow
PTB info : (0 1 0)(0 0 0.5 0 0 0 0.2 0.1 0.09)(0 0 0)
Angles OK, linear
PTB info : (1 2 0)(0 0.299587 1.67268 0.25 0 0 0.2 0.1
0.09)(0.533475 0.466525 0.905) Warning, inconsistent
angles, A != B !?
```

Figure 15: Elbow, correct and incorrect final plane position

Figure 16 shows us the examples for final plane rotation vector for elbow on Y-Z and X-Z plane. It is obvious that the same result can be obtain with different rotation vector. For example, final (green) plane on x-z plane is describe by vector (0.5 0.5 0.5). The same results of plane rotating is coming with rotating vector (0 0.5 0).

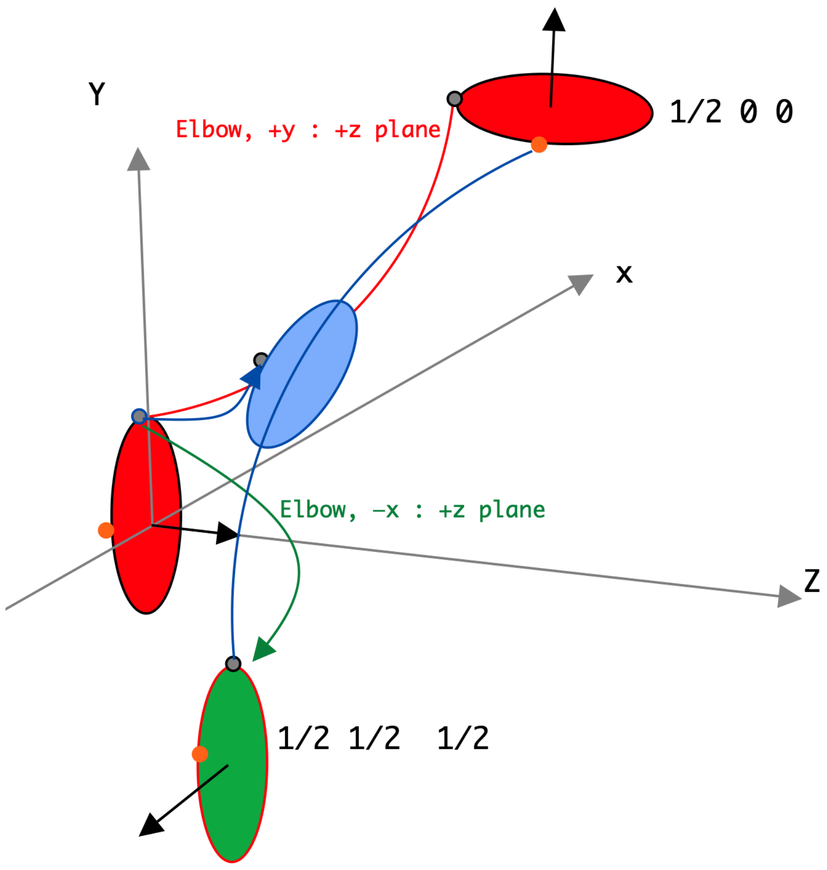


Figure 16: Example of Top point position and rotor vector for planes x-z and y-z

We recommend to construct elbow max 90 degrees. The bigger angle is possible, but not recommend due to collapse of surfaces, see Figure 17.

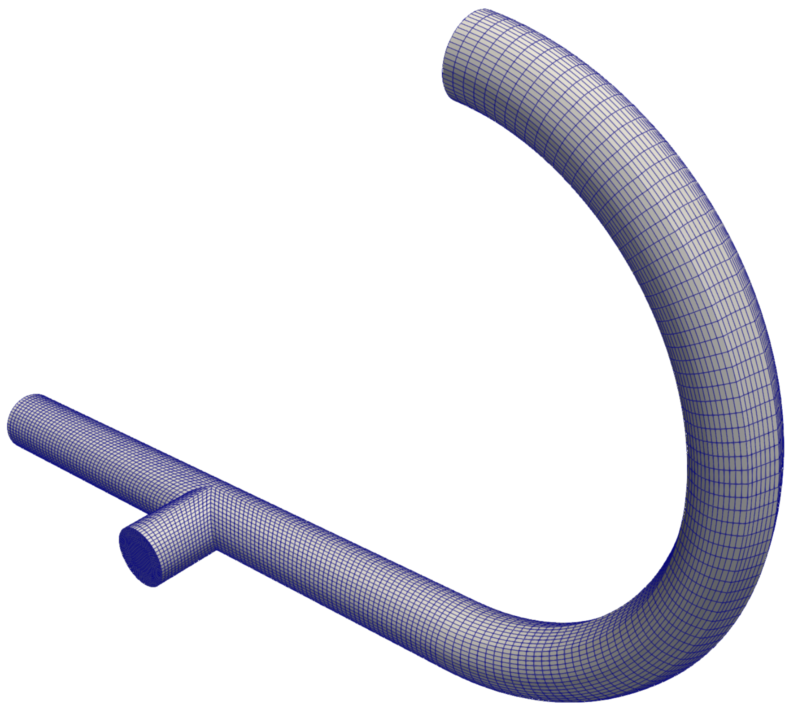


Figure 17: Elbow 180 dgr. Not recommended due to edge on surface

To define the correct plane is probably the most complicated for elbow construction. For right rotation vector definition, we recommend to think about actual orientation of tangential (T) and normal (N) vector, see Figure 18. The origin orientation are :T(0 1 0) and N(0 0 1). The figure shows us to elbows on two different planes, fist one on y-z, second one on x-y. The effect of double rotation on T and N vectors are obvious.

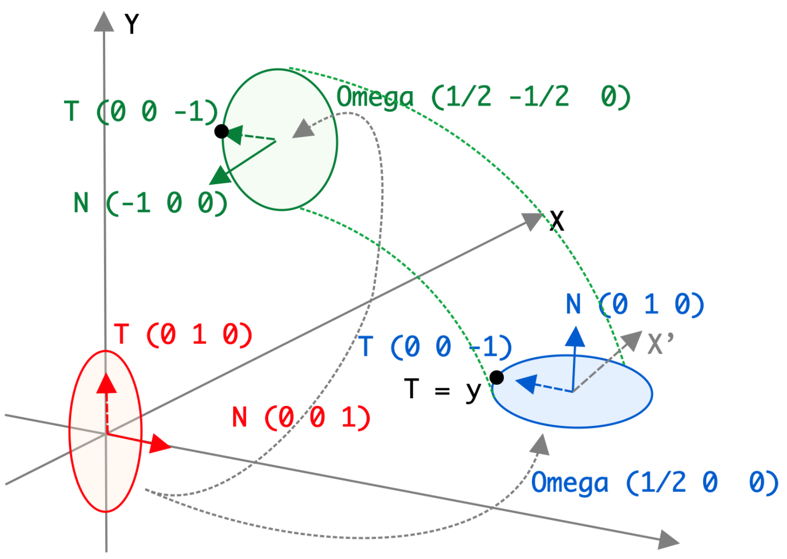
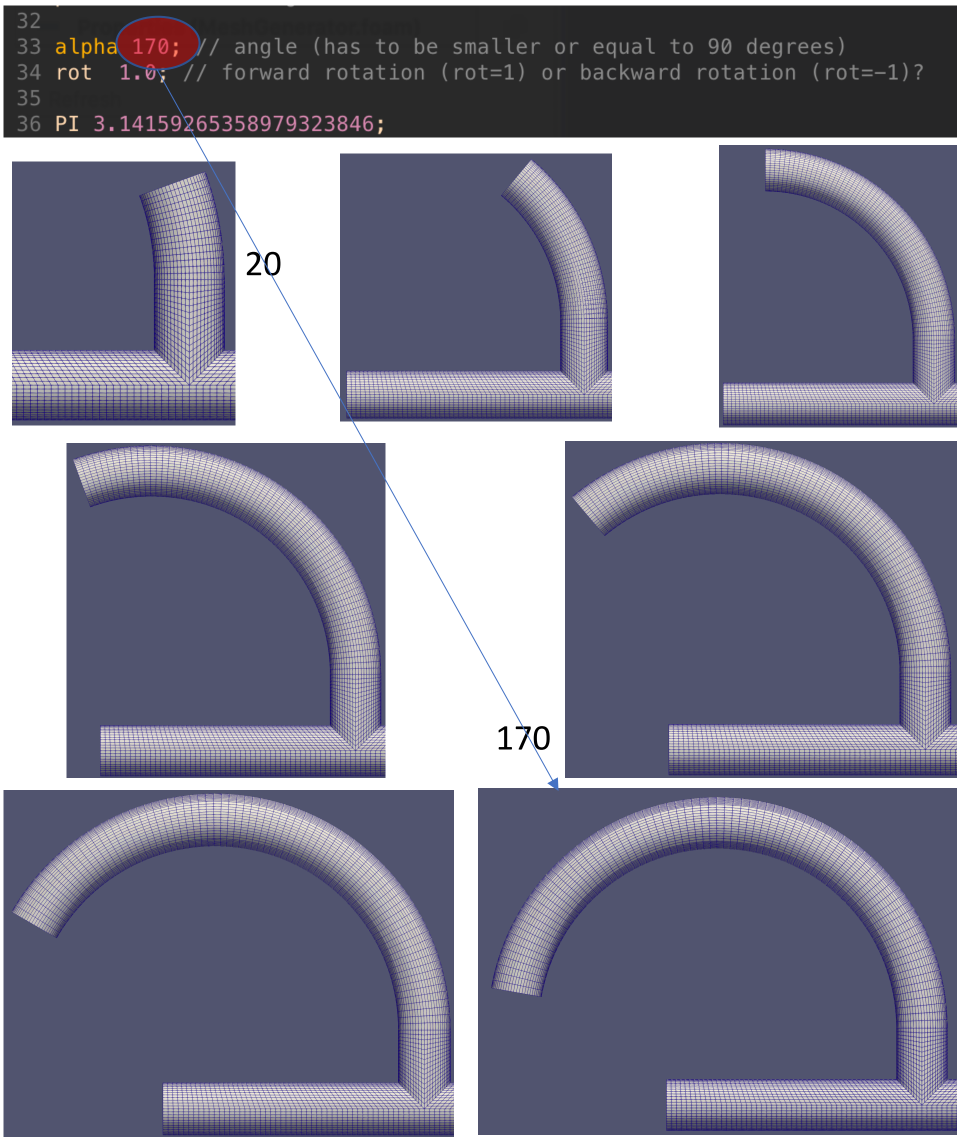


Figure 18: Couple of elbows on different planes

It is good practice to define user variable to be able simply modify angle of elbow on your model. On the Figure 19 is apparent effect of input parameter ALPHA of final geometry. The example show elbow from 20 to 170 degree, but it is strongly recommend define elbow just up-to 90 degree. The variable plane include sinus and cosine function of alpha variable define before variable plane.



```
alpha 170; // angle (has to be smaller or equal to 90 degrees)
rot  1.0; // forward rotation (rot=1) or backward rotation (rot=-1)?

PI 3.14159265358979323846;
```

Figure 19: Elbow construction, angle input,

4.4.3 T-junction

The most complicated part of pipeline construction is T-junction. The complication is on the connection. There is no anymore simple pair of two planes such as linear or elbow construction. **The strategy is create tree (respectively 4) planes for different pipeline directions. Then, system automatically remove duplicity of the points.**

The T-junction connection required non-planar plane definition. We are using variable **planeSpec** for non planar definition. PlaneSpec is field of 2D vectors. First position is number of plane (equivalent of plane position in plane field), second is plane specification. The specification parameter inform system about direction and orientation of non-planar part. All possible combinations are apparent from Figure 20.

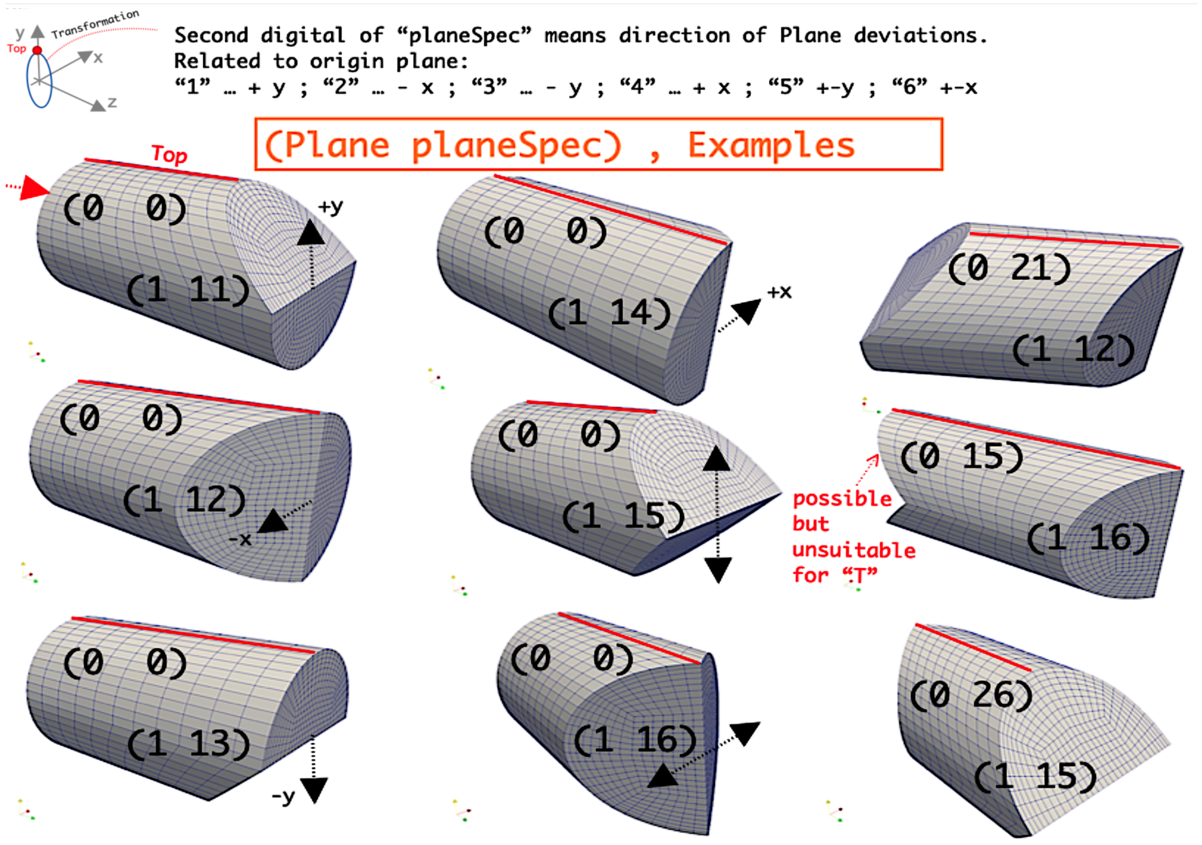

y
Top
Transformation
x
z
Second digital of "planeSpec" means direction of Plane deviations.
Related to origin plane:
"1" … + y ; "2" … - x ; "3" … - y ; "4" … + x ; "5" +-y ; "6" +-x
(Plane planeSpec) , Examples
Top
(0 0)
(1 11)
+y
(0 0)
(1 14)
+x
(0 21)
(1 12)
(0 0)
(1 12)
-x
(0 0)
(1 15)
possible but unsuitable for "T"
(0 15)
(1 16)
(0 0)
(1 13)
-y
(0 0)
(1 16)
(0 26)
(1 15)

Figure 20: T junction strategy, list of planes

Parameter equal 11,12,13 and 14 is necessary for T-junction, depend on direction +y,-x,-y,+x. Parameter value 15 and 16 is necessary for +-junction and minor branch of T-junction, please see Figure 21. Figure 22 shows three examples of T and + -junction on plane +y +z, respectively -y +z.

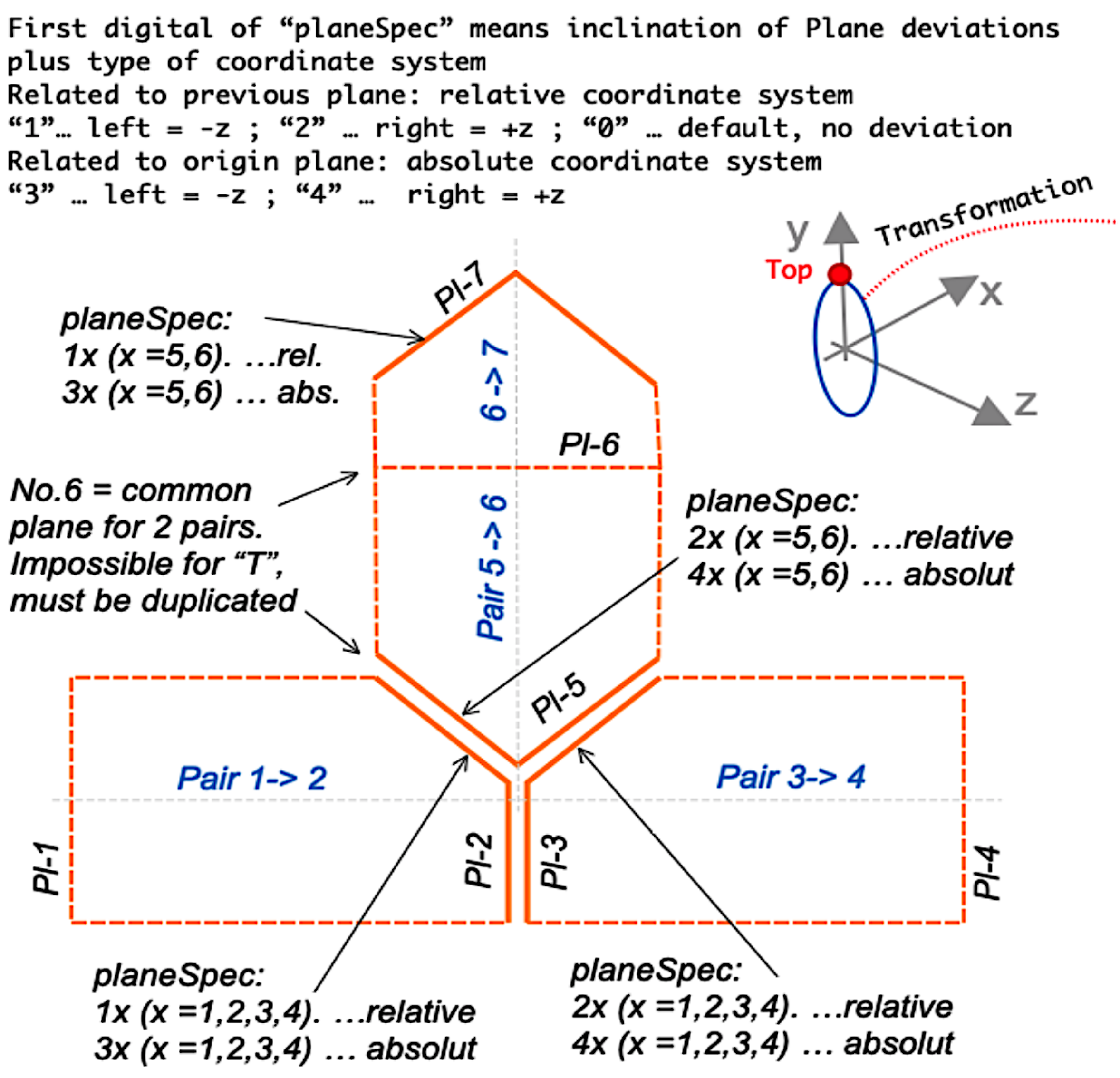

First digital of “planeSpec” means inclination of Plane deviations plus type of coordinate system
Related to previous plane: relative coordinate system
“1”… left = -z ; “2” … right = +z ; “0” … default, no deviation
Related to origin plane: absolute coordinate system
“3” … left = -z ; “4” … right = +z

Figure 21: T junction strategy, list of planes

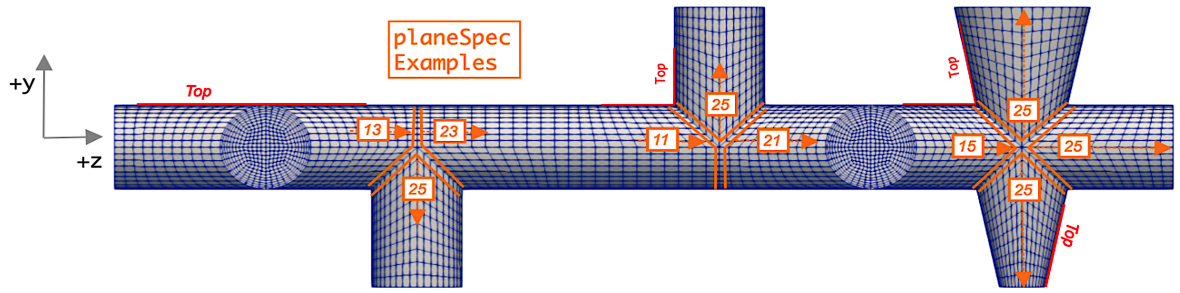


Figure 22: T junction strategy, list of planes

The following Figures 23, 24 and 25 show us examples of geometry including all possible junctions with all possible directions.

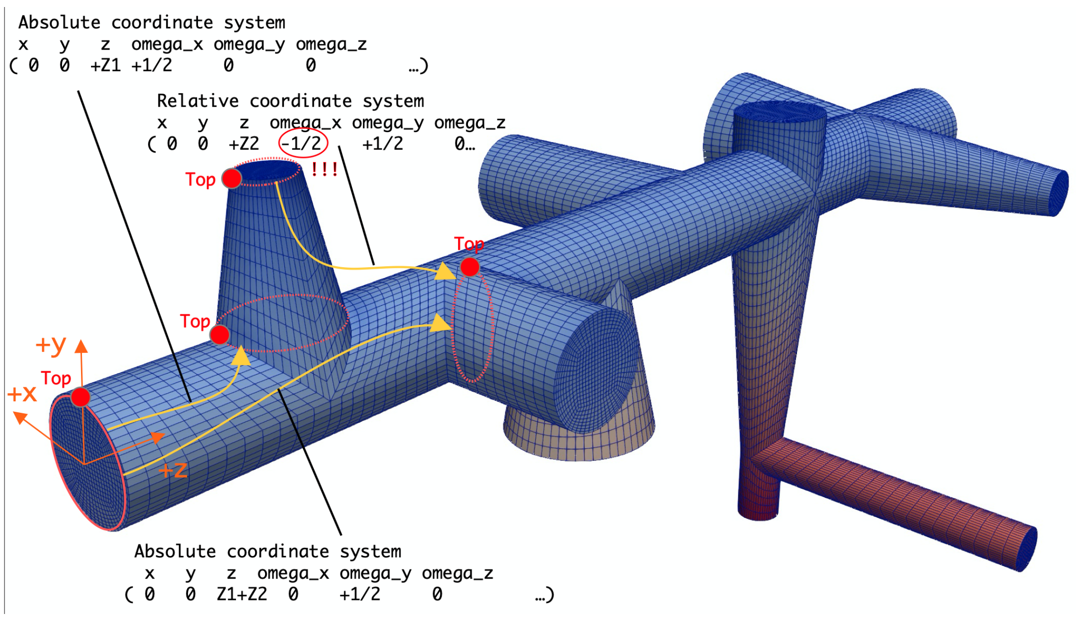

Absolute coordinate system
x y z omega_x omega_y omega_z
(0 0 +Z1 +1/2 0 0 …)
Relative coordinate system
x y z omega_x omega_y omega_z
(0 0 +Z2 -1/2 +1/2 0…
!!!
Top
Top
Top
Top
+y
+x
+z
Absolute coordinate system
x y z omega_x omega_y omega_z
(0 0 Z1+Z2 0 +1/2 0 …)

Figure 23: T junction types, T and plus

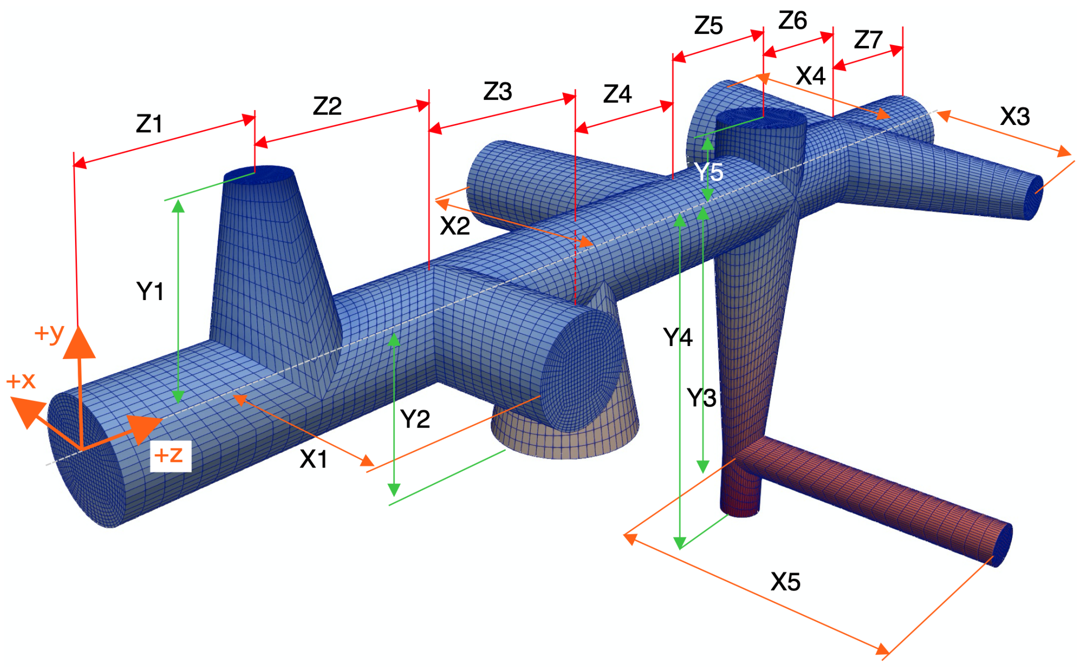


Figure 24: Example of used parameters

Main parameters, suplementary parameters

```
pZ1 1;
pZ2 1;
pZ3 1;
pZ4 1;
pZ5 1;
pZ6 1;
pZ7 1;
```

Positive Z-direction

```
pY1 1;
pY2 1;
pY3 2;
pY4 2.5;
pY5 0.5;
```

Positive Y-direction

```
pX1 1;
pX2 1.5;
pX3 1;
pX4 1.5;
pX5 2;
```

Positive X-direction

```
R1  1.0;
R1s 0.5; // smoller R
R1b 1.5; // bigger  R
R2 0.50;
R3 0.45;
```

Set of radius

Negative directions

```
nZ17 #calc " -$pZ1-$pZ2-$pZ3-$pZ4-$pZ5-$pZ6-$pZ7 ";
pZ15 #calc "  $pZ1+$pZ2+$pZ3+$pZ4+$pZ5 ";

nZ1  #calc " -$pZ1";
nZ2  #calc " -$pZ2";
nZ3  #calc " -$pZ3";
nZ4  #calc " -$pZ4";
nZ5  #calc " -$pZ5";
nZ6  #calc " -$pZ6";
nZ7  #calc " -$pZ7";

nY1  #calc " -$pY1";
nY2  #calc " -$pY2";
nY3  #calc " -$pY3";
nY4  #calc " -$pY4";
nY5  #calc " -$pY5";

nX1  #calc " -$pX1";
nX2  #calc " -$pX2";
nX3  #calc " -$pX3";
nX4  #calc " -$pX4";
nX5  #calc " -$pX5";

nA  #calc " -$pA";
p2A #calc " 2*$pA";
```

Anges

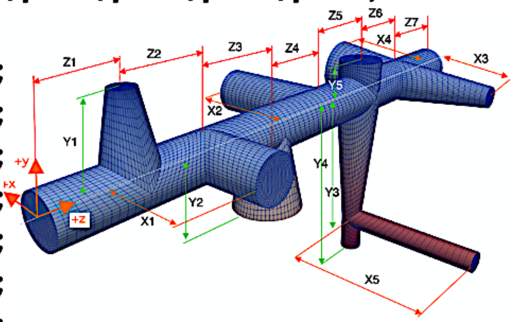
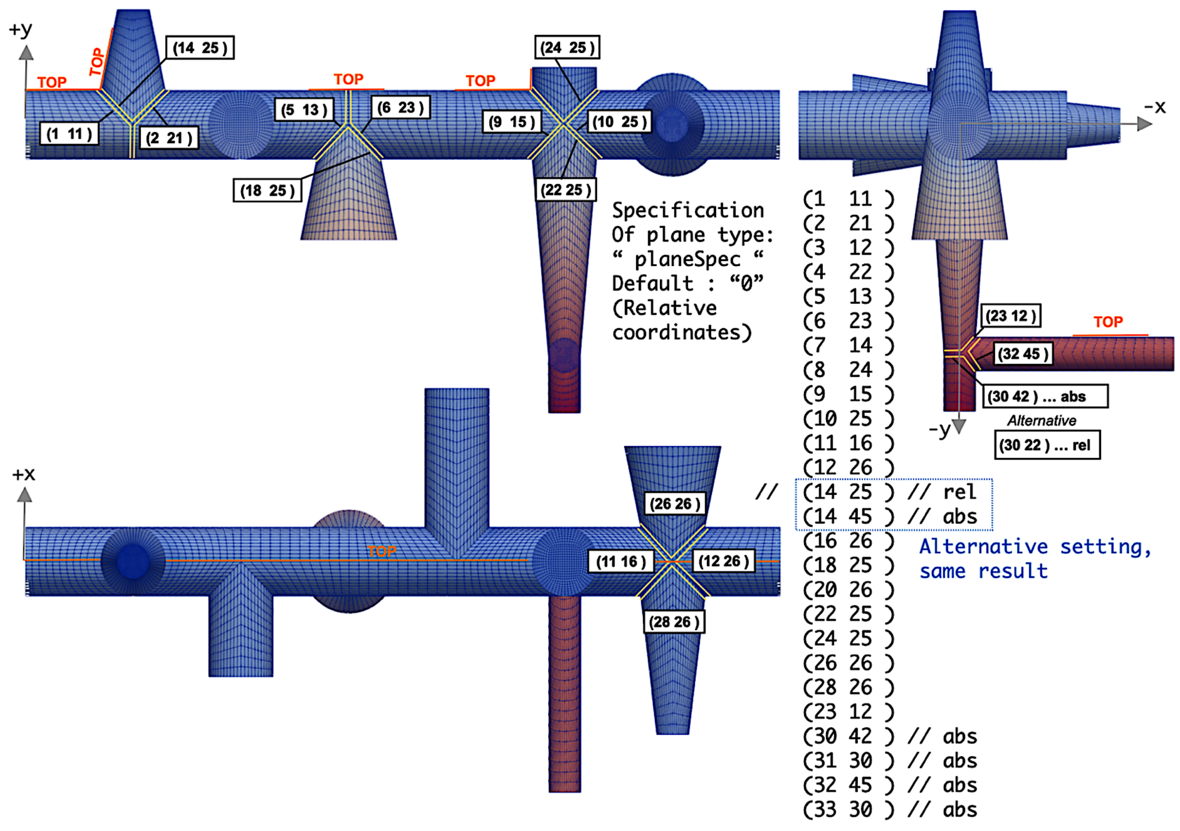


Figure 25: Main geometrical parameters



```
Specification
Of plane type:
" planeSpec "
Default : "0"
(Relative
coordinates)
```

```
(1  11 )
(2  21 )
(3  12 )
(4  22 )
(5  13 )
(6  23 )
(7  14 )
(8  24 )
(9  15 )
(10 25 )
(11 16 )
(12 26 )
(14 25 ) // rel
(14 45 ) // abs
(16 26 )
(18 25 )
(20 26 )
(22 25 )
(24 25 )
(26 26 )
(28 26 )
(23 12 )
(30 42 ) // abs
(31 30 ) // abs
(32 45 ) // abs
(33 30 ) // abs
```

Figure 26: List specification for T junction construction

4.4.4 Mesh definition

Density and distribution of mesh is define though variable **mesh**. Examples of parameters is obvious from Figure 27. The meaning of variable parameters are:

- **number of inner core cells** , radial cell information on pipeline core
- **number of outer core cells** , radial cell information on boundary layer area
- **number of cells per unit** , longitudinal cell information
- **expansion ratio**, $\frac{\Delta_{lastcell}}{\Delta_{firstcell}}$

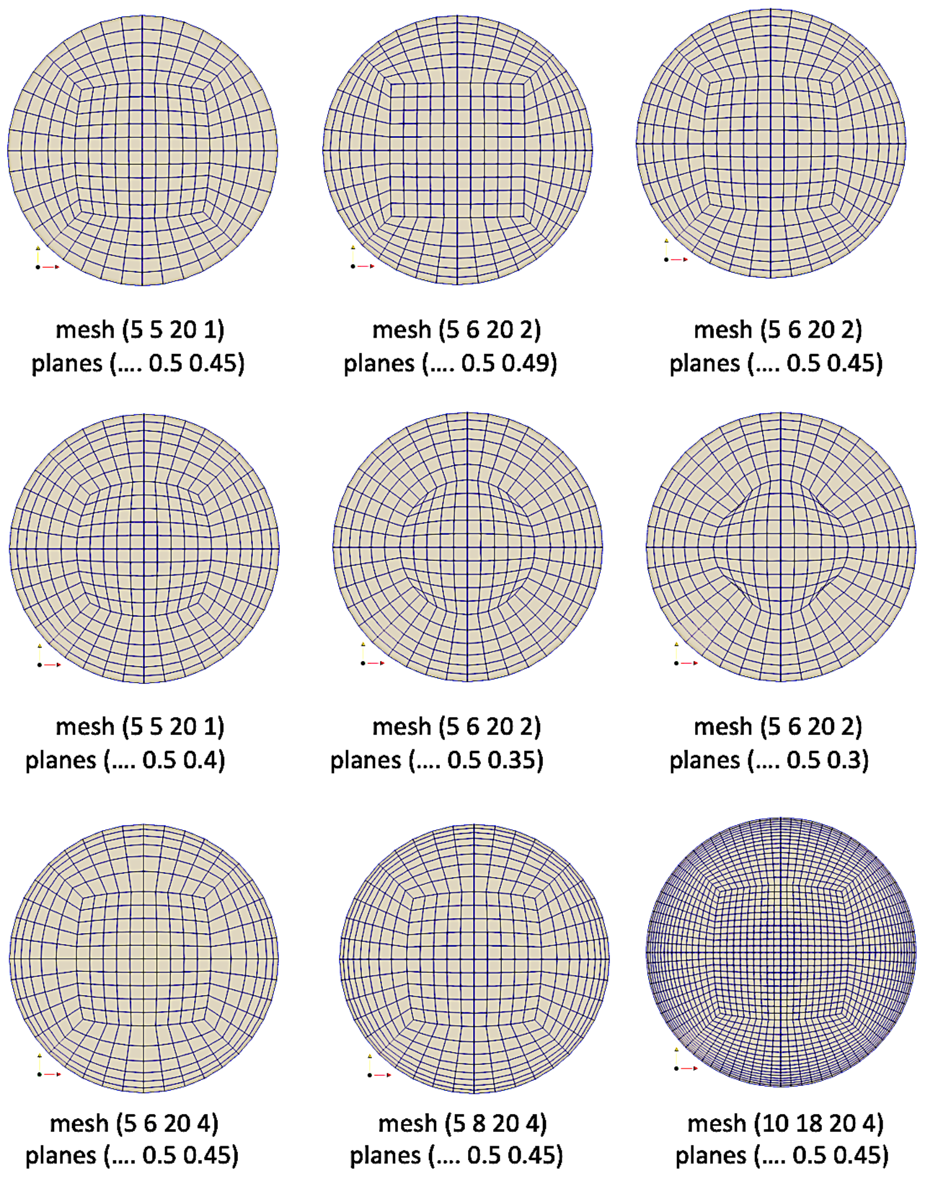


Figure 27: Parameters of variable mesh

4.5 Coordinate system

4.5.1 Plane rotation

Variable **planes** include 9 parameters. First three is translation x,y,z, according actual coordinate system, next three position are rotation related to coordinates axis. The next three Figures 28, 29 and 30 describe rotation around X, Y and Z. **The unit of rotation is PI multiplication, for example, angle 90 degree is: 0.5 etc.**

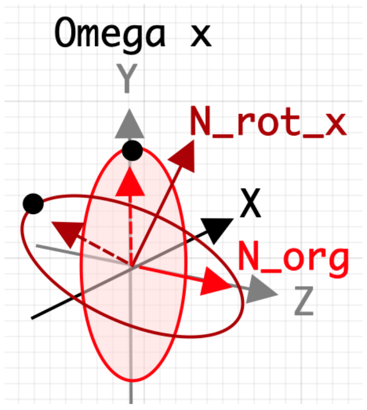


Figure 28: Plane rotation around axis X, ω_x

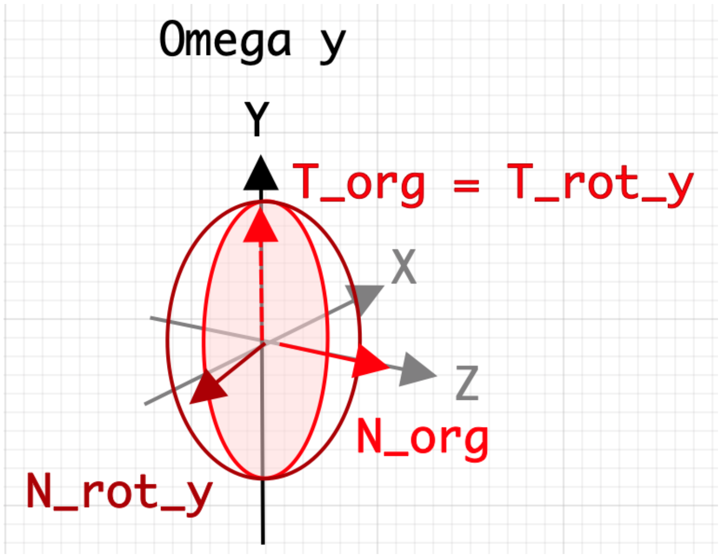


Figure 29: Plane rotation around axis Y, ω_y

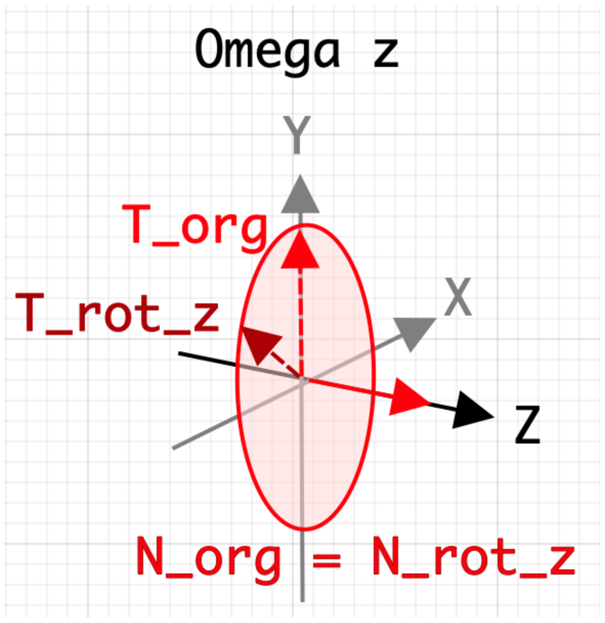
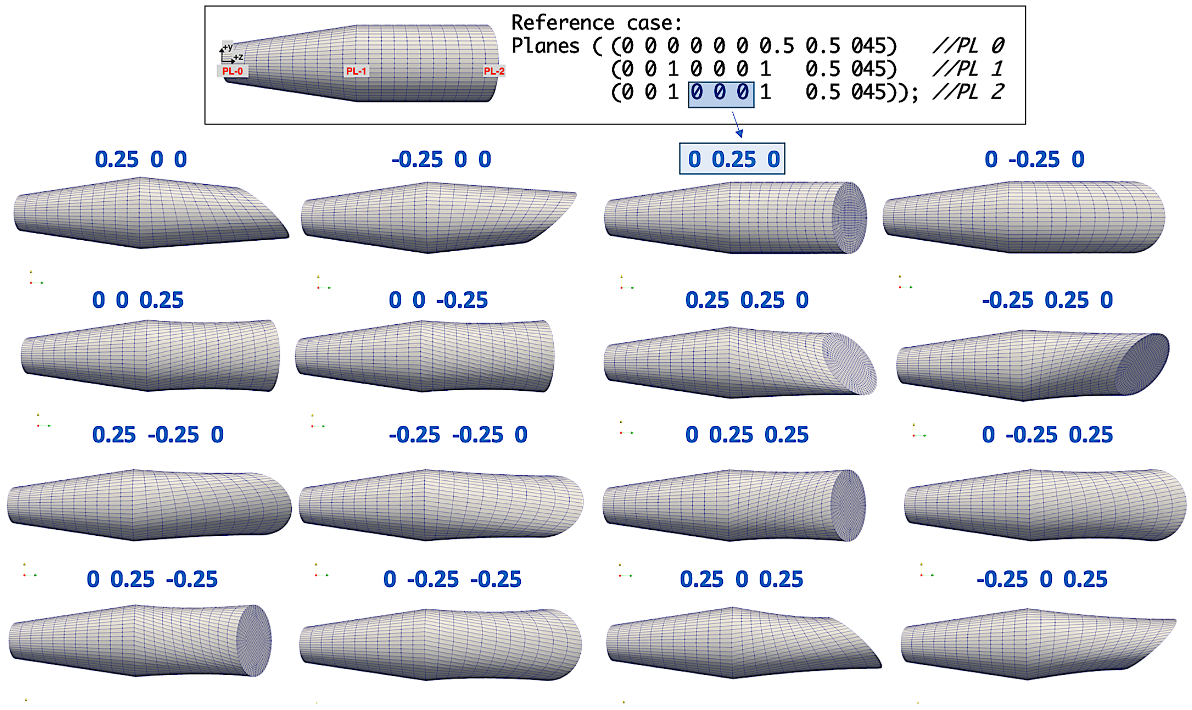


Figure 30: Plane rotation around axis Z, ω_z

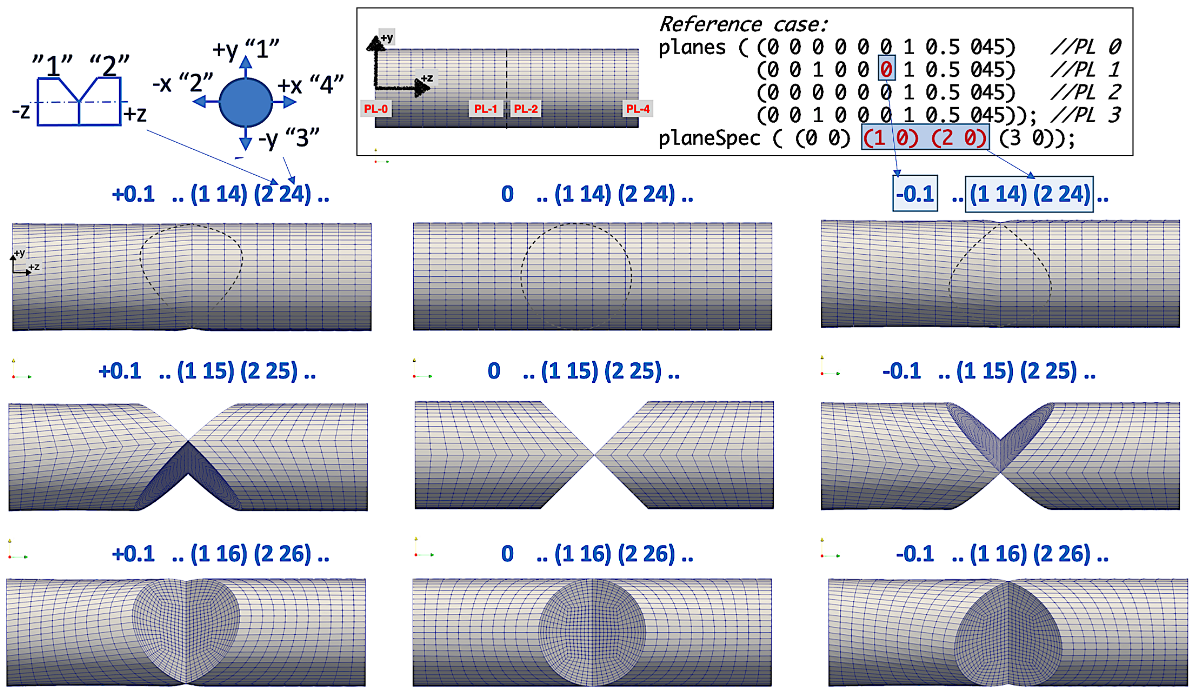
4.5.2 Examples of plane rotation

To understand the strategy of plane definition (rotation, inclination) and non-planar specification can help us follow examples, see Figures 31, 32 and 33.



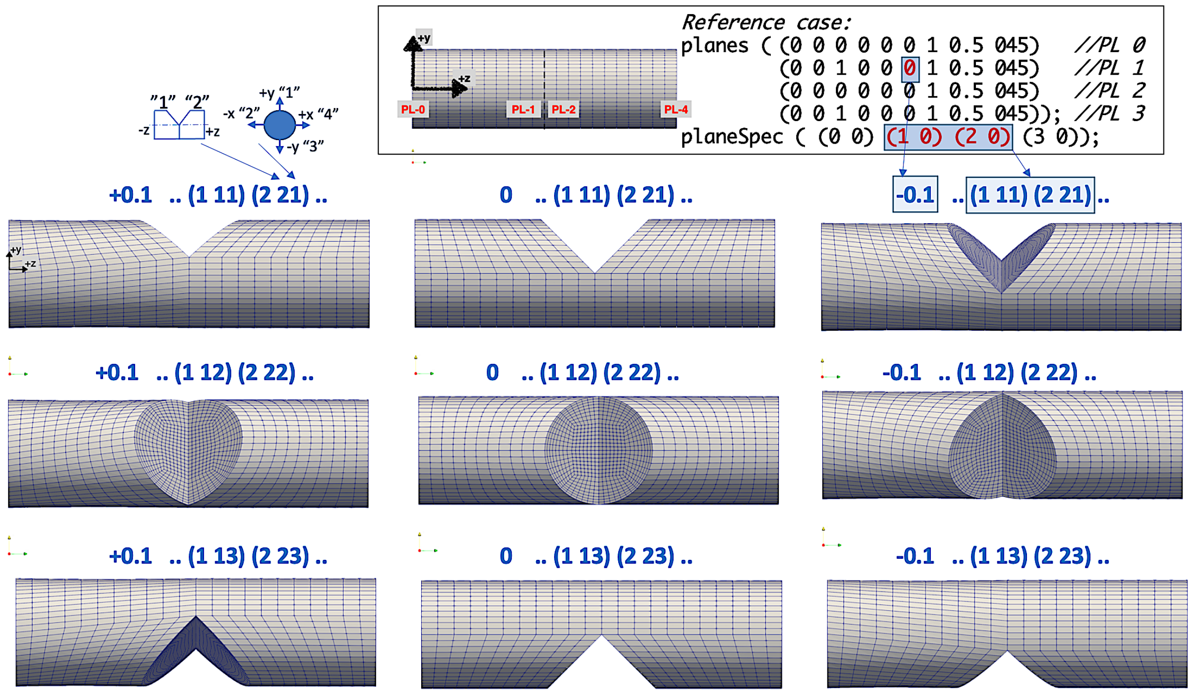
```
Reference case:
Planes ( (0 0 0 0 0 0 0.5 0.5 045)   //PL 0
         (0 0 1 0 0 0 1   0.5 045)   //PL 1
         (0 0 1 0 0 0 1   0.5 045)); //PL 2
```

Figure 31: catalog of shape a



```
Reference case:
planes ( (0 0 0 0 0 0 1 0.5 045)   //PL 0
         (0 0 1 0 0 0 1 0.5 045)   //PL 1
         (0 0 0 0 0 0 1 0.5 045)   //PL 2
         (0 0 1 0 0 0 1 0.5 045)); //PL 3
planeSpec ( (0 0) (1 0) (2 0) (3 0));
```

Figure 32: catalog of shape b, T junction



```
Reference case:
planes ( (0 0 0 0 0 0 1 0.5 045)   //PL 0
         (0 0 1 0 0 0 1 0.5 045)   //PL 1
         (0 0 0 0 0 0 1 0.5 045)   //PL 2
         (0 0 1 0 0 0 1 0.5 045)); //PL 3
planeSpec ( (0 0) (1 0) (2 0) (3 0));
```

Figure 33: catalog of shape c, plus junction

4.5.3 Point orientation

User must be careful fit orientation of pairs planes. for linear pipeline, the orientation should be the same or just slightly different, see Figure 34. The problem with different orientation (90,180,270 degree) can be solved with duplicity of the plane as we are using for T-junction construction.

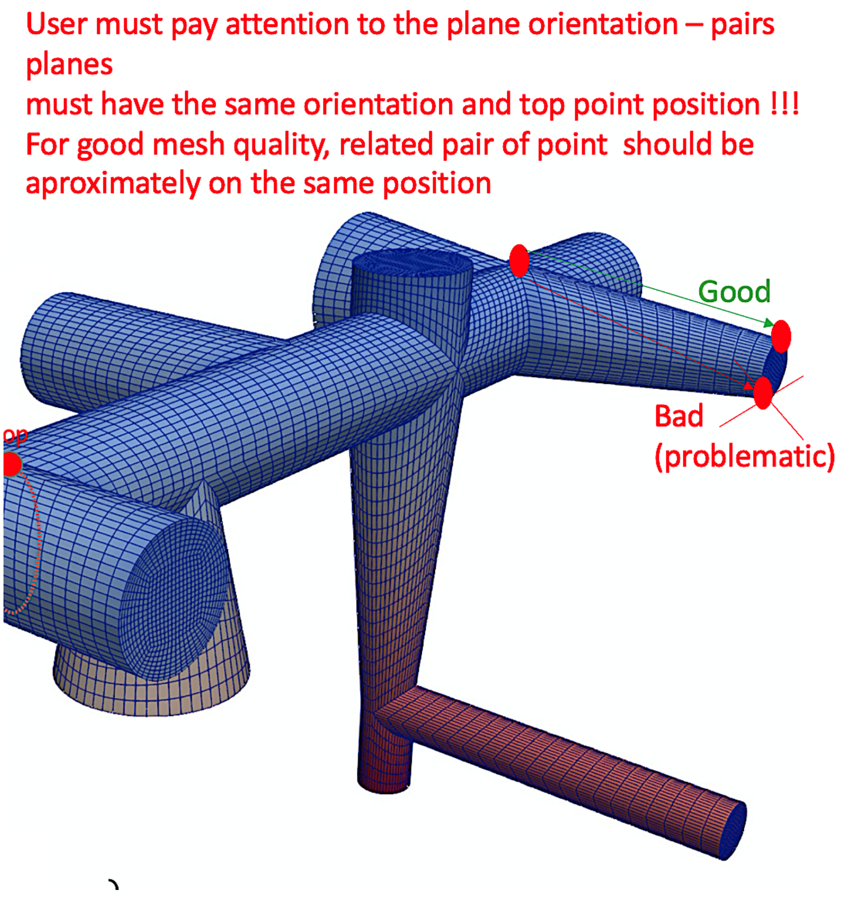

User must pay attention to the plane orientation – pairs planes
must have the same orientation and top point position !!!
For good mesh quality, related pair of point should be aproximately on the same position
Good
Bad (problematic)
op

Figure 34: Plane connection, good and bad orientation

4.6 Point duplicity check

T-junction is created from three planes on the same position. This fact create duplicity of the points which must be remove. The system check distance between each point and if it is smaller than specific value (10^{-5}), he program automatically remove the point with higher index. The final expect of duplicity check can be seen on the Figure 35. On the figure, just for demonstration, the distance on the left T-junction was artificially increase over limit, and then, this T-junction is not connected. The other T-junctions on the figure are correctly connected and duplicity of the points, respectively planes was deleted.

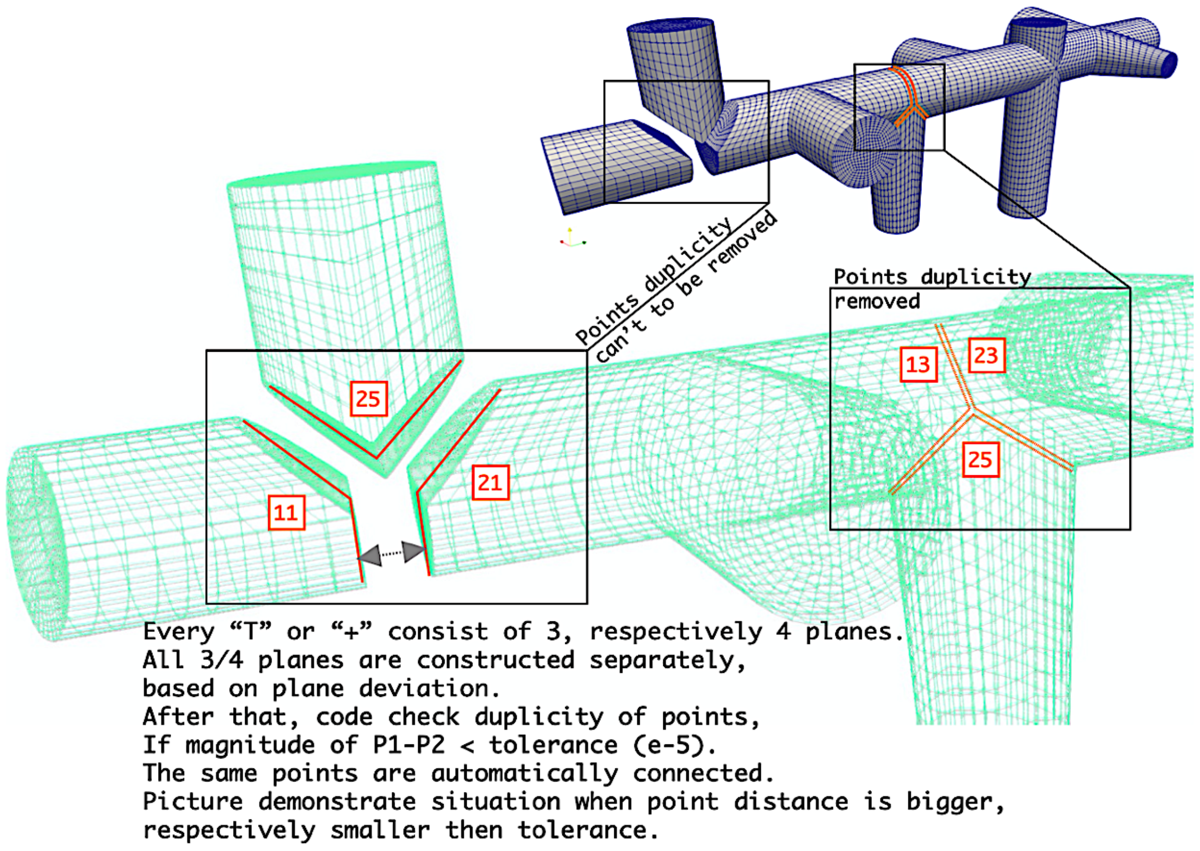

Points duplicity can't to be removed
Points duplicity removed
25
11
21
13
23
25
Every "T" or "+" consist of 3, respectively 4 planes.
All 3/4 planes are constructed separately,
based on plane deviation.
After that, code check duplicity of points,
If magnitude of P1-P2 < tolerance (e-5).
The same points are automatically connected.
Picture demonstrate situation when point distance is bigger,
respectively smaller then tolerance.

Figure 35: Plane duplicity, check and connection

4.6.1 Output to gnuplot graphs

Latest version v.2.5 include automatic gnuplot graph generation. it is given by follow code:

```
...
    system("gnuplot -e \"set grid; stats 'logGNU_3' u 1:2 nooutput;\
      set terminal png size 1600,1200;\
      set output 'graph_1.png';blocks = STATS_blocks ;\
      set key autotitle columnheader;\
      set multiplot layout 2,2 columnsfirst;\
      set xlabel 'Z axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_3' i i u 3:2 notitle w lines lw 10;\
      set xlabel 'Z axis'; set ylabel 'X axis';\
      plot for[i=0:blocks-1] 'logGNU_3' i i u 3:1 notitle w lines lw 10;\
      set xlabel 'X axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_3' i i u 1:2 w lines lw 10;\
      set xlabel 'X'; set ylabel 'Z'; set zlabel 'Y';\
      splot for[i=0:blocks-1] 'logGNU_3' i i u 1:3:2 notitle w lines lw 10\" ");

    forGnuplotPoint();
    system("gnuplot -e \"set grid; stats 'logGNU_2' u 1:2 nooutput;\
      set terminal png size 1600,1200;\
      set output 'graph_points.png';blocks = STATS_blocks ;\
      set key autotitle columnheader;\
      set style fill solid 1.00 border lt -1;\
      set multiplot layout 2,2 columnsfirst;\
      set xlabel 'Z axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_2' i i u 3:2 notitle w lines lw 10;\
      set xlabel 'Z axis'; set ylabel 'X axis';\
      plot for[i=0:blocks-1] 'logGNU_2' i i u 3:1 notitle w lines lw 10;\
      set xlabel 'X axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_2' i i u 1:2 w lines lw 10;\
      set xlabel 'X'; set ylabel 'Z'; set zlabel 'Y';\
      splot for[i=0:blocks-1] 'logGNU_2' i i u 1:3:2 notitle w lines lw 10\" ");
    os << points;
...
```

The system use information about center of pipelines for each plane and using pairs we can reconstruct simplified graph of geometry. This tool can be very useful for quick check of planes position. Graph are created based on variables and it is independent of real geometry description. It means, if user define wrong planes position and system crash with geometry construction, the gnuplot graph is independent and it is plot anytime and can help to understand the error. example of gnuplot output is on Figure 36.

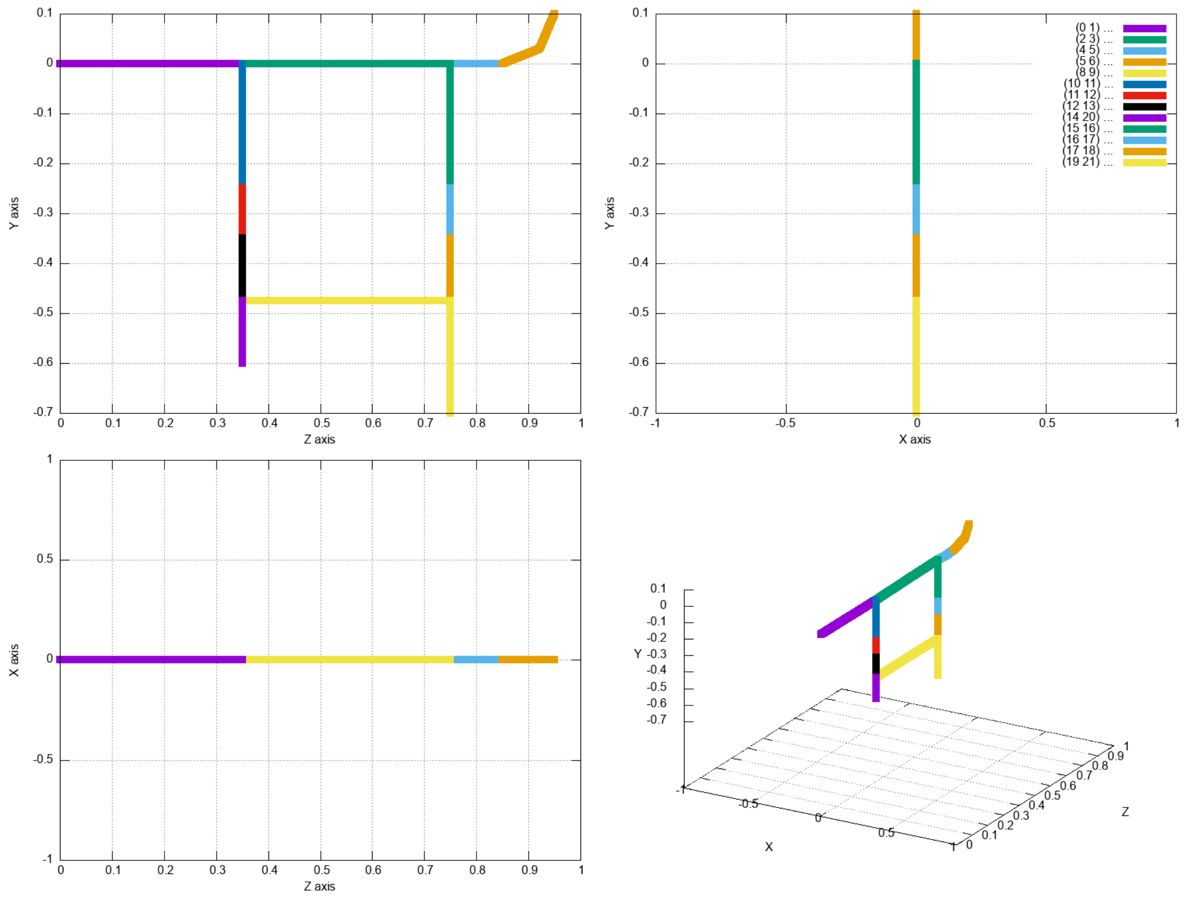


Figure 36: Gnuplot for quick geometry check

4.7 Log file

CodeStream log include plenty of information about the mesh compilation. To be able filter our output from system output, we are using signature "PTB" on the beginning of report line. After that, filtering the log file using :

```
        $ cat log | grep "PTB"
```

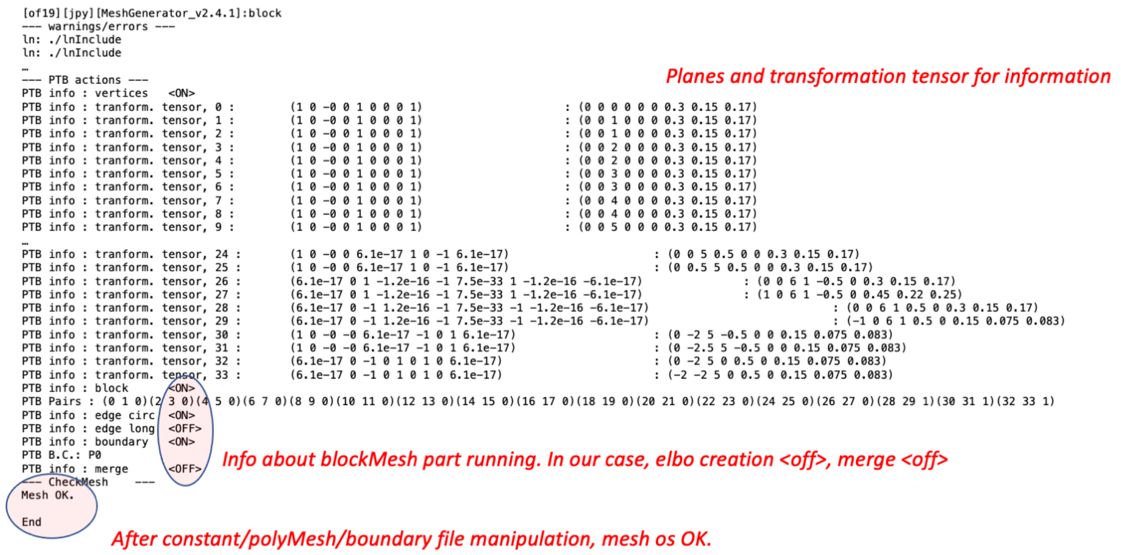
we can see follow report:

```
--- PTB actions ---
PTB info : MeshGenerator v.2.4.4, 12/12/2020, elbow bux fixed
PTB info : vertices <ON>
PTB info: no.: specif. of plane (plane) :(transform. tensor) :(N vec.):(T vec.)
PTB info : 0 : 30 :  (0 0 0 0 0 0 0.05 0.025 0.0225) :(1...1) :(0 0 1):(0 1 0)
PTB info : 1 : 33 :  (0 0 0.35 0 0 0 0.05 0.025 0.0225) :(1...1) :(0 0 1):(0 1 0)
PTB info : 2 : 43 :  (0 0 0.35 0 0 0 0.05 0.025 0.0225) :(1...1) :(0 0 1):(0 1 0)
PTB info : 3 : 33 :  (0 0 0.75 0 0 0 0.05 0.025 0.0225) :(1...1) :(0 0 1):(0 1 0)
PTB info : 4 : 43 :  (0 0 0.75 0 0 0 0.05 0.025 0.0225) :(1...1) :(0 0 1):(0 1 0)
PTB info : 5 : 30 :  (0 0 0.85 0 0 0 0.05 0.025 0.0225) :(1...1) :(0 0 1):(0 1 0)
...
PTB info : 21 : 30 :  (0 -0.7 0.75 -0.5...0.0125 0.0112) :(1...0) :(0 -1 0):(0 0 1)
PTB info : block <ON>
PTB info : Pairs : (0 1 0)(2 3 0)(4 5 0)(5 6 0)(8 9 0)(10 11 0)(11 12 0)(12 13 0)
(14 20 0)(15 16 0)(16 17 0)(17 18 0)(19 21 0)
PTB info : edge circ <ON>
PTB info : edge long <ON>
```

4.8 Elbow log

```
PTB info : pair  Middle plane    (A, B, R)   check for elbow
PTB info : (0 1 0) (0 0 0.175 0 0 0 0.05 0.025 0.0225) (0 0 0) Angles OK, linear
PTB info : (2 3 0) (0 0 0.55 0 0 0 0.05 0.025 0.0225) (0 0 0) Angles OK, linear
PTB info : (4 5 0) (0 0 0.8 0 0 0 0.05 0.025 0.0225) (0 0 0) Angles OK, linear
PTB info : (5 6 0) (0 0.0292893 0.920711 0.25 0 0 0.05 0.025 0.0225) (0.5 0.5 0.1)
Angles OK, elbow
PTB info : (8 9 0) (0 -0.475 0.55 0 0 0 0.025 0.0125 0.01125) (0 0 0)
Angles OK, linear
PTB info : (10 11 0) (0 -0.125 0.35 -0.5 0 0 0.05 0.025 0.0225) (0 0 0)
Angles OK, linear
PTB info : (11 12 0) (0 -0.3 0.35 -0.5 0 0 0.05 0.025 0.0225) (0 0 0)
Angles OK, linear
PTB info : (12 13 0) (0 -0.4125 0.35 -0.5 0 0 0.025 0.0125 0.01125) (0 0 0)
Angles OK, linear
PTB info : (14 20 0) (0 -0.5375 0.35 -0.5 0 0 0.025 0.0125 0.01125) (0 0 0)
Angles OK, linear
PTB info : (15 16 0) (0 -0.125 0.75 -0.5 0 0 0.05 0.025 0.0225) (0 0 0)
Angles OK, linear
PTB info : (16 17 0) (0 -0.3 0.75 -0.5 0 0 0.05 0.025 0.0225) (0 0 0)
Angles OK, linear
PTB info : (17 18 0) (0 -0.4125 0.75 -0.5 0 0 0.025 0.0125 0.01125) (0 0 0)
Angles OK, linear
PTB info : (19 21 0) (0 -0.5875 0.75 -0.5 0 0 0.025 0.0125 0.01125) (0 0 0)
Angles OK, linear
PTB info : boundary <ON>
PTB info: B.C. : P0 , P6 ,
PTB info : merge <OFF>
--- CheckMesh ---
Mesh OK.

End
```

Another example of log, including comments, can be seen on Figure 37



```
[of19][jpy][MeshGenerator_v2.4.1]:block
--- warnings/errors ---
ln: ./lnInclude
ln: ./lnInclude
…
--- PTB actions ---
PTB info : vertices   <ON>
PTB info : tranform. tensor, 0 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 0 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 1 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 1 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 2 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 1 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 3 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 2 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 4 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 2 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 5 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 3 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 6 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 3 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 7 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 4 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 8 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 4 0 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 9 :       (1 0 -0 0 1 0 0 0 1)                    : (0 0 5 0 0 0 0.3 0.15 0.17)
…
PTB info : tranform. tensor, 24 :      (1 0 -0 0 6.1e-17 1 0 -1 6.1e-17)                    : (0 0 5 0.5 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 25 :      (1 0 -0 0 6.1e-17 1 0 -1 6.1e-17)                    : (0 0.5 5 0.5 0 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 26 :      (6.1e-17 0 1 -1.2e-16 -1 7.5e-33 1 -1.2e-16 -6.1e-17)             : (0 0 6 1 -0.5 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 27 :      (6.1e-17 0 1 -1.2e-16 -1 7.5e-33 1 -1.2e-16 -6.1e-17)             : (1 0 6 1 -0.5 0 0.45 0.22 0.25)
PTB info : tranform. tensor, 28 :      (6.1e-17 0 -1 1.2e-16 -1 7.5e-33 -1 -1.2e-16 -6.1e-17)                        : (0 0 6 1 0.5 0 0.3 0.15 0.17)
PTB info : tranform. tensor, 29 :      (6.1e-17 0 -1 1.2e-16 -1 7.5e-33 -1 -1.2e-16 -6.1e-17)                        : (-1 0 6 1 0.5 0 0.15 0.075 0.083)
PTB info : tranform. tensor, 30 :      (1 0 -0 -0 6.1e-17 -1 0 1 6.1e-17)                   : (0 -2 5 -0.5 0 0 0.15 0.075 0.083)
PTB info : tranform. tensor, 31 :      (1 0 -0 -0 6.1e-17 -1 0 1 6.1e-17)                   : (0 -2.5 5 -0.5 0 0 0.15 0.075 0.083)
PTB info : tranform. tensor, 32 :      (6.1e-17 0 -1 0 1 0 1 0 6.1e-17)                     : (0 -2 5 0 0.5 0 0.15 0.075 0.083)
PTB info : tranform. tensor, 33 :      (6.1e-17 0 -1 0 1 0 1 0 6.1e-17)                     : (-2 -2 5 0 0.5 0 0.15 0.075 0.083)
PTB info : block      <ON>
PTB Pairs : (0 1 0)(2 3 0)(4 5 0)(6 7 0)(8 9 0)(10 11 0)(12 13 0)(14 15 0)(16 17 0)(18 19 0)(20 21 0)(22 23 0)(24 25 0)(26 27 0)(28 29 1)(30 31 1)(32 33 1)
PTB info : edge circ  <ON>
PTB info : edge long  <OFF>
PTB info : boundary   <ON>
PTB B.C.: P0
PTB info : merge      <OFF>
--- CheckMesh    ---
Mesh OK.

End
```

Planes and transformation tensor for information
Info about blockMesh part running. In our case, elbo creation <off>, merge <off>
After constant/polyMesh/boundary file manipulation, mesh os OK.

Figure 37: Example of log file, running function monitoring

4.9 Run blockMesh for generator

We recommend to prepare alias for running block Mesh including few specific parameters. If some changes on codestream, it it necessary to delete directory dynamicCode, see follow bash stile code:

```
alias block="[ -d dynamicCode ] && rm -r dynamicCode;\
[ -d constant/polyMesh ] && rm -r constant/polyMesh;\
blockMesh > logPTB 2>logERR ;\
echo '--- warnings/errors ---'; \
cat logERR; echo '--- PTB actions ---'; cat logPTB | grep 'PTB';\
sed -i'' -e s/'empty'/'wall'/g constant/polyMesh/boundary;\
sed -i'' -e s/'defaultFaces'/'wall'/g constant/polyMesh/boundary;\
sed -i'' -e s/'P0'/'inlet1'/g constant/polyMesh/boundary;\
sed -i'' -e s/'P15'/'inlet2'/g constant/polyMesh/boundary;\
echo '--- CheckMesh ---'; checkMesh | tail -n 4 "
```

The present alias is specific for one geometry and can't to be use generally, but user can use for inspiration. The meaning of selected commends:

```
[ -d constant/polyMesh ] && rm -r constant/polyMesh
```

Check if previous mesh is present and if yes, delete it.

```
blockMesh > logPTB 2>logERR
```

Forward the output to logPTB file and warnings and error output to logERR file.

```
echo '--- warnings/errors ---'; cat logERR;
echo '--- PTB actions ---'; cat logPTB | grep 'PTB'
```

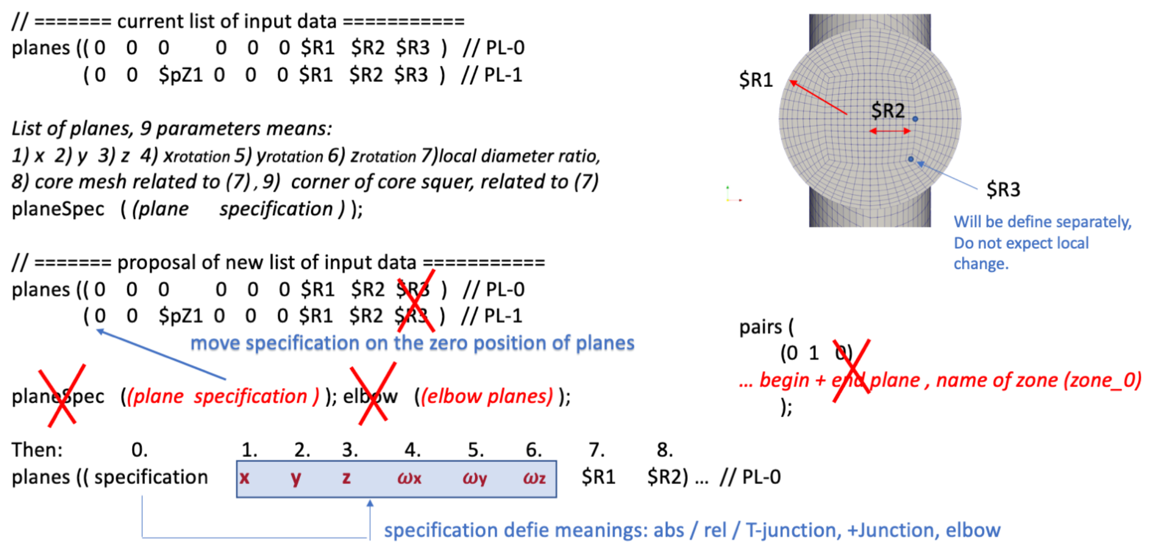
Plot error and warnings, if any, plot lines start with pattern 'PTB'.

Another example of alias. This one also call gnuplot for quick overview the geometry, but it is not necessary anymore, the new version include gnuplot output inside the codeStream.

```
alias blockPlot="blockMesh > logPTB 2>logERR;\
awk '/ForGnu1/ {print \$2, \$3, \$4}' logPTB > logGNU;\
echo '--- PTB actions ---'; cat logPTB | grep 'PTB';\
gnuplot -e \"set grid; stats 'logGNU' u 1:2 nooutput;
set terminal png size 1600,1200; set output 'xyz.png';
blocks = STATS_blocks ;set key autotitle columnheader;
set multiplot layout 2,2 columnsfirst;
plot for[i=0:blocks-1] 'logGNU' i i u 3:2 notitle w lines lw 10;
plot for[i=0:blocks-1] 'logGNU' i i u 3:1 notitle w lines lw 10;
plot for[i=0:blocks-1] 'logGNU' i i u 1:2 w lines lw 10;
splot for[i=0:blocks-1] 'logGNU' i i u 1:3:2 notitle w lines lw 10\" "
```

4.10 Potential future modification and extension

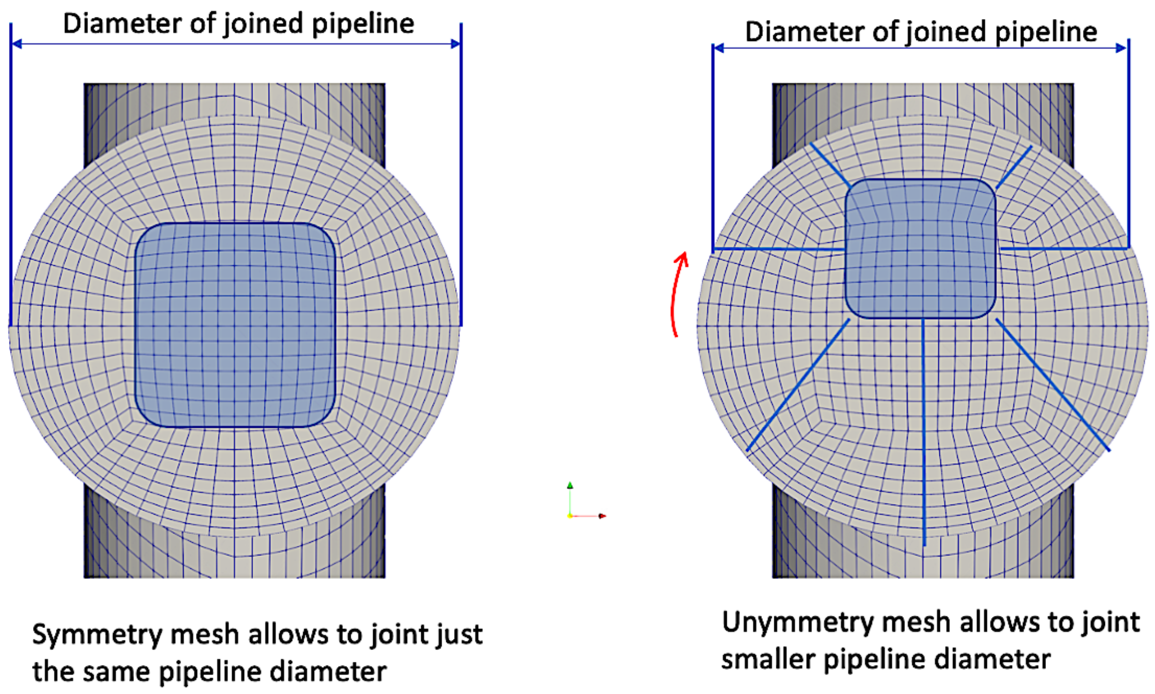
Current version 2.5 allow all required functionality for pipeline system mesh generation. But still, it is possible to extend functionality and improve the current code. One of the possible future change is to reorganize the variable planes as proposed in Figure 38. The idea is remove parameter $R3 and free position use rather for plane specification. The parameter $R3 can be set as a default string for all pipeline system.



```
// ======= current list of input data ===========
planes (( 0   0   0        0   0   0  $R1   $R2  $R3  )   // PL-0
        ( 0   0   $pZ1  0   0   0  $R1   $R2  $R3  )   // PL-1

List of planes, 9 parameters means:
1) x  2) y  3) z  4) xrotation 5) yrotation 6) zrotation 7)local diameter ratio,
8) core mesh related to (7) , 9)  corner of core squer, related to (7)
planeSpec   ( (plane     specification ) );

// ======= proposal of new list of input data ===========
planes (( 0   0   0        0   0   0  $R1   $R2  $R3  )   // PL-0
        ( 0   0   $pZ1  0   0   0  $R1   $R2  $R3  )   // PL-1
move specification on the zero position of planes

planeSpec   ((plane  specification ) ); elbow   ((elbow planes) );

Then:          0.                1.     2.    3.       4.       5.      6.       7.       8.
planes (( specification     x      y     z      ωx      ωy     ωz      $R1     $R2) ...  // PL-0
specification defie meanings: abs / rel / T-junction, +Junction, elbow

pairs (
      (0  1  0)
... begin + end plane , name of zone (zone_0)
      );
```

Figure 38: BlockMeshDict, list of input parameters

Second important extension can be manipulation with the mesh core, see Figure 39. This modification can allow us T-junction of two different pipeline diameters.



Diameter of joined pipeline
Diameter of joined pipeline
Symmetry mesh allows to joint just the same pipeline diameter
Unymmetry mesh allows to joint smaller pipeline diameter

Figure 39: T junction, different pipeline diameter

4.11 Transformation

The matrix of transformation is used for new plane creation. We are using general rotations matrix:

$$\begin{pmatrix} C(\omega_y)C(\omega_z) & C(\omega_y)S(\omega_z) & -S(\omega_y) \\ S(\omega_x)S(\omega_y)C(\omega_z) - C(\omega_x)S(\omega_z) & C(\omega_x)C(\omega_z) + S(\omega_x)S(\omega_y)S(\omega_z) & S(\omega_x)C(\omega_y) \\ C(\omega_x)S(\omega_y)C(\omega_z) + S(\omega_x)S(\omega_z) & C(\omega_x)S(\omega_y)S(\omega_z) - S(\omega_x)C(\omega_z) & C(\omega_x)C(\omega_y) \end{pmatrix} \tag{1}$$

where $C()$ is $cos()$, and $S()$ is $sin()$ and ω_x, ω_y and ω_z represents a rotation whose yaw, pitch, and roll angles. These matrices produce the desired effect only if they are used to premultiply column vectors, and (since in general matrix multiplication is not commutative) only if they are applied in the specified order.

5 Mesh Generator code

Next part include the code of all three files: **blockmeshDict**, include variables, **codeStreamPTB**, include blockmesh code and calling propriety function from third file **functionsPTB.H**.

5.1 File blockMeshDict

5.1.1 User define variables

Example of geometrical variables used for parametrisation of geometry.

```
// --------- User define variables ------

pZ1 1;  // positive Z direction
pZ2 1;
pZ3 1;
pZ4 1;
pZ5 1;
pZ6 1;
pZ7 1;

pY1 1;  // positive Y direction
pY2 1;
pY3 2;
pY4 2.5;
pY5 0.5;

pX1 1;  // positive X direction
pX2 1.5;
```

```
pX3 1;
pX4 1.5;
pX5 2;

R1 1.0;
R1s 0.5; // smaller Radius
R1b 1.5; // bigger Radius
R2 0.50;
R3 0.45;

pA 0.5; // 1/2*Pi angel

// ---- aditional derived variables -----
nZ17 #calc " -$pZ1-$pZ2-$pZ3-$pZ4-$pZ5-$pZ6-$pZ7 ";
pZ15 #calc " $pZ1+$pZ2+$pZ3+$pZ4+$pZ5 ";

nZ1 #calc " -$pZ1";
nZ2 #calc " -$pZ2";
nZ3 #calc " -$pZ3";
nZ4 #calc " -$pZ4";
nZ5 #calc " -$pZ5";
nZ6 #calc " -$pZ6";
nZ7 #calc " -$pZ7";

nY1 #calc " -$pY1";
nY2 #calc " -$pY2";
nY3 #calc " -$pY3";
nY4 #calc " -$pY4";
nY5 #calc " -$pY5";

nX1 #calc " -$pX1";
nX2 #calc " -$pX2";
nX3 #calc " -$pX3";
nX4 #calc " -$pX4";
nX5 #calc " -$pX5";

nA #calc " -$pA";
p2A #calc " 2*$pA";
```

Another example for additional geometrical variables.

```

D 0.2; // diameter of pipe

G1 0.5; // o-grid parameters
```

```
G2 0.45; // o-grid parameters

H 0.2; // height of T above pipe

Z1 1.0; // position of T
Z2 2.0; // position of bend

nA -0.5; // -90 degrees
pA 0.5; // 90 degrees

alpha 89; // angle (has to be smaller or equal to 90 degrees)
rot -1.0; // forward rotation (rot=1) or backward rotation (rot=-1)?

PI 3.14159265358979323846;


// ---- additional derived variables -----

R #calc " $D/2.0"; // pipe radius

Y1 #calc " $R+$H"; // height of T-outlet

alpharad #calc " $PI*$alpha/180.0"; // alpha in rad
nalpha #calc " $rot*$alpha/180.0"; // alpharad in multiples of pi times rotation

Rc #calc " 4*$D"; // curvature radius (Rc=4D)

deltax #calc " $Rc*(1-cos($alpharad))";
deltaz #calc " $Rc*sin($alpharad)";
```

5.1.2 Obligatory variables

```
scale 1;
radiuS 1.0;

planes (( 0 0 0 0 0 0 $R $G1 $G2 ) // PL-0
       ( 0 0 $Z1 0 0 0 $R $G1 $G2 ) // PL-1 abs
       ( 0 0 $Z1 0 0 0 $R $G1 $G2 ) // PL-2 abs
       ( 0 0 $Z1 $pA 0 0 $R $G1 $G2 ) // PL-3 abs
       ( 0 $Y1 $Z1 $pA 0 0 $R $G1 $G2 ) // PL-4 abs
       ( 0 0 $Z2 0 0 0 $R $G1 $G2 ) // PL-5 abs
//
       ( $deltax 0 $deltaz 0 $nalpha 0 $R $G1 $G2 ) // PL-6 rel
// ( 0.526384 0 0.751754 0 -0.388889 0 $R $G1 $G2 )
// PL-6 rel for alpha = 70 degrees
// ( $Rc 0 $Rc 0 $nA 0 $R $G1 $G2 ) // PL-6 rel for alpha = 90 degrees
       );

// if no present, default coordinate system will be relativ = 0
// defaultSystem 30;

planeSpec (
           (1 31 )
           (2 41 )
           (3 45 )
           (4 30 )
           (5 30 )
           //(6 0 ) // elbow
           );

pairs (
       (0 1 0)
       (2 5 0)
       (3 4 0)
       (5 6 0)
       );

freePlanes ( 0 4 6);


// ---- filter -----
blockUsed ( );
planeUsed ( );

// ---- MESH ----
mesh (7 8 50 2);

// ====== switch =====

shape 0;
```

5.1.3 Optional variables

```

// --- OPTIONAL VARIABLES ---

```

```
switcher ( 1 // vortices
           1 // blocks
           1 // edges circumferential
           1 // edges longitudial
           0 // boundary
           0 // mergePatchPairs
       ); // default (1 1 1 1 1 1)
```

5.1.4 Inclusion of CodeStream part

Every blockMeshDict must be closed by including codeStreamPTB file.

```
// ---- codeStream for pipeline Mesh generator, v.2.4, 29.11.2020 ---
#include "codeStreamPTB";
```

5.2 File codeStreamPTB

codeStreamPTB file generate original blockMeshDick code by calling specific functions from functionsPTB.H. The hierarchy is given: **vertices**, generate list of points, **blocks** generate list of hexahedral blocks, **edge**, generate arc, elbow, cylinders, **boundary** specify patches for future boundary conditions and **mergePatchPairs** for patch merges.

5.2.1 Call geometry functions

```
// --- blockMesh main body ---
vertices
#codeStream
{
    codeInclude #{#include "functionsPTB.H" #};
    codeOptions #{-I$(FOAM_CASE)/system #};
    code #{generator genPTB; genPTB.VerticeS(os,dict);#};
};

blocks
#codeStream
{
    codeInclude #{#include "functionsPTB.H" #};
    codeOptions #{-I$(FOAM_CASE)/system #};
    code #{generator genPTB; genPTB.BlockS(os,dict); #};
};

edges
#codeStream
{
    codeInclude #{#include "functionsPTB.H" #};
    codeOptions #{-I$(FOAM_CASE)/system #};
    code #{generator genPTB; genPTB.EdgeS(os,dict); #};
};

boundary
#codeStream
{
    codeInclude #{#include "functionsPTB.H" #};
    codeOptions #{-I$(FOAM_CASE)/system #};
    code #{generator genPTB; genPTB.BoundarY(os,dict); #};
};

mergePatchPairs
#codeStream
{
    codeInclude #{#include "functionsPTB.H" #};
    codeOptions #{-I$(FOAM_CASE)/system #};
    code #{generator genPTB; genPTB.MergeP(os,dict); #};
};

// ********************************************* //
```

5.2.2 System variables for core

```
// --- SYSTEM VARIABLES ---
// 0 1 2 3 4 5 6 7 8
pointPAR (
        (7 0 0 0 0 1 0 1 -1) // 0
        (8 0 1 0 0 1 1 1 -1) // 1
        (7 0 1 0 0 1 1 -1 -1) // 2
        (8 0 1 1 0 0 1 -1 -1) // 3
        (7 0 1 1 0 0 1 -1 -1) // 4
        (8 0 0 1 1 0 -1 -1 -1) // 5
        (7 0 0 1 1 1 -1 -1 -1) // 6
        (8 0 0 0 1 1 -1 1 -1) // 7
        (6 0 0 0 1 1 -1 1 -1) // 8
        (6 0 1 0 0 1 1 1 -1) // 9
        (6 0 1 0 0 1 1 1 1) //10
        (6 0 1 1 0 0 1 -1 -1) //11
        (6 0 1 1 0 0 1 -1 -1) //12
        (6 0 0 1 1 0 -1 -1 -1) //13
        (6 0 0 1 1 0 -1 -1 -1) //14
        (6 0 0 0 1 1 -1 1 -1) //15
        (6 0 0 0 1 1 -1 1 -1) //16
        );

R0 (
    (1 0) (0 1) (0 1) (-1 0) (-1 0) (0 -1) (0 -1) (1 0)
    (0 0) (0 0) (0 0) ( 0 0) ( 0 0) (0 0) (0 0) (0 0)
    );
```

5.2.3 System variables for reference blocks

```
blockUsed (0 1 2 3 4 5 6 7 8 9 10 11 );

blockREF (
           (0 1 2 3)
           (0 3 4 5)
           (0 5 6 7)
           (0 7 8 1)
           (1 9 10 2)
           (2 10 11 3)
           (3 11 12 4)
           (4 12 13 5)
           (5 13 14 6)
           (6 14 15 7)
           (7 15 16 8)
           (8 16 9 1)
           );
```

5.2.4 System variables for plane deviation

```
deviation (
           (0 1 1 1 )
           (11 1 2 -1 )
           (12 3 1 1 )
           (13 1 4 1 )
           (14 5 1 -1 )
           (15 1 6 -1 )
           (16 1 6 1 )
           (17 7 1 -1 )
           (18 7 1 1 )
           (21 1 2 1 )
           (22 3 1 -1 )
           (23 1 4 -1 )
```

```
            (24 5 1 1 )
            );
```

5.3 File functionsPTB.H

5.3.1 Including header files

```
#include "Vector2D.H"
#include "tensor2D.H"
#include "pointField.H"
#include "IOstream.H"
#include "codeStream.H"
#include "functionEntry.H"
#include <math.h>
#include "IOmanip.H"
#include "OFstream.H"

//#include <vector2D>

using namespace Foam;

class generator{
public:
symmTensor switcher; // 6
label noPlanes, noPoints, noPointsA, noPairs,
                                    noBlocks, blockOnPlane;// 1 int
scalar radiuS, scale; // 1
pointField pREF, pointsROT, cells, pairs, points,
                                    pointsA, pAUX, N, T, M, I, U, pointsC;// n * 3
scalarField blockUsed, mesh, BL, BCa, BCb, freePlanes, usedPlane; // n * 1
labelField planeS; // n * 1
Field<tensor2D> blockREF, deviation ; // n * 2
Field<vector2D> RO;
symmTensorField BB; // n * 4
tensorField planes, pointPAR, planesC; // n * 9
# define PI 3.14159265358979323846264338327950288
label NO=17; // no of points on plane
label NOAUX = 145;
word fName;
tensor tRef;
autoPtr<OFstream> LOGPTB;
```

5.3.2 Variable declaration

```
// --- generator member functions ----

  // --- aditional functions ---

  void LookuP(const dictionary& dict)
  {
    planes = tensorField(dict.lookup("planes"));
    radiuS = readScalar(dict.lookup("radiuS"));
    scale = readScalar(dict.lookup("scale"));
    pointPAR = tensorField(dict.lookup("pointPAR"));
    noPlanes = planes.size();
    noPoints = planes.size() * NO;
    noPointsA = planes.size() * NOAUX;
    points = pointField(noPoints, point(0,0,0));
    points = pointField(noPoints, point(0,0,0));
    points = pointField(noPoints, point(0,0,0));
    pointsA = pointField(noPointsA, point(0,0,0));
    pointsROT = pointField(noPoints, point(0,0,0));
    Field<vector2D> planeSpec = Field<vector2D>(dict.lookup("planeSpec"));
    scalar defaultSystem(dict.lookupOrDefault<scalar>("defaultSystem", 0) );

    planeS = labelField(noPlanes, label(defaultSystem));
    changeDef(planeS, planeSpec );
    pREF = pointField(NO, point(0,0,0));
    pAUX = pointField(NOAUX,point(0,0,0));
    planeSET();
    pairs = pointField(dict.lookup("pairs"));
    noPairs = pairs.size();
    N = pointField(noPlanes, point(0,0,0)); // normal unit vectro
    T = pointField(noPlanes, point(0,0,0)); // tangential unit vector
    M = pointField(noPairs, point(0,0,0)); // vector of pair
    I = pointField(noPairs, point(0,0,0)); // angel + magnitud M
    U = pointField(noPairs, point(0,0,0)); // angel + magnitud M
    pointsC = pointField(NO*noPairs,point(9,9,9)); // points on midle planes
    planesC = tensorField(noPairs, tensor(0,0,0,0,0,0,0,0,0));
                                                        // planes on the midle
    blockREF = Field<tensor2D>(dict.lookup("blockREF"));
    R0 = Field<vector2D>(dict.lookup("R0"));
    mesh = scalarField(dict.lookup("mesh"));
    blockUsed = scalarField(dict.lookup("blockUsed"));
    blockOnPlane = 12;
    deviation = Field<tensor2D>(dict.lookup("deviation"));
    BL = scalarField(noPlanes * NO, scalar(0));
    for (label i=0 ; i < BL.size(); i++){BL[i] = i;}
    BCa = scalarField(noPlanes * NO, scalar(0));
    BCb = scalarField(noPlanes * NO, scalar(0));
    freePlanes = scalarField(dict.lookup("freePlanes"));
    usedPlane = scalarField(2*noPairs, scalar(0));
    LOGPTB.reset(new OFstream("log_PTB"));
    LOGPTB() << "# ====== logPTB ======" << endl;
    label x =0;
    for (label i=0; i < noPairs ; i++ )
    {
      usedPlane[x]  = pairs[i][0];
      usedPlane[x+1]  = pairs[i][1];
      x=x+2;
    }
  }
```

5.3.3 Function for duplicity check

```
  void checkDup()
  {
    scalar ERR = 0.001;
    for (label i=0 ; i < BL.size() ; i++)
    {
      for (label j=i+1 ; j < BL.size(); j++)
      {
        scalar D = mag(points[BL[i]] - points[BL[j]]);
        if (D < ERR ){ BL[j] = BL[i]; }
      }
    }
  }
```

5.3.4 Function of coordinate system

```
  void changeDef(labelField& A, Field<vector2D> B)
  {
    for (label i=0; i < B.size(); i++ )A[B[i][0]] = B[i][1];
  }
```

5.3.5 Function empty code

```
 void emptyCode(Ostream& os, word empty="function was bypassed")
  {
    INFO(empty);
    os << " ( " << nl << " ) " << nl;
  }
```

5.3.6 Function for gnuplot data of plane generation

```
void forGnuplotLine()
{
  autoPtr<OFstream> outputFilePtr;
  outputFilePtr.reset(new OFstream("logGNU_3"));
  outputFilePtr() << "# The list of planes central point,
  grouped according pairs, including middle plane" << endl;

  for (label pa=0; pa < noPairs; pa++)
  {
    label a = pairs[pa][0], b = pairs[pa][1];
    outputFilePtr() << "\"" << "(" << a << " " << b << ") ... \" " << nl;
    outputFilePtr() << planes[a][0] << " " << planes[a][1] << " "
    << planes[a][2] << nl;
    outputFilePtr() << planesC[pa][0] << " " << planesC[pa][1] << " "
    << planesC[pa][2] << nl;
    outputFilePtr() << planes[b][0] << " " << planes[b][1] << " "
    << planes[b][2] << nl;
    outputFilePtr() << nl;
    outputFilePtr() << nl;
  }
}
```

5.3.7 Function for gnuplot data of points generation

```
void forGnuplotPoint()
{
  autoPtr<OFstream> outputFilePoint;
  outputFilePoint.reset(new OFstream("logGNU_2"));
  outputFilePoint() << "# The list of points" << endl;

  for (label p=0; p < noPlanes; p++)
  {
      outputFilePoint() << "Plane:" << p << nl;
      for (label i =9; i < NO; i++)
      {
        outputFilePoint() << points[i+NO*p][0] << "\t" << points[i+NO*p][1] <<
        "\t" << points[i+NO*p][2] << nl;
      }
      outputFilePoint() << points[9+NO*p][0] << "\t" << points[9+NO*p][1] <<
      "\t" << points[9+NO*p][2] << nl;
      outputFilePoint() << nl;
      outputFilePoint() << nl;
    }
  }
```

5.3.8 Function of switch variable definition

```
 void SwitcheR(const dictionary& dict)
  {
      switcher =symmTensor(dict.lookupOrDefault("switcher", symmTensor(1,1,1,1,1,1)))
  }
```

5.3.9 Function point setting

```
  void pointSET(label pl, scalar S=0.25)
   {
    label k1,k2,k3, PLA;
    PLA = planeS[pl];
    if ( PLA == 0 ){k1=1 ; k2=1; k3= 1; }
    if ( PLA == 11 || PLA == 31 ){k1=1 ; k2=2; k3=-1; }
    if ( PLA == 12 || PLA == 32 ){k1=3 ; k2=1; k3= 1; }
    if ( PLA == 13 || PLA == 33 ){k1=1 ; k2=4; k3= 1; }
    if ( PLA == 14 || PLA == 34 ){k1=5 ; k2=1; k3=-1; }
    if ( PLA == 15 || PLA == 35 ){k1=1 ; k2=6; k3=-1; }
    if ( PLA == 25 || PLA == 45 ){k1=1 ; k2=6; k3= 1; }
    if ( PLA == 16 || PLA == 36 ){k1=7 ; k2=1; k3=-1; }
    if ( PLA == 26 || PLA == 46 ){k1=7 ; k2=1; k3= 1; }
    if ( PLA == 21 || PLA == 41 ){k1=1 ; k2=2; k3= 1; }
    if ( PLA == 22 || PLA == 42 ){k1=3 ; k2=1; k3=-1; }
    if ( PLA == 23 || PLA == 43 ){k1=1 ; k2=4; k3=-1; }
    if ( PLA == 24 || PLA == 44 ){k1=5 ; k2=1; k3= 1; }
    // ---- main reference list of points (17 per planes)
    for (label i=0; i < NO-1 ; i++ )
    {
      scalar R = planes[pl][pointPAR[i][0]];
      if ( i==1 || i==3 || i==5 || i==7 ){ R = R * std::pow(2.0,0.5);}
      pREF[i+1] = point(std::cos(i*S*PI)*R,
                         std::sin(i*S*PI)*R,
                        (std::cos(i*S*PI)*R*pointPAR[i][k1]
                        +std::sin(i*S*PI)*R*pointPAR[i][k2])*k3);
    }
    if ( 1 > 0 ) //fName == "EdgeS")
    {
      // ---- auxilitary reference list of points ( per planes)
      label po = 1;
      for (label q = 0 ; q < 16; q++)
      {
        for ( scalar a = 0 ; a < 45 ; a=a+5)
        {
          scalar Alfa;
          scalar R;
          if (q < 8)
          {
            R = (pow(planes[pl][7]-planes[pl][8],2)+pow(planes[pl][8],2))/
              (planes[pl][7]-planes[pl][8]) * 0.5;
            scalar RA = std::asin(planes[pl][8]/R)*1.25;
            //Alfa = q * 45 + a * RA ;
            if ( q % 2 == 0) {Alfa = q*45 + a*RA;}
            if ( q % 2 != 0) {Alfa = (q+1)*45 + RA*(a-45);}
          }
          if (q >= 8)
          {
            Alfa = q * 45 + a;
            R = planes[pl][pointPAR[q][0]];
          }
          pAUX[po] = point(std::cos(Alfa*PI/180)*R + R0[q][0]*(planes[pl][7]-R),
                           std::sin(Alfa*PI/180)*R + R0[q][1]*(planes[pl][7]-R),
                   (std::cos(Alfa*PI/180)*R*pointPAR[q+1][k1]
                   +std::sin(Alfa*PI/180)*R*pointPAR[q+1][k2])*k3 );
               // +R0[q][0]*(planes[pl][7]-R) + R0[q][0]*(planes[pl][7]-R));
          po++;
        }
      }
    }
  }
```

5.3.10 Function plane setting

```
  void planeSET()
  {
    for (label i =0 ; i < noPlanes; i++)
    {
      planes[i][6] = planes[i][6]*radiuS;
      planes[i][7] = planes[i][6]*planes[i][7];
      planes[i][8] = planes[i][6]*planes[i][8];
    }
    for (label i =1 ; i < noPlanes; i++)
    {
      if ( planeS[i] < 29 && planeS[i] != 3){for (label j=0; j <6; j++)
          {planes[i][j]= planes[i][j] + planes[i-1][j];}}
    }
  }
```

5.3.11 Function trigonometric

```
  scalar SIN(label i,label j, label m=1)
  {
    if (m==1) {return std::sin( planes[i][j]*PI);}
    else {return std::sin(planesC[i][j]*PI);}
  }

  scalar COS(label i,label j, label m=1)
  {
    if (m==1) {return std::cos( planes[i][j]*PI);}
    else {return std::cos(planesC[i][j]*PI);}
  }
```

5.3.12 Function check edge existing

```
  bool CheckE(label A, label B)
  {
    for (label i=0 ; i < BCa.size() ; i++)
    {
      if (BCa[i] == A && BCb[i] == B){ return false;}
      if (BCa[i] == B && BCb[i] == A){ return false;}
    }
    return true;
  }
```

5.3.13 Function check point existing

```
  bool checkP(label A)
  {
    for (label i=0 ; i < usedPlane.size() ; i++)
    {
      if (usedPlane[i] == A ){ return true;}
    }
    return false;
  }
```

5.3.14 Function of reference plane

```
void tRefGEN(label i, label m)
{
    tRef = tensor( COS(i,4,m)*COS(i,5,m), COS(i,4,m)*SIN(i,5,m) , -1*SIN(i,4,m),
       SIN(i,3,m)*SIN(i,4,m)*COS(i,5,m)-COS(i,3,m)*SIN(i,5,m) ,
       COS(i,3,m)*COS(i,5,m)+SIN(i,3,m)*SIN(i,4,m)*SIN(i,5,m) ,
       SIN(i,3,m)*COS(i,4,m),
       COS(i,3,m)*SIN(i,4,m)*COS(i,5,m)+SIN(i,3,m)*SIN(i,5,m) ,
       COS(i,3,m)*SIN(i,4,m)*SIN(i,5,m)-SIN(i,3,m)*COS(i,5,m) ,
       COS(i,3,m)*COS(i,4,m));
}
```

5.3.15 Function point generation

```
void pointGen(word fName)
{
  label po =0;
  label poA=0;
  //if ( fName == "VerticeS" )
  //{
    //Info << "PTB info: plane settings, ";
    //for (label i; i < planeS.size() ; i++) {Info << planeS[i] << " , ";}
    //Info << nl;
  //}
  if ( fName == "VerticeS" )
  {
    Info << "PTB info: no.: specification of plane"
          << "\t(plane)\t\t:(transformation tensor)\t :(M vector):(T vector)" << nl
   // LOGPTB << "No. & specification of plane"
   // << "\t(plane)\t\t:(transformation tensor)\t :(M vector):(T vector)" << nl;
  }
  for (label i=0 ; i < noPlanes; i++ )
  {
    tRefGEN(i,1);
    // Info << "PTB info tREF for i (planes): " << i << "\t" << tRef << nl;
    for (label i=0; i < 9; i++){if (abs(tRef[i]) < 1e-12){tRef[i]=0;}}
    T[i] = tRef & point(0,1,0);
    N[i] = tRef & point(0,0,1);
```

```
     if ( fName == "VerticeS" )
     {
      Info << "PTB info : " << i << " : " << planeS[i] << " : "
          << "\t" << planes[i] << "\t" << ":"
          << tRef << "\t" << ":" << N[i] << ":" << T[i] << nl;
     }
     pointSET(i);
     for (label p=0; p < NO; p++ ) // --- main points on plane ---
     {
      points[po] = (tRef & pREF[p]) + point(planes[i][0],planes[i][1],planes[i][2]);
      po++ ;
     }
     for (label a=0; a < NOAUX; a++ ) // --- auxility points on plane ---
     {
     pointsA[poA] = (tRef & pAUX[a]) + point(planes[i][0],planes[i][1],planes[i][2])
      poA++ ;
     }
    }
     label poC=0;
     if ( fName == "elbow" )
     {
      Info << "PTB info : pair " << "\t" << "Middle plane" << "\t\t\t\t" << "(A, B, R) "
             << "\t\t" << "check for elbow" << nl;
     }
     for (label p=0; p < noPairs; p++)
     {
      word Z = "Angles OK, elbow ";
      scalar PA = pairs[p][0];
      scalar PB = pairs[p][1];
      point pointA = point(planes[PA][0],planes[PA][1],planes[PA][2] );
      point pointB = point(planes[PB][0],planes[PB][1],planes[PB][2] );
      point angleA = point(planes[PA][3],planes[PA][4],planes[PA][5] );
      point angleB = point(planes[PB][3],planes[PB][4],planes[PB][5] );
      M[p] = (pointB - pointA);
    // scalar Mm = mag(M[p]);
      scalar BETA = std::acos((M[p] & N[PA])/(mag(M[p])*mag(N[PA])))/PI;
      scalar BETB = std::acos((M[p] & N[PB])/(mag(M[p])*mag(N[PB])))/PI;
      U[p] = angleB - angleA;
      scalar R(0);
      if (BETA != 0){R = mag(M[p])/2/std::sin(PI*BETA);}
      I[p] = point(BETA*2,BETB*2,R);
      point PO = normalised((M[p] ^ N[PA]) ^ M[p]);
      PO = PO * (R*(1-std::cos(PI*BETA)));
```

```
        planesC[p] = tensor(planes[PA][0] + M[p][0]/2 + P0[0],
                            planes[PA][1] + M[p][1]/2 + P0[1],
                            planes[PA][2] + M[p][2]/2 + P0[2],
                            0.5*(planes[PA][3] + planes[PB][3]),
                            0.5*(planes[PA][4] + planes[PB][4]),
                            0.5*(planes[PA][5] + planes[PB][5]),
                            planes[PA][6],
                            planes[PA][7],
                            planes[PA][8]);
        pointSET(PA);
        tRefGEN(p,2);
        if ( fName == "elbow" )
        {
          if (abs(BETA-BETB) > 0.01) Z = "Warning, inconsistent angles, A != B !? ";
          if (BETA == BETB && BETA == 0) Z = "Angles OK, linear ";
          Info << "PTB info : "
          << pairs[p] << "\t" << planesC[p] << "\t" << I[p] << "\t"<< Z << nl;
        }
        for (label i=0; i < NO; i++ ) // --- main points on plane ---
        {
          pointsC[poC] =
          (tRef & pREF[i]) + point(planesC[p][0],planesC[p][1],planesC[p][2]);
          poC++ ;
        }
      }
}
```

5.3.16 Function block existing

```
label checkB(label A)
{ label P=0;
  for (label i=0 ; i < usedPlane.size() ; i++)
  {
    if (usedPlane[i] == A ){P++;}
  }
  return P;
}

void INFO(word WORD)
{
  Info << "PTB info : " << WORD << nl;
}
```

5.3.17 Function vortices generation

```
// ====================== ENTITY GENERATORS ===========

// --- VORTICES GENERATOR ---
void VerticeS(Ostream& os, const dictionary& dict)
{
  INFO("MeshGenerator v.2.4.4, 12/12/2020, elbow bux fixed ");
  fName = __FUNCTION__;
  SwitcheR(dict);
  if (switcher[0] == 0){emptyCode(os, "vertices <OFF>");}
  else
  {
    INFO("vertices <ON>");
    LookuP(dict);
    pointGen(fName);
    forGnuplotLine();
    system("gnuplot -e \"set grid; stats 'logGNU_3' u 1:2 nooutput;\
      set terminal png size 1600,1200;\
      set output 'graph_1.png';blocks = STATS_blocks ;\
      set key autotitle columnheader;\
      set multiplot layout 2,2 columnsfirst;\
      set xlabel 'Z axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_3' i i u 3:2 notitle w lines lw 10;\
      set xlabel 'Z axis'; set ylabel 'X axis';\
      plot for[i=0:blocks-1] 'logGNU_3' i i u 3:1 notitle w lines lw 10;\
      set xlabel 'X axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_3' i i u 1:2 w lines lw 10;\
      set xlabel 'X'; set ylabel 'Z'; set zlabel 'Y';\
      splot for[i=0:blocks-1] 'logGNU_3' i i u 1:3:2 notitle w lines lw 10\" ");

    forGnuplotPoint();
    system("gnuplot -e \"set grid; stats 'logGNU_2' u 1:2 nooutput;\
      set terminal png size 1600,1200;\
      set output 'graph_points.png';blocks = STATS_blocks ;\
      set key autotitle columnheader;\
      set style fill solid 1.00 border lt -1;\
      set multiplot layout 2,2 columnsfirst;\
      set xlabel 'Z axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_2' i i u 3:2 notitle w lines lw 10;\
      set xlabel 'Z axis'; set ylabel 'X axis';\
      plot for[i=0:blocks-1] 'logGNU_2' i i u 3:1 notitle w lines lw 10;\
      set xlabel 'X axis'; set ylabel 'Y axis';\
      plot for[i=0:blocks-1] 'logGNU_2' i i u 1:2 w lines lw 10;\
      set xlabel 'X'; set ylabel 'Z'; set zlabel 'Y';\
      splot for[i=0:blocks-1] 'logGNU_2' i i u 1:3:2 notitle w lines lw 10\" ");
    os << points;
  }
}
```

5.3.18 Function block generation

```
// --- BLOCKS GENERATOR ---
void BlockS(Ostream& os, const dictionary& dict)
{
  SwitcheR(dict);
  fName = __FUNCTION__;
  if (switcher[1] == 0){emptyCode(os, "blocks <OFF>");}
  else
  {
    INFO("block <ON>");
    LookuP(dict);
    pointGen(fName);
    checkDup();
    os << " ( " << nl;
    noBlocks = noPairs * 12;
    tensorField blocks = tensorField(noBlocks, tensor(0));
    pointField distribution = pointField(noBlocks, point(0,0,0));
    pointField cells = pointField(noBlocks, point(0,0,0));
    Info << "PTB info : Pairs : " ;
    for (label p=0; p < noPairs; p++)
    {
      Info << pairs[p] ;
      point point1 =
      point(planes[pairs[p][0]][0],planes[pairs[p][0]][1],planes[pairs[p][0]][2] );
      point point2 =
      point(planes[pairs[p][1]][0],planes[pairs[p][1]][1],planes[pairs[p][1]][2] );
      scalar L = mag( point1 - point2)/scale;
      for (label b= 0 ; b < 12 ; b++)
      {
        label Bl = p * blockOnPlane + b;
        blocks[Bl] = tensor(
            BL[ blockREF[label(blockUsed[b])][0] + pairs[p][0]*NO] ,
            BL[ blockREF[label(blockUsed[b])][1] + pairs[p][0]*NO] ,
            BL[ blockREF[label(blockUsed[b])][2] + pairs[p][0]*NO] ,
            BL[ blockREF[label(blockUsed[b])][3] + pairs[p][0]*NO] ,
            BL[ blockREF[label(blockUsed[b])][0] + pairs[p][1]*NO] ,
            BL[ blockREF[label(blockUsed[b])][1] + pairs[p][1]*NO] ,
            BL[ blockREF[label(blockUsed[b])][2] + pairs[p][1]*NO] ,
            BL[ blockREF[label(blockUsed[b])][3] + pairs[p][1]*NO] ,
             0 );
        if (blockUsed[b] < 4 )
              {distribution[Bl] = point(1, 1 , 1); cells[Bl]
                      point(mesh[0],mesh[0],round(mesh[2]*L*scale+1));}
        else {distribution[Bl] = point(1/mesh[3],1,1); cells[Bl]
                             point(mesh[1],mesh[0],round(mesh[2]*L*scale+1));}
        os << "hex " << blocks[Bl] << " zone_" << pairs[p][2] << "
        " << cells[Bl] << " simpleGrading " << distribution[Bl] << nl;
      }
    }
    Info << nl;
   os << " ) " << nl;
  }
}
```

5.3.19 Function edge generation

```
// --- EDGE GENERATOR ---
void EdgeS(Ostream& os, const dictionary& dict)
{
  os << " ( " << nl;
  fName = __FUNCTION__;
  SwitcheR(dict);
  if (switcher[2] == 0 || switcher[1] == 0 ){INFO("edge circ <OFF>");}
  else
  {
    INFO("edge circ <ON>");
    LookuP(dict);
    pointGen(fName);
    checkDup();
    label posBC=0;
    for (label p=0 ; p < noPlanes ; p++ )
    {
      if (checkP(p) == true)
      {
        for (label o=9 ; o < 17; o++ )
        {
          label A, B;
          A = BL[o+p*17];
          if (o < 16 ) B = BL[o+p*17+1] ;
          if (o == 16 ) B = BL[9+p*17 ] ;
          if ( CheckE(A,B) == true)
          {
            BCa[posBC] = A;
            BCb[posBC] = B;
            posBC++;
            os << "BSpline " << A << " " << B << " ( ";
            Info << "BSpline " << A << " " << B << " ( ";
            for (label s=1; s < 10 ; s++ )
            {
              os << pointsA[72 + (o-9)*9 + s + p*145];
              Info << pointsA[72 + (o-9)*9 + s + p*145];
            }
            os << " ) " << nl;
            Info << " ) " << nl;
          }
        }
      }
    }
   // --- internal circl ----
    for (label p=0 ; p < noPlanes ; p++ ) // p=planes
    {
      if (checkP(p) == true)
      {
        if ( planeS[p]==0 || planeS[p]==1 || planeS[p]==3 || planeS[p]==30 )
        {
          for (label o=1 ; o < 9; o++ ) // o=point pairs 1-8
          {
            label A, B;
            A = BL[o+p*17];
            if (o < 8 ) B = BL[o+p*17+1] ;
            if (o == 8 ) B = BL[1+p*17 ] ;
            if ( CheckE(A,B) == true)
            {
              BCa[posBC] = A;
              BCb[posBC] = B;
              posBC++;
              os << "BSpline " << A << " " << B << " ( ";
              Info << "BSpline " << A << " " << B << " ( ";
              for (label s=2; s < 10 ; s++ ) //s=1-9 point between A-B
              {
                os << pointsA[0 + (o-1)*9 + s + p*145];
                Info << pointsA[0 + (o-1)*9 + s + p*145];
              }
```

```
                os << " ) " << nl;
                Info << " ) " << nl;
              }
            }
          }
        }
      }
    }
    if (switcher[3] == 0){INFO("edge long <OFF>");}
    else
    {
      //emptyCode(os, "edge long <ON>");
      INFO("edge long <ON>");
      LookuP(dict);
      fName = "elbow";
      pointGen(fName);
      for (label p=0 ; p < noPairs ; p++ )
      {
        if (I[p][2] != 0)
        {
          label PA = pairs[p][0];
          label PB = pairs[p][1];
          for (label o=0 ; o < NO; o++ )
          {
            label A = BL[o+PA*17];
            label B = BL[o+PB*17];
            os << "arc " << A << " " << B << " ";
            Info << "arc " << A << " " << B << " ";
            os << pointsC[o + p*17];
            Info << pointsC[o + p*17];
            os << " " << nl;
            Info << " " << nl;
          }
        }
      }
    }
    os << " ) " << nl;
  }
```

5.3.20 Function boundary condition generation

```
  // --- BOUNDARY GENERATOR ---
  void BoundarY(Ostream& os, const dictionary& dict)
```

```
{
  fName = __FUNCTION__;
  SwitcheR(dict);
  if (switcher[4] == 0 || switcher[1] == 0 ){emptyCode(os, "boundary <OFF>");}
  else
  {
    INFO("boundary <ON>");
    LookuP(dict);
    pointGen(fName);
    checkDup();
    os << " ( " << nl;
    Info << "PTB info: B.C. : " ;
    for (label F = 0; F < freePlanes.size(); F++)
    {
      label bcNO = freePlanes[F];
      if ( checkB(bcNO) == 1 )
      {
        Info << "P" << bcNO << " , " ;
        os << " P" << bcNO << " { type patch; faces ( " << nl;
        for (label i =0; i < blockUsed.size() ; i++)
        {
            os << blockREF[blockUsed[i]] + tensor2D(bcNO,bcNO,bcNO,bcNO)*NO
<< nl;
        }
        os << ");} " << nl;
      }
    }
    os << " ) " << nl;
    Info << nl;
  }
}
```

5.3.21 Function merge of pairs

```
// --- MERGE-PAIRS GENERATOR ---
void MergeP(Ostream& os, const dictionary& dict)
{
  SwitcheR(dict);
  if (switcher[5] == 0){emptyCode(os, "merge <OFF>");}
  else
  {
    emptyCode(os, "merge <ON>");
    LookuP(dict);
    }
  }
```

6 Examples

6.1 Pipeline PF2021

6.1.1 Geometry

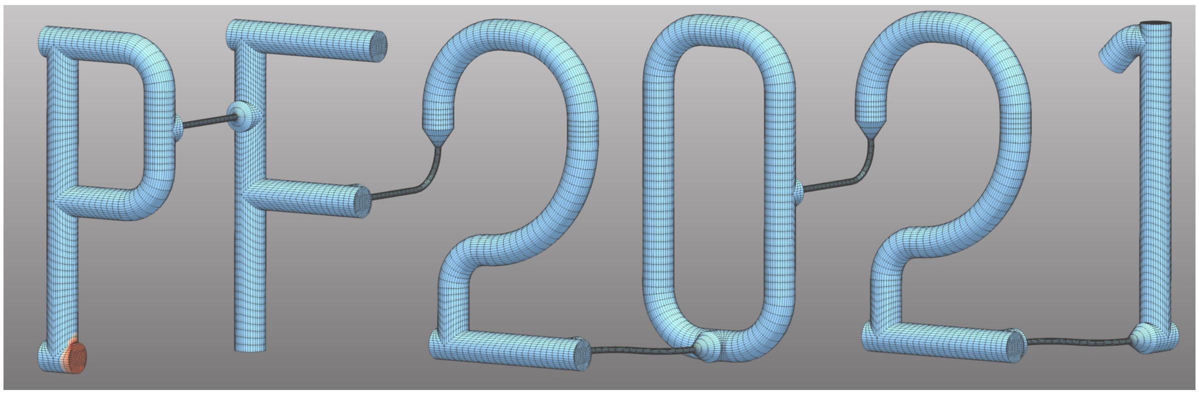


Figure 40: Pipeline example PF2021

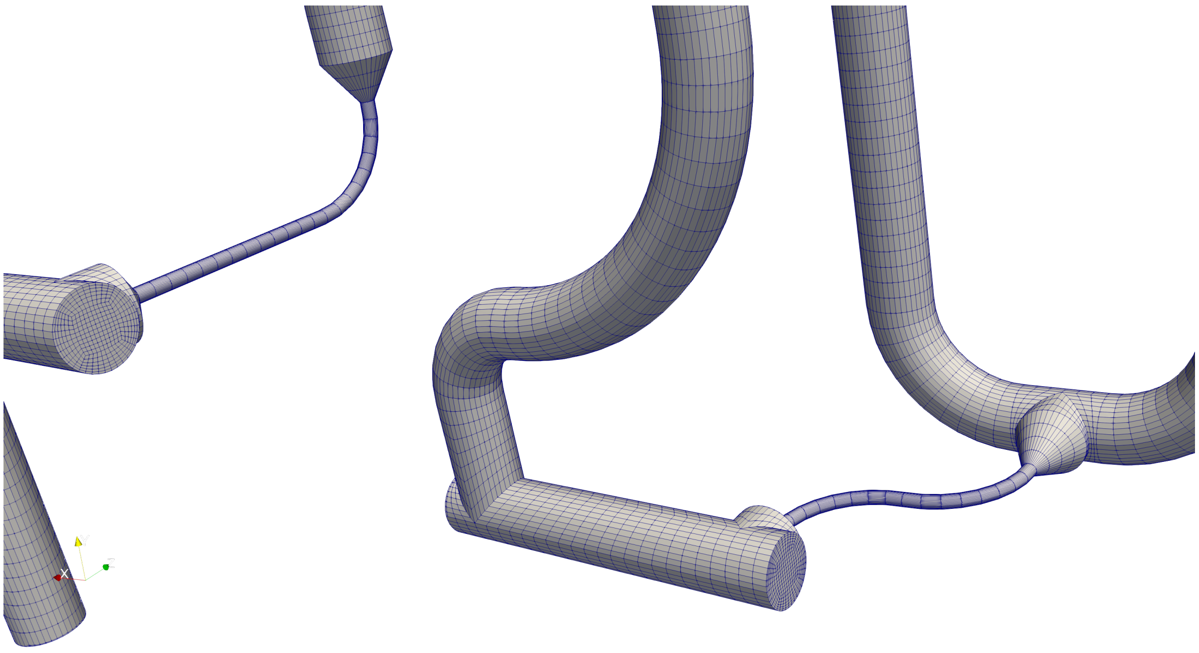


Figure 41: Pipeline detail

Alias for running the meshGenerator of PF2021 is following bash style code:

```
alias PF="blockMesh > logPTB 2>logERR ;\
echo '--- warnings/errors ---'; cat logERR; \
echo '--- PTB actions ---'; cat logPTB | grep 'PTB';\
sed -i'' -e s/'empty'/'wall'/g constant/polyMesh/boundary;\
sed -i'' -e s/'defaultFaces'/'wall'/g constant/polyMesh/boundary;\
sed -i'' -e s/'P113'/'outlet'/g constant/polyMesh/boundary;\
sed -i'' -e s/'P0'/'inlet'/g constant/polyMesh/boundary;\
echo '--- CheckMesh ---'; checkMesh | tail -n 4 "
```

6.1.2 File blockMeshDict

```

//OF header

FoamFile
{
        version 2.0;
        format ascii;
        class dictionary;
        object blockMeshDict;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

// --------- User define variables ------

Rc 0.25; // curvature radius
Rc2 0.5; // curvature radius
Rc3 0.75; // curvature radius
Rc0 0.35; // curvature radius

z1 0.12;
z1n #calc "-$z1";
z2 0.17;
z0 1;

Z1 0.12; // length of inlet
Z2 #calc "$Z1+$z0";
Z3 #calc "$Z2+$z0";
//Z4 #calc "$Z3+$z0";

Z11 #calc "$Z1+$z1";
Z12 #calc "$Z1+$z2";
```

```
Z13 #calc "$Z2-$z2";
Z14 #calc "$Z2-$z1";
Z21 #calc "$Z2+$z1";
Z22 #calc "$Z2+$z2";
Z23 #calc "$Z3-$Rc";
Z31 #calc "$Z3+$z1";
Z32 #calc "$Z3+$z2";
Z33 #calc "$Z32+$Rc";
//Z33 #calc "$Z4-$z2";
Z34 #calc "$Z33+$Rc";
Z35 #calc "$Z34+$z2";
Z4 #calc "$Z35+$z1";
Z5 #calc "$Z4+$z0";
Z6 #calc "$Z5+$z0";
Z61 #calc "$Z6+$z1";

Z41 #calc "$Z21+2*$z0";
Z42 #calc "$Z22+2*$z0";
Z43 #calc "$Z23+2*$z0";
Z46 #calc "$Z4+$z1";

Z51 #calc "$Z31+2*$z0";
Z52 #calc "$Z32+2*$z0";
Z53 #calc "$Z33+2*$z0";
Z54 #calc "$Z34+2*$z0";
Z55 #calc "$Z35+2*$z0";
Z56 #calc "$Z52+$Rc";

nA -0.5; // -90 degrees
pA 0.5; // 90 degrees

YP1 1;
YP2 2;
YP9 2.2;
YP3 #calc "$YP2-$Rc";
YP30 #calc "$YP2-$Rc0";
YP32 #calc "$YP2-$Rc2";
YP4 #calc "$YP1+$Rc";
YP5 1.5;
YP6 #calc "$YP1+$Rc ";
YP7 #calc "$YP6+$z1 ";
YP33 #calc "$YP7 - $Rc3 ";
YP34 #calc "$YP33 -$Rc";
YP21 #calc "$YP2 - $z1";
```

```
YP22 #calc "$YP21 - $z2*0.2929";

//YP33 #calc "$YP1+$Rc2 ";
//YP34 #calc "$YP1-$Rc2";

XP1 -0.5;
XP2 #calc "$XP1-$Rc";
XP3 -1.5;
XP5 #calc "$XP3 -$z1";
XP4 #calc "$XP5 - $z1";
XP6 #calc "$XP1 + $z1";
XP7 #calc "$XP3 - $Rc2 ";
XP8 #calc "$XP7 - $z1 ";
XP9 #calc "$XP8 - $Rc2 ";
XP10 #calc "$XP8 + $Rc2";
XP33 #calc "$XP9 + $Rc3" ;
XP34 #calc "$XP33 + $Rc" ;
XP22 #calc "$XP2+$z1";
XP35 #calc "$XP9+$z1";
XP91 #calc "$XP9+$z1";
XP92 #calc "$XP91-$Rc";
XP00 #calc "$XP92-$Rc";
XP01 #calc "$XP00+$z1";
XP02 #calc "$XP01+$Rc0";
XP03 #calc "$XP00-$z1";
XP04 #calc "$XP03-$Rc0";

S #calc "$XP04 - $XP3";
XP43 #calc "$XP3 + $S";
XP47 #calc "$XP7 + $S";
XP48 #calc "$XP8 + $S";
XP49 #calc "$XP9 + $S";
XP433 #calc "$XP33 + $S";
XP434 #calc "$XP34 + $S";
XP491 #calc "$XP91 + $S";
XP492 #calc "$XP92 + $S";
XP493 #calc "$XP492 + $z1";
XP400 #calc "$XP00 + $S";
XP435 #calc "$XP434+$z1";
XP492 #calc "$XP491 - $Rc ";
XP493 #calc "$XP492 - $Rc ";
XP494 #calc "$XP493 + $z1 ";
XP495 #calc "$XP494 + $z2*0.7071 ";
N 0.1;
```

```
XP496 #calc "$XP495 + $N";
YP23 #calc "$YP22 - $N";
// ---- additional derived variables -----
R1 1;
R2 0.2;
R4 0.6;
R3 1.4;
G1 0.50;
G2 0.45;

// ======= obligatory variables ===========
PI 3.14159265358979323846;
scale 1;
radiuS 0.1;
```

```
planes (( $z1n 0 $Z1 0 0.5 0 $R1 $G1 $G2 ) // PL-00
( 0 0 $Z1 0.5 0 0 $R1 $G1 $G2 ) // PL-01
( 0 $YP1 $Z1 0.5 0 0 $R1 $G1 $G2 ) // PL-02 T "P3"
( 0 $YP1 $Z1 0.5 0 0 $R1 $G1 $G2 ) // PL-03 T "P3"
( 0 $YP1 $Z1 0.5 1.5 0 $R1 $G1 $G2 ) // PL-04 T "P3"
( 0 $YP2 $Z1 0.5 0 0 $R1 $G1 $G2 ) // PL-05
( $XP1 $YP2 $Z1 0.5 0.5 0 $R1 $G1 $G2 ) // PL-06
( $XP2 $YP3 $Z1 0.5 1 0 $R1 $G1 $G2 ) // PL-07
( $XP2 $YP5 $Z1 0.5 1 0 $R1 $G1 $G2 ) // PL-08 T "PF"
( $XP2 $YP5 $Z1 0.5 1 0 $R1 $G1 $G2 ) // PL-09 T "PF"
( $XP2 $YP5 $Z1 0.0 0 0 $R1 $G1 $G2 ) // PL-10 T "PF"
( $XP2 $YP4 $Z1 0.5 1 0 $R1 $G1 $G2 ) // PL-11
( $XP1 $YP1 $Z1 0.5 1.5 0 $R1 $G1 $G2 ) // PL-12
( $XP2 $YP5 $Z11 0.0 0 0 $R1 $G1 $G2 ) // PL-13
( $XP2 $YP5 $Z14 0.0 0 0 $R1 $G1 $G2 ) // PL-14
( $XP2 $YP5 $Z2 0.0 0 0 $R1 $G1 $G2 ) // PL-15 T "F1"
( $XP2 $YP5 $Z2 0.5 0 0 $R1 $G1 $G2 ) // PL-16 T "F1"
( $XP2 $YP5 $Z2 0.5 0 0 $R1 $G1 $G2 ) // PL-17 T "F1"
( $XP2 $YP1 $Z2 0.5 0 0 $R1 $G1 $G2 ) // PL-18 T "F2"
( $XP2 $YP1 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-19 T "F2"
( $XP2 $YP1 $Z2 0.5 0 0 $R1 $G1 $G2 ) // PL-20 T "F2"
( $XP2 0 $Z2 0.5 0 0 $R1 $G1 $G2 ) // PL-21
( $XP3 $YP1 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-22 T "F3"
( $XP3 $YP1 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-23 T "F3"
( $XP3 $YP1 $Z2 0 0 0 $R1 $G1 $G2 ) // PL-24 T "F3"
( $XP5 $YP1 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-25
( $XP3 $YP1 $Z21 0 0 0 $R1 $G1 $G2 ) // PL-26
( $XP2 $YP2 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-27 T "F4"
( $XP2 $YP2 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-28 T "F4"
( $XP2 $YP2 $Z2 0.5 0 0 $R1 $G1 $G2 ) // PL-29 T "F4"
( $XP22 $YP2 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-30
( $XP2 $YP5 $Z12 0.0 0 0 $R2 $G1 $G2 ) // PL-31 P-F
( $XP2 $YP5 $Z13 0.0 0 0 $R2 $G1 $G2 ) // PL-32 P-F
( $XP4 $YP2 $Z2 0.5 0.5 0 $R1 $G1 $G2 ) // PL-33
( 0 0 $Z1 0 0.5 0 $R1 $G1 $G2 ) // PL-34 inlet T
( 0 0 $Z1 0 0.5 0 $R1 $G1 $G2 ) // PL-35 inlet T
( $z1 0 $Z1 0 0.5 0 $R1 $G1 $G2 ) // PL-36 inlet T
( 0 $YP2 $Z1 0.5 0.5 0 $R1 $G1 $G2 ) // PL-37
( 0 $YP2 $Z1 0.5 0.5 0 $R1 $G1 $G2 ) // PL-38
( $z1 $YP2 $Z1 0.5 0.5 0 $R1 $G1 $G2 ) // PL-39
// ---- 2 ---
( $XP3 $YP1 $Z22 0 0 0 $R2 $G1 $G2 ) // PL-40
( $XP3 $YP1 $Z23 0 0 0 $R2 $G1 $G2 ) // PL-41
( $XP3 $YP6 $Z3 0.5 0 0 $R2 $G1 $G2 ) // PL-42
( $XP3 $YP7 $Z3 0.5 0 0 $R1 $G1 $G2 ) // PL-43
( $XP3 $YP32 $Z3 0.5 0 0 $R1 $G1 $G2 ) // PL-44
( $XP7 $YP2 $Z3 0.5 0.5 0 $R1 $G1 $G2 ) // PL-45
( $XP8 $YP2 $Z3 0.5 0.5 0 $R1 $G1 $G2 ) // PL-46
( $XP9 $YP32 $Z3 0.5 1 0 $R1 $G1 $G2 ) // PL-47
( $XP9 $YP7 $Z3 0.5 1 0 $R1 $G1 $G2 ) // PL-48
( $XP33 $YP33 $Z3 0.5 1.5 0 $R1 $G1 $G2 ) // PL-49 vely oblouk 2
( $XP34 $YP34 $Z3 0.5 1 0 $R1 $G1 $G2 ) // PL-50 maly obl. 2
( $XP34 0 $Z3 0.5 1 0 $R1 $G1 $G2 ) // PL-51 T - 2
( $XP34 0 $Z3 0 0.5 0 $R1 $G1 $G2 ) // PL-52 T - 2
( $XP34 0 $Z3 0 0.5 0 $R1 $G1 $G2 ) // PL-53 T - 2
( -1.5 0 $Z3 0 0.5 0 $R1 $G1 $G2 ) // PL-54
( $XP91 0 $Z3 0 0.5 0 $R1 $G1 $G2 ) // PL-55 T - 2-0
( $XP91 0 $Z3 0 0 0 $R1 $G1 $G2 ) // PL-56 T - 2-0
( $XP91 0 $Z3 0 0.5 0 $R1 $G1 $G2 ) // PL-57 T - 2-0
( $XP9 0 $Z3 0 0.5 0 $R1 $G1 $G2 ) // PL-58
( $XP91 0 $Z31 0 0 0 $R1 $G1 $G2 ) // PL-59
( $XP91 0 $Z32 0 0 0 $R2 $G1 $G2 ) // PL-60
( $XP92 0 $Z33 0 0.5 0 $R2 $G1 $G2 ) // PL-61
( $XP00 0 $Z34 0 0 0 $R2 $G1 $G2 ) // PL-62
( $XP00 0 $Z35 0 0 0 $R1 $G1 $G2 ) // PL-63
( $XP00 0 $Z4 0 0 0 $R1 $G1 $G2 ) // PL-64 T "0"
// numero 0
( $XP00 0 $Z4 0.5 1.5 0 $R1 $G1 $G2 ) // PL-65 T "0"
( $XP00 0 $Z4 0.5 1.5 0 $R1 $G1 $G2 ) // PL-66 T "0"
( $XP01 0 $Z4 0.5 1.5 0 $R1 $G1 $G2 ) // PL-67
( $XP02 $Rc0 $Z4 0.5 2 0 $R1 $G1 $G2 ) // PL-68
( $XP02 $YP30 $Z4 0.5 2 0 $R1 $G1 $G2 ) // PL-68
( $XP01 $YP2 $Z4 0.5 2.5 0 $R1 $G1 $G2 ) // PL-70
( $XP03 $YP2 $Z4 0.5 2.5 0 $R1 $G1 $G2 ) // PL-71
( $XP04 $YP30 $Z4 0.5 3.0 0 $R1 $G1 $G2 ) // PL-72
( $XP04 $YP1 $Z4 0.5 3.0 0 $R1 $G1 $G2 ) // PL-73 T 0->2
( $XP04 $YP1 $Z4 0 0 0 $R1 $G1 $G2 ) // PL-74 T 0->2
( $XP04 $YP1 $Z4 0.5 3.0 0 $R1 $G1 $G2 ) // PL-75 T 0->2
( $XP04 $Rc0 $Z4 0.5 3.0 0 $R1 $G1 $G2 ) // PL-76
( $XP03 0 $Z4 0.5 3.5 0 $R1 $G1 $G2 ) // PL-77
( $XP04 $YP1 $Z46 0 0 0 $R1 $G1 $G2 ) // PL-78 after T 0->2

// ---- second 2 ---

( $XP43 $YP1 $Z42 0 0 0 $R2 $G1 $G2 ) // PL-79
( $XP43 $YP1 $Z43 0 0 0 $R2 $G1 $G2 ) // PL-80
( $XP43 $YP6 $Z5 0.5 0 0 $R2 $G1 $G2 ) // PL-81
( $XP43 $YP7 $Z5 0.5 0 0 $R1 $G1 $G2 ) // PL-82
( $XP43 $YP32 $Z5 0.5 0 0 $R1 $G1 $G2 ) // PL-83
( $XP47 $YP2 $Z5 0.5 0.5 0 $R1 $G1 $G2 ) // PL-84
( $XP48 $YP2 $Z5 0.5 0.5 0 $R1 $G1 $G2 ) // PL-85
( $XP49 $YP32 $Z5 0.5 1 0 $R1 $G1 $G2 ) // PL-86
( $XP49 $YP7 $Z5 0.5 1 0 $R1 $G1 $G2 ) // PL-87
( $XP433 $YP33 $Z5 0.5 1.5 0 $R1 $G1 $G2 ) // PL-88 vely oblouk 2
( $XP434 $YP34 $Z5 0.5 1 0 $R1 $G1 $G2 ) // PL-89 maly obl. 2
( $XP434 0 $Z5 0.5 1 0 $R1 $G1 $G2 ) // PL-90 T - 2
( $XP434 0 $Z5 0 0.5 0 $R1 $G1 $G2 ) // PL-91 T - 2
( $XP434 0 $Z5 0 0.5 0 $R1 $G1 $G2 ) // PL-92 T - 2
( $XP435 0 $Z5 0 0.5 0 $R1 $G1 $G2 ) // PL-93
( $XP491 0 $Z5 0 0.5 0 $R1 $G1 $G2 ) // PL-94 T - 2-0
( $XP491 0 $Z5 0 0 0 $R1 $G1 $G2 ) // PL-95 T - 2-0
( $XP491 0 $Z5 0 0.5 0 $R1 $G1 $G2 ) // PL-96 T - 2-0
( $XP49 0 $Z5 0 0.5 0 $R1 $G1 $G2 ) // PL-97
( $XP491 0 $Z51 0 0 0 $R1 $G1 $G2 ) // PL-98
( $XP491 0 $Z52 0 0 0 $R2 $G1 $G2 ) // PL-99
( $XP492 0 $Z56 0 0.5 0 $R2 $G1 $G2 ) // PL-100

( $XP493 0 $Z54 0 0 0 $R2 $G1 $G2 ) // PL-101

( $XP493 0 $Z55 0 0 0 $R1 $G1 $G2 ) // PL-102
( $XP493 0 $Z6 0 0 0 $R1 $G1 $G2 ) // PL-103T "0"
( $XP493 0 $Z6 0.5 0 0 $R1 $G1 $G2 ) // PL-104T "0"
( $XP493 0 $Z6 0 0 0 $R1 $G1 $G2 ) // PL-105T "0"
( $XP493 0 $Z61 0 0 0 $R1 $G1 $G2 ) // PL-106 inlet
( $XP493 $YP21 $Z6 0.5 0 0 $R1 $G1 $G2 ) // PL-107 T 1 no. 1 top
( $XP493 $YP21 $Z6 0.5 0.5 0 $R1 $G1 $G2 ) // PL-108 T 1 no. 1 top
( $XP493 $YP21 $Z6 0.5 0 0 $R1 $G1 $G2 ) // PL-109 T 1 no. 1 top
( $XP493 $YP2 $Z6 0.5 0 0 $R1 $G1 $G2 ) // PL-110 no. 1 top
( $XP494 $YP21 $Z6 0.5 0.5 0 $R1 $G1 $G2 ) // PL-111 no. 1 top
( $XP495 $YP22 $Z6 0.5 0.25 0 $R1 $G1 $G2 ) // PL-112 no. 1 top
( $XP496 $YP23 $Z6 0.5 0.25 0 $R1 $G1 $G2 ) // PL-113 inlet
);
```

Plane specification for T-junctions:

```
defaultSystem 30;

planeSpec (
(1 46)
(2 32 )
(3 42 )
(4 36 )
(5 36 )
(8 33)
(9 43)
(10 45)
(15 35)
(16 41)
(17 31)
(18 42)
(19 46)
(20 32)
(22 33)
(23 43)
(24 46)
(27 32)
(28 42)
(29 36)
(34 41 )
(35 31 )
(37 42)
(38 32)
(51 36)
(52 31)
(53 41)
(55 34)
(56 46)
(57 44)
(64 36)
(65 31)
(66 41)
(73 33)
(74 45)
(75 43)
(90 36)
(91 31)
(92 41)
(94 34)
(95 46)
(96 44)
(103 31)
(104 45)
(105 41)
(107 34)
(108 36)
(109 44)
);
```

Pairs definitions:

```
pairs
(
(34 0 0)
(36 35 0)
(1 2 0)
(3 5 0)
(37 6 0)
(39 38 0)
(6 7 0)
(7 8 0)
(9 11 0)
(11 12 0)
(12 4 0)
(10 13 0)
(13 31 0 )
(31 32 0)
(32 14 0)
(14 15 0)
(18 17 0)
(21 20 0)
(19 22 0)
(23 25 0)
(24 26 0)
(16 29 0)
(30 27 0)
(28 33 0)
// ---- F -> 2 ----
```

```
(26 40 0)
(40 41 0)
(41 42 0)
// ---- 2 ---
(42 43 0)
(43 44 0)
(44 45 0)
(45 46 0)
(46 47 0)
(47 48 0)
(48 49 0)
(49 50 0)
(50 51 0)
(53 55 0)
(54 52 0)
(57 58 0)
(56 59 0)
(59 60 0)
(60 61 0)
(61 62 0)
(62 63 0)
(63 64 0)
(66 67 0)
(67 68 0)
(68 69 0)
(69 70 0)
(70 71 0)
(71 72 0)
(72 73 0)
(75 76 0)
(76 77 0)
(77 65 0) //close "0"
(74 78 0)
(78 79 0)
(79 80 0)
(80 81 0)
(81 82 0)
(82 83 0)
(83 84 0)
(84 85 0)
(85 86 0)
(86 87 0)
(87 88 0)
(88 89 0)
```

```
(89 90 0)
(93 91 0)
(92 94 0)
(96 97 0)
(95 98 0)
(98 99 0)

(99 100 0)
(100 101 0)
(101 102 0)
(102 103 0)
(105 106 0 )
(104 107 0)
(109 110 0)
(111 108 0)
(112 111 0)
(113 112 0)
);
```

```
freePlanes ( 0 113);

// ---- filter -----
blockUsed ( );
planeUsed ( );

// ---- MESH ----
mesh (5 6 20 2);

// ====== switch ======
switcher ( 1 // vortices
1 // blocks
1 // edges circumferential
1 // edges longitudial
1 // boundary
0 // mergePatchPairs
); // default (1 1 1 1 1 1)

// ---- codeStream for pipeline Mesh generator, v.2.4, 29.11.2020 ---
#include "codeStreamPTB";
```

References

[1] OpenFOAM User's guide, ESI (2020)