

Vamos a simular el flujo en el interior de una tubería con simpleFoam. La tubería será tridimensional, y la haremos con Salome-MECA. La malla se generará con snappyHexMesh.

En el escritorio (o donde queramos), creamos una carpeta que llamamos “pipeSnappyHexMesh”. Copiamos las carpetas “0”, “system” y “constant” de la carpeta

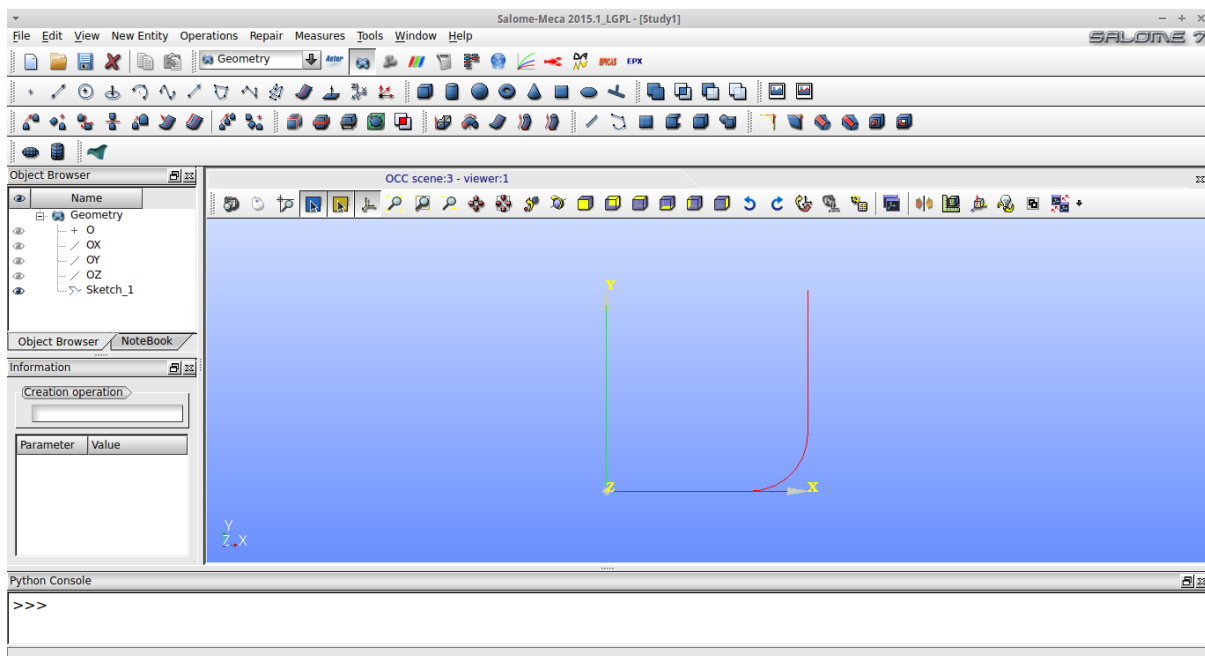
```
/opt/openfoam231/tutorials/incompressible/simpleFoam/pitzDaily
```

a nuestra carpeta creada para la tubería.

Abrimos el Salome-MECA y activamos el módulo de geometría.

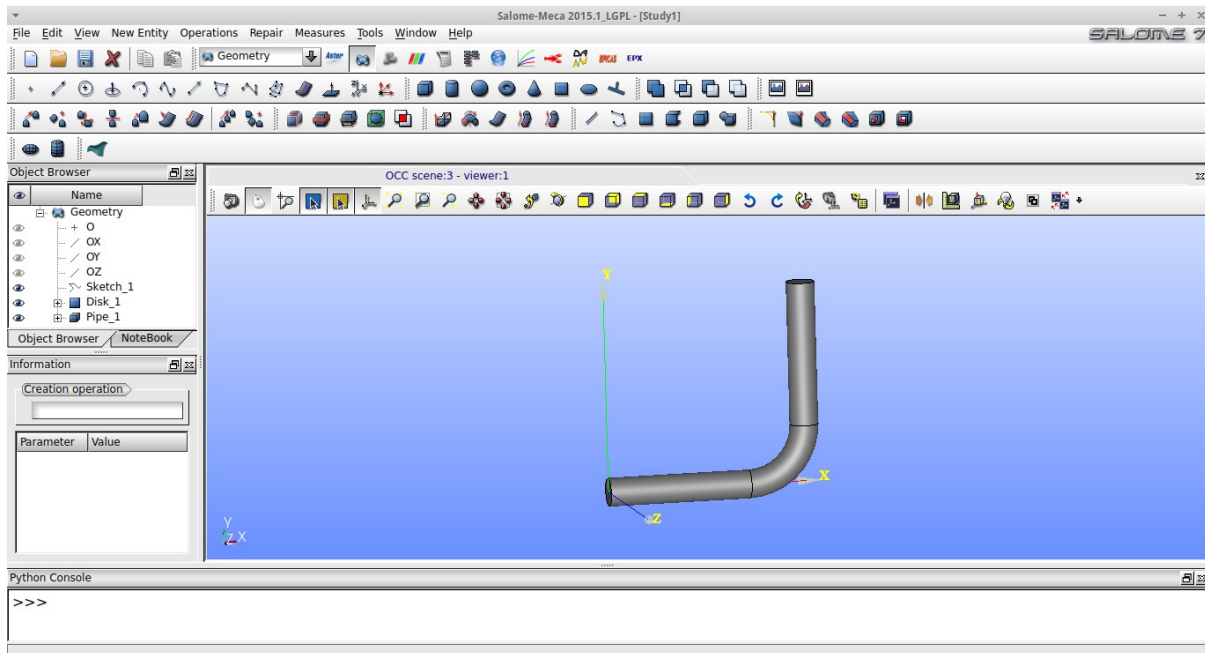
Hacemos un “2D sketch” con el menú New Entity -> Basic

Primero hacemos una recta del punto (0,0) (“apply”) al punto (5,0) (“apply”). Luego activamos el botón de arco, “direction”, “tangent” y radio “2”, ángulo “90”. Finalmente, de nuevo, el botón de segmento, “direction”, “tangent”, y una distancia de 5 metros más, y “close” (recordemos de hacer “apply” cada vez...). Obtendremos algo así:



Creamos un disco, de radio 0.5 metros, centrado en (0,0,0) y normal al eje X. New Entity - Primitives -> Disk

Extruimos el disco siguiendo el sketch creado. New Entity -> Generation -> Extrusion along path  
Ya tenemos la tubería creada:

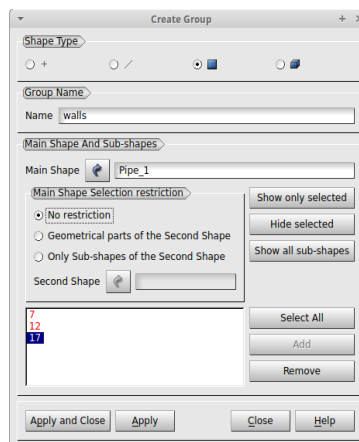


Hacemos un “explode” para dividirla en las diferentes caras que la componen. New Entity -> Explode (argumento Face).

Nos crea 5 caras nuevas.

La “Face\_1” posiblemente será nuestro disco inicial. Los renombramos como “inlet”. Buscamos la cara del final, y la renombramos como “outlet”.

Las otras 3 caras las englobamos en un grupo que llamaremos “walls”. New Entity -> Group -> Create group. Seleccionamos la figura de caras, para señalar que haremos un grupo de las mismas. Lo llamamos “walls” y como figura geometrica decimos el “Pipe\_1”. Luego vamos escogiendo las caras que queremos en el grupo, mientras le damos al boton “Add”.

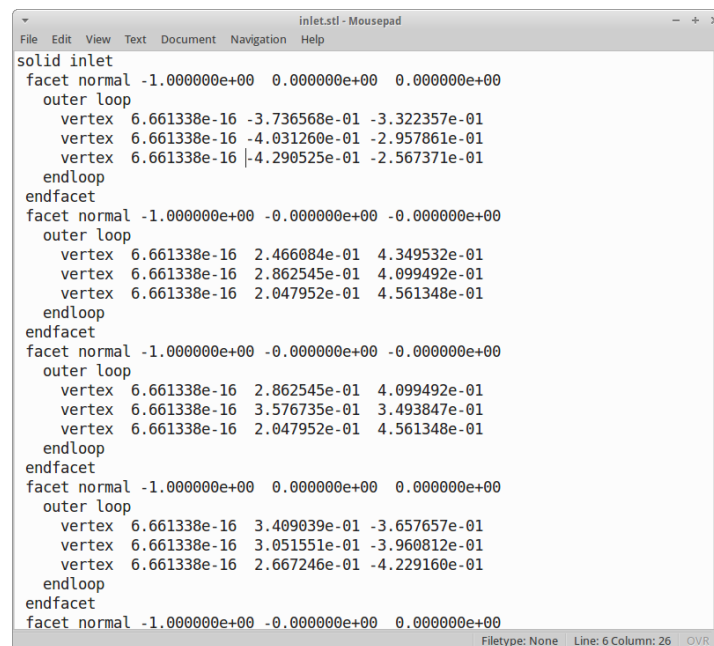


Por último, seleccionando la cara “inlet”, hacemos File -> Export -> STL, y, en el escritorio, lo salvamos como “inlet.stl”. Y hacemos igual con “outlet” y con el grupo “walls”.. Podemos ahora grabar nuestra sesion con Salome y salir.

Movemos estos tres ficheros STL (y el pipe.hdf del slaome también, si queremos) a una carpeta “triSurface” que crearemos dentro de la carpeta “constant” en nuestro caso.

Podemos visualizar las superficies STL creadas con el paraview, por ejemplo.

Ahora vamos a juntarlas en una sola superficie. Editamos el pipe.stl (son texto ASCII) y al final de la primera y la ultima linea, después de “solid” y “endsolid” ponemos “inlet”



```
solid inlet
facet normal -1.000000e+00 0.000000e+00 0.000000e+00
  outer loop
    vertex 6.661338e-16 -3.736568e-01 -3.322357e-01
    vertex 6.661338e-16 -4.031260e-01 -2.957861e-01
    vertex 6.661338e-16 -4.290525e-01 -2.567371e-01
  endloop
endfacet
facet normal -1.000000e+00 -0.000000e+00 -0.000000e+00
  outer loop
    vertex 6.661338e-16 2.466084e-01 4.349532e-01
    vertex 6.661338e-16 2.862545e-01 4.099492e-01
    vertex 6.661338e-16 2.047952e-01 4.561348e-01
  endloop
endfacet
facet normal -1.000000e+00 -0.000000e+00 -0.000000e+00
  outer loop
    vertex 6.661338e-16 2.862545e-01 4.099492e-01
    vertex 6.661338e-16 3.576735e-01 3.493847e-01
    vertex 6.661338e-16 2.047952e-01 4.561348e-01
  endloop
endfacet
facet normal -1.000000e+00 0.000000e+00 0.000000e+00
  outer loop
    vertex 6.661338e-16 3.409039e-01 -3.657657e-01
    vertex 6.661338e-16 3.051551e-01 -3.960812e-01
    vertex 6.661338e-16 2.667246e-01 -4.229160e-01
  endloop
endfacet
endsolid inlet
```

Ahora las convertimos a un formato más práctico para OF, y que ocupa menos espacio de disco:

```
surfaceConvert -clean inlet.stl inlet.obj
surfaceConvert -clean outlet.stl outlet.obj
surfaceConvert -clean walls.stl walls.obj
```

Y, por último, las juntamos en una sola superficie, con caras “etiquetadas”.

```
surfaceAdd inlet.obj outlet.obj inletOutlet.obj
surfaceAdd walls.obj inletOutlet.obj pipe.obj
```

Y borramos los pasos intermedios

```
rm *.stl inletOutlet.obj
```

## Mallado

Vamos a mallar con snappyHexMesh.

Primero tenemos que hacer una malla “background” con blockMesh. Borramos el “blockMeshDict” de constant/polyMesh, que es demasiado complicado, y copiamos el de cavity para adaptarlo hasta que quede así:

```
/*-----* C++ *-----*/
|=====| F ield      | OpenFOAM: The Open Source CFD Toolbox
|  \ \  / | O peration | Version:  2.3.1
|  \ \  / | A nd       | Web:      www.OpenFOAM.org
|  \ \  / | M anipulation|
|=====|
/*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       blockMeshDict;
}
// *****

convertToMeters 1;

vertices
(
    (-1 -1 -1)
    (9 -1 -1)
    (9 9 -1)
    (-1 9 -1)
    (-1 -1 1)
    (9 -1 1)
    (9 9 1)
    (-1 9 1)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (30 30 6) simpleGrading (1 1 1)
);

edges
(
);

boundary
(
    defaultFaces
    {
        type wall;
        faces
        (
            (3 7 6 2)
            (0 4 7 3)
            (2 6 5 1)
            (1 5 4 0)
            (0 3 2 1)
            (4 5 6 7)
        );
    }
);
```

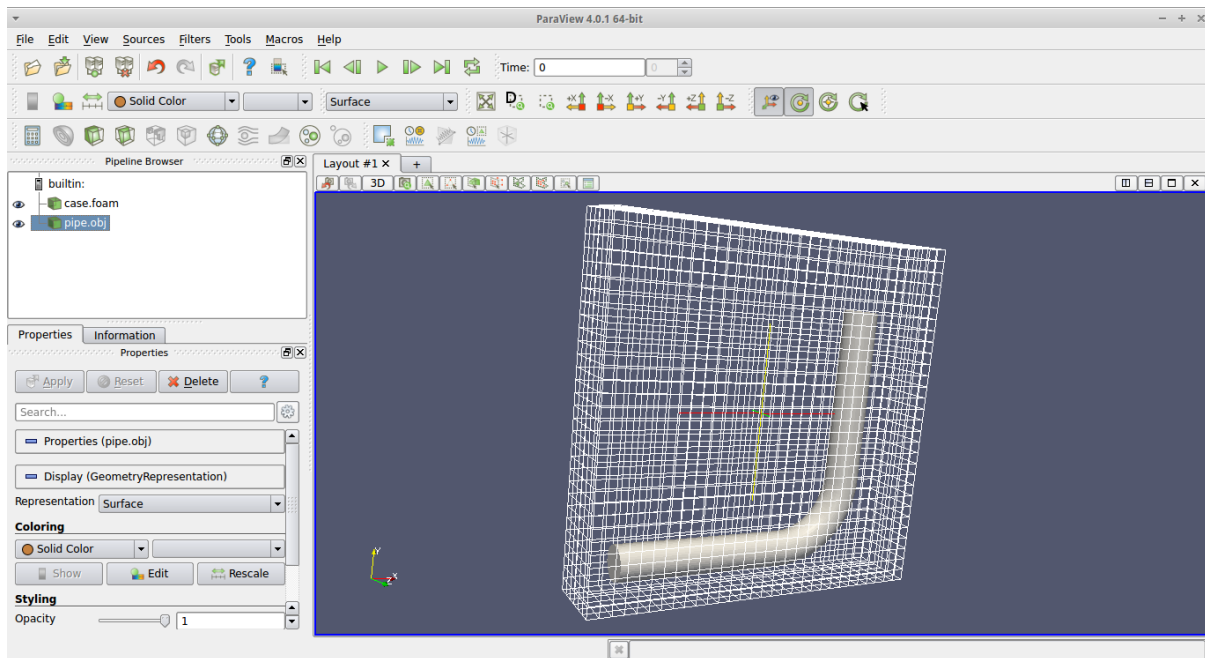
```
);

mergePatchPairs
(
);

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

Y hacemos la malla, comprobando que nuestra tubería se encuentra dentro en su totalidad.

```
blockMesh
paraFoam
```



Ahora vamos a hacer la malla.

Copiamos los ficheros snappyHexMeshDict y meshQualityDict de la carpeta

```
/
opt/openfoam231/applications/utilities/mesh/generation/snappyHexMesh/snappyHex
MeshDict/
```

en la carpeta “system” de nuestro caso, y lo editamos el snappyHexMeshDict adaptándolo a nuestra geometría:

```
/*-----* C++ *-----*/
=====
\ \ \ F i e l d
\ \ \ O p e r a t i o n
\ \ \ A n d
\ \ \ M a n i p u l a t i o n
\ \ \
/*-----*

FoamFile
{
    version      2.0;
    format       ascii;
```

```

        class      dictionary;
        object      snappyHexMeshDict;
    }
// * * * * *

// Which of the steps to run
castellatedMesh true;
snap            true;
addLayers       false;

//Optional: single region surfaces get patch names according to
//            surface only. Multi-region surfaces get patch name
//            surface " "region. Default is true
//singleRegionName false;

// Geometry. Definition of all surfaces. All surfaces are of class
// searchableSurface.
// Surfaces are used
// - to specify refinement for any mesh cell intersecting it
// - to specify refinement for any mesh cell inside/outside/near
// - to 'snap' the mesh boundary to the surface
geometry
{
/*    box1x1x1
    {
        type searchableBox;
        min (1.5 1 -0.5);
        max (3.5 2 0.5);
    }*/

    pipe.obj
    {
        type triSurfaceMesh;

        //tolerance 1E-5; // optional:non-default tolerance on intersections
        //maxTreeDepth 10; // optional:depth of octree. Decrease only in case
                        // of memory limitations.

        // Per region the patchname. If not provided will be <surface>_<region>.
        // Note: this name cannot be used to identity this region in any
        //       other part of this dictionary; it is only a name
        //       for the combination of surface+region (which is only used
        //       when creating patches)
        regions
        {
            inlet
            {
                name inlet;
            }
            outlet
            {
                name outlet;
            }
            walls
            {
                name walls;
            }
        }
    }
}

/*    sphere2
    {

```

```

        type searchableSphere;
        centre (1.5 1.5 1.5);
        radius 1.03;
    }*/
};

// Settings for the castellatedMesh generation.
castellatedMeshControls
{
    // Refinement parameters
    // ~~~~~

    // If local number of cells is >= maxLocalCells on any processor
    // switches from refinement followed by balancing
    // (current method) to (weighted) balancing before refinement.
    maxLocalCells 100000;

    // Overall cell limit (approximately). Refinement will stop immediately
    // upon reaching this number so a refinement level might not complete.
    // Note that this is the number of cells before removing the part which
    // is not 'visible' from the keepPoint. The final number of cells might
    // actually be a lot less.
    maxGlobalCells 2000000;

    // The surface refinement loop might spend lots of iterations refining just a
    // few cells. This setting will cause refinement to stop if <= minimumRefine
    // are selected for refinement. Note: it will at least do one iteration
    // (unless the number of cells to refine is 0)
    minRefinementCells 0;

    // Allow a certain level of imbalance during refining
    // (since balancing is quite expensive)
    // Expressed as fraction of perfect balance (= overall number of cells /
    // nProcs). 0=balance always.
    maxLoadUnbalance 0.10;

    // Number of buffer layers between different levels.
    // 1 means normal 2:1 refinement restriction, larger means slower
    // refinement.
    nCellsBetweenLevels 1;

    // Explicit feature edge refinement
    // ~~~~~

    // Specifies a level for any cell intersected by explicitly provided
    // edges.
    // This is a featureEdgeMesh, read from constant/triSurface for now.
    // Specify 'levels' in the same way as the 'distance' mode in the
    // refinementRegions (see below). The old specification
    //     level 2;
    // is equivalent to
    //     levels ((0 2));

    features
    (
        //{
        //     file "someLine.eMesh";
        //     //level 2;
        //     levels ((0.0 2) (1.0 3));
        //}
    );
};

```

```

// Surface based refinement
// ~~~~~

// Specifies two levels for every surface. The first is the minimum level,
// every cell intersecting a surface gets refined up to the minimum level.
// The second level is the maximum level. Cells that 'see' multiple
// intersections where the intersections make an
// angle > resolveFeatureAngle get refined up to the maximum level.

refinementSurfaces
{
    pipe.obj
    {
        // Surface-wise min and max refinement level
        level (2 2);

        // Optional region-wise level specification
        regions
        {
            inlet
            {
                level (3 3);
            }
            outlet
            {
                level (3 3);
            }
        }

        // Optional specification of patch type (default is wall). No
        // constraint types (cyclic, symmetry) etc. are allowed.
        patchInfo
        {
            type patch;
            inGroups (meshedPatches);
        }

        //- Optional increment (on top of max level) in small gaps
        //gapLevelIncrement 2;

        //- Optional angle to detect small-large cell situation
        // perpendicular to the surface. Is the angle of face w.r.t.
        // the local surface normal. Use on flat(ish) surfaces only.
        // Otherwise leave out or set to negative number.
        //perpendicularAngle 10;

        //- Optional faceZone and (for closed surface) cellZone with
        // how to select the cells that are in the cellZone
        // (inside / outside / specified insidePoint)
        // The orientation of the faceZone is
        // - if on cellZone(s) : point out of (maximum) cellZone
        // - if freestanding : oriented according to surface

        //faceZone sphere;
        //cellZone sphere;
        //cellZoneInside inside; //outside/insidePoint

        //- Optional specification of what to do with faceZone faces:
        // internal : keep them as internal faces (default)
        // baffle : create baffles from them. This gives more
        // freedom in mesh motion
        // boundary : create free-standing boundary faces (baffles

```



```

        // but without the shared points)
        //faceType baffle;
    }
}

// Feature angle:
// - used if min and max refinement level of a surface differ
// - used if feature snapping (see snapControls below) is used
resolveFeatureAngle 30;

//- Optional increment (on top of max level) in small gaps
//gapLevelIncrement 2;

// Planar angle:
// - used to determine if surface normals
//   are roughly the same or opposite. Used
//   - in proximity refinement
//   - to decide when to merge free-standing baffles
//     (if e.g. running in surfaceSimplify mode set this to 180 to
//     merge all baffles)
//   - in snapping to avoid snapping to nearest on 'wrong' side
//     of thin gap
//
// If not specified same as resolveFeatureAngle
planarAngle 30;

// Region-wise refinement
// ~~~~~

// Specifies refinement level for cells in relation to a surface. One of
// three modes
// - distance. 'levels' specifies per distance to the surface the
//   wanted refinement level. The distances need to be specified in
//   increasing order.
// - inside. 'levels' is only one entry and only the level is used. All
//   cells inside the surface get refined up to the level. The surface
//   needs to be closed for this to be possible.
// - outside. Same but cells outside.

refinementRegions
{
/*
    box1x1x1
    {
        mode inside;
        levels ((1.0 4));
    }*/
    //sphere.stl
    //{
    //    mode distance;
    //    levels ((1.0 5) (2.0 3));
    //}
}

// Mesh selection
// ~~~~~

// After refinement patches get added for all refinementSurfaces and
// all cells intersecting the surfaces get put into these patches. The
// section reachable from the locationInMesh is kept.
// NOTE: This point should never be on a face, always inside a cell, even
// after refinement.
locationInMesh (02.3040 0.0028 0.00043);

```

```

// Whether any faceZones (as specified in the refinementSurfaces)
// are only on the boundary of corresponding cellZones or also allow
// free-standing zone faces. Not used if there are no faceZones.
allowFreeStandingZoneFaces true;

// Optional: do not remove cells likely to give snapping problems
// handleSnapProblems false;

// Optional: switch off topological test for cells to-be-squashed
// and use geometric test instead
//useTopologicalSnapDetection false;
}

// Settings for the snapping.
snapControls
{
    // Number of patch smoothing iterations before finding correspondence
    // to surface
    nSmoothPatch 3;

    // Maximum relative distance for points to be attracted by surface.
    // True distance is this factor times local maximum edge length.
    // Note: changed(corrected) w.r.t 17x! (17x used 2* tolerance)
    tolerance 2.0;

    // Number of mesh displacement relaxation iterations.
    nSolveIter 30;

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 5;

    // Feature snapping

    // Number of feature edge snapping iterations.
    // Leave out altogether to disable.
    nFeatureSnapIter 10;

    // Detect (geometric only) features by sampling the surface
    // (default=false).
    implicitFeatureSnap false;

    // Use castellatedMeshControls::features (default = true)
    explicitFeatureSnap true;

    // Detect features between multiple surfaces
    // (only for explicitFeatureSnap, default = false)
    multiRegionFeatureSnap false;

    // wip: disable snapping to opposite near surfaces (revert to 22x behaviour)
    // detectNearSurfacesSnap false;
}

// Settings for the layer addition.
addLayersControls
{
    // Are the thickness parameters below relative to the undistorted
    // size of the refined cell outside layer (true) or absolute sizes (false).
    relativeSizes true;

    // Layer thickness specification. This can be specified in one of following

```

```

// ways:
// - expansionRatio and finalLayerThickness (cell nearest internal mesh)
// - expansionRatio and firstLayerThickness (cell on surface)
// - overall thickness and firstLayerThickness
// - overall thickness and finalLayerThickness
// - overall thickness and expansionRatio
//
// Note: the mode thus selected is global, i.e. one cannot override the
//       mode on a per-patch basis (only the values can be overridden)

// Expansion factor for layer mesh
expansionRatio 1.0;

// Wanted thickness of the layer furthest away from the wall.
// If relativeSizes this is relative to undistorted size of cell
// outside layer.
finalLayerThickness 0.3;

// Wanted thickness of the layer next to the wall.
// If relativeSizes this is relative to undistorted size of cell
// outside layer.
//firstLayerThickness 0.3;

// Wanted overall thickness of layers.
// If relativeSizes this is relative to undistorted size of cell
// outside layer.
//thickness 0.5

// Minimum overall thickness of total layers. If for any reason layer
// cannot be above minThickness do not add layer.
// If relativeSizes this is relative to undistorted size of cell
// outside layer..
minThickness 0.25;

// Per final patch (so not geometry!) the layer information
// Note: This behaviour changed after 21x. Any non-mentioned patches
//       now slide unless:
//       - nSurfaceLayers is explicitly mentioned to be 0.
//       - angle to nearest surface < slipFeatureAngle (see below)
layers
{
/*
    sphere.stl_firstSolid
    {
        nSurfaceLayers 1;

    }
    maxY
    {
        nSurfaceLayers 1;
        // Per patch layer data
        expansionRatio    1.3;
        finalLayerThickness 0.3;
        minThickness      0.1;
    }

    // Disable any mesh shrinking and layer addition on any point of
    // a patch by setting nSurfaceLayers to 0
    frozenPatches
    {
        nSurfaceLayers 0;
    }
*/
}

```

```

// If points get not extruded do nGrow layers of connected faces that are
// also not grown. This helps convergence of the layer addition process
// close to features.
// Note: changed(corrected) w.r.t 17x! (didn't do anything in 17x)
nGrow 0;

// Advanced settings

// Static analysis of starting mesh

// When not to extrude surface. 0 is flat surface, 90 is when two faces
// are perpendicular
featureAngle 130;

// Stop layer growth on highly warped cells
maxFaceThicknessRatio 0.5;

// Patch displacement

// Number of smoothing iterations of surface normals
nSmoothSurfaceNormals 1;

// Smooth layer thickness over surface patches
nSmoothThickness 10;

// Medial axis analysis

// Angle used to pick up medial axis points
// Note: changed(corrected) w.r.t 17x! 90 degrees corresponds to 130
// in 17x.
minMedialAxisAngle 90;

// Reduce layer growth where ratio thickness to medial
// distance is large
maxThicknessToMedialRatio 0.3;

// Number of smoothing iterations of interior mesh movement direction
nSmoothNormals 3;

// Optional: limit the number of steps walking away from the surface.
// Default is unlimited.
//nMedialAxisIter 10;

// Optional: smooth displacement after medial axis determination.
// default is 0.
//nSmoothDisplacement 90;

// (wip)Optional: do not extrude a point if none of the surrounding points is
// not extruded. Default is false.
//detectExtrusionIsland true;

// Mesh shrinking

// Optional: at non-patched sides allow mesh to slip if extrusion
// direction makes angle larger than slipFeatureAngle. Default is
// 0.5*featureAngle.
slipFeatureAngle 30;

```

```

    // Maximum number of snapping relaxation iterations. Should stop
    // before upon reaching a correct mesh.
    nRelaxIter 5;

    // Create buffer region for new layer terminations
    nBufferCellsNoExtrude 0;

    // Overall max number of layer addition iterations. The mesher will
    // exit if it reaches this number of iterations; possibly with an
    // illegal mesh.
    nLayerIter 50;

    // Max number of iterations after which relaxed meshQuality controls
    // get used. Up to nRelaxedIter it uses the settings in
    // meshQualityControls,
    // after nRelaxedIter it uses the values in
    // meshQualityControls::relaxed.
    nRelaxedIter 20;

    // Additional reporting: if there are just a few faces where there
    // are mesh errors (after adding the layers) print their face centres.
    // This helps in tracking down problematic mesh areas.
    //additionalReporting true;
}

// Generic mesh quality settings. At any undoable phase these determine
// where to undo.
meshQualityControls
{
    // Specify mesh quality constraints in separate dictionary so can
    // be reused (e.g. checkMesh -meshQuality)
    #include "meshQualityDict"

    // Optional : some meshing phases allow usage of relaxed rules.
    // See e.g. addLayersControls::nRelaxedIter.
    relaxed
    {
        // Maximum non-orthogonality allowed. Set to 180 to disable.
        maxNonOrtho 75;
    }

    // Advanced

    // Number of error distribution iterations
    nSmoothScale 4;
    // amount to scale back displacement at error points
    errorReduction 0.75;
}

// Advanced

///// Debug flags
//debugFlags
//(
//    mesh           // write intermediate meshes
//    intersections  // write current mesh intersections as .obj files
//    featureSeeds   // write information about explicit feature edge
//                  // refinement
//    attraction     // write attraction as .obj files
//    layerInfo      // write information about layers
//);
//

```

```

//// Write flags
//writeFlags
//
//    scalarLevels    // write volScalarField with cellLevel for postprocessing
//    layerSets        // write cellSets, faceSets of faces in layer
//    layerFields      // write volScalarField for layer coverage
//);

// Merge tolerance. Is fraction of overall bounding box of initial mesh.
// Note: the write tolerance needs to be higher than this.
mergeTolerance 1e-6;

// ***** //

```

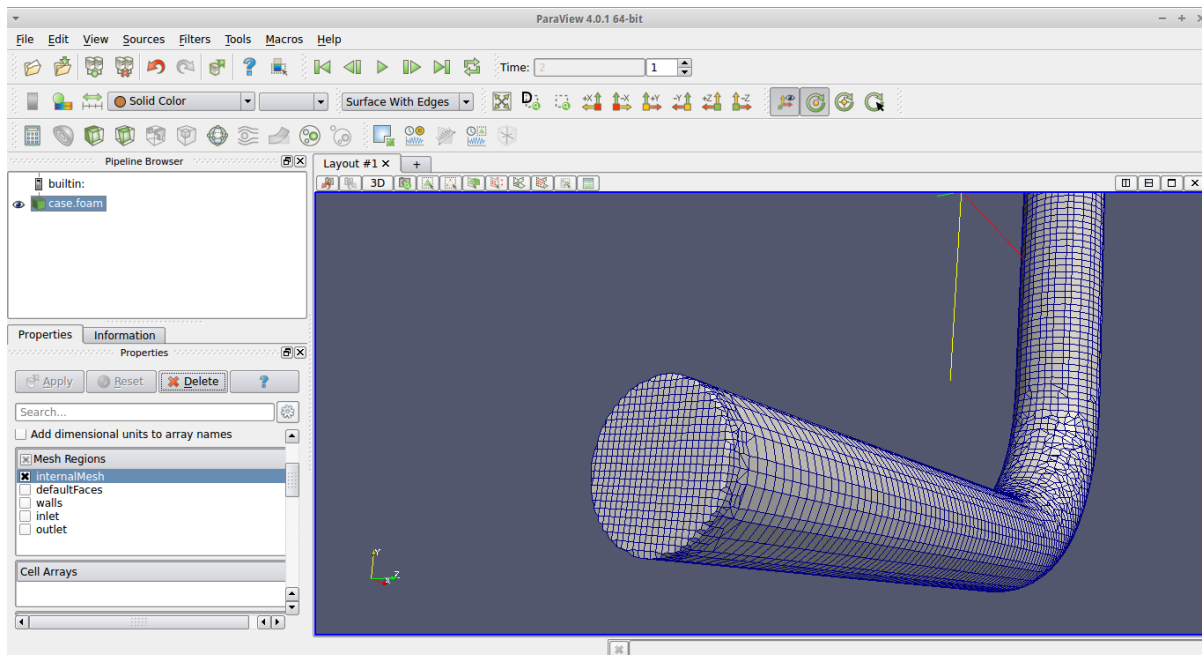
Ha de prestarse especial atención al “locationInMesh”. Determina si queremos mallar dentro (flujo interno) o fuera (flujo externo)

Y ya podeos ejecutar el mallador:

```
snappyHexMesh > log.sHM &
```

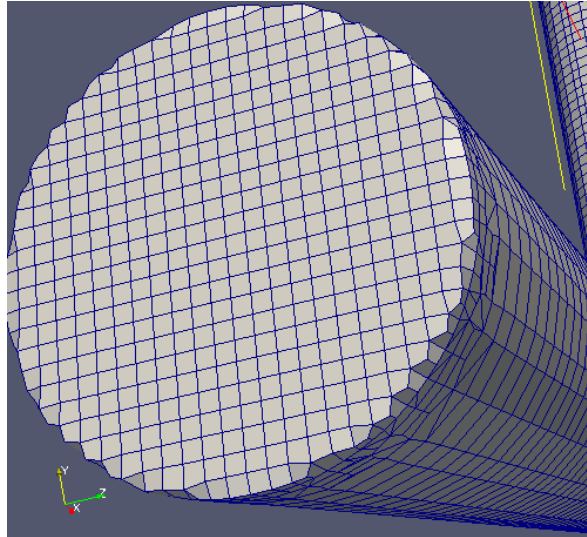
Nos crea dos carpetas, llamadas 1 y 2 (tiempos segun system/controlDict. En cada carpeta hay el resultado parcial de cada fase: castellated, snap y addLayer. En nuestro caso, solo tenemos las dos primeras.

El resultado final:



Si queremos que nos cree una sola malla en constant/polyMesh, sobrescribiendo la creada con blockMesh, debemos ejecutar snappyHexMesh con la opción -overwrite. Lo haremos después. Ahora vamos a fijarnos en la calidad de la malla.

Notamos que las aristas no estan muy bien definidas



Para poder resolver bien las aristas, necesitamos “extraerlas” de la geometria. Esto lo hace una utilidad llamada “surfaceFeatureExtract”. Copiamos el diccionario surfaceFeatureExtractDict, que se encuentra en

/opt/openfoam231/applications/utilities/surface/surfaceFeatureExtract/

en nuestra carpeta system, y hacemos los cambios necesarios.

```
/*-----*- C++ -*-----*\
|=====| F ield      | OpenFOAM: The Open Source CFD Toolbox
| \ \ \ \ | O peration | Version: 2.3.1
| \ \ \ \ | A nd       | Web:      www.OpenFOAM.org
| \ \ \ \ | M anipulation|
|=====|
/*-----*- C++ -*-----*\
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       surfaceFeatureExtractDict;
}
// *****

pipe.obj
{
    // How to obtain raw features (extractFromFile || extractFromSurface)
    extractionMethod    extractFromSurface;

    extractFromSurfaceCoeffs
    {
        // Mark edges whose adjacent surface normals are at an angle less
        // than includedAngle as features
        // - 0 : selects no edges
    }
}
```

```

        // - 180: selects all edges
        includedAngle 120;

        // Do not mark region edges
        geometricTestOnly yes;
    }

    // Write options

        // Write features to obj format for postprocessing
        writeObj yes;
    }

/*surface2.nas
{
    // How to obtain raw features (extractFromFile || extractFromSurface)
    extractionMethod extractFromFile;

    extractFromFileCoeffs
    {
        // Load from an existing feature edge file
        featureEdgeFile "constant/triSurface/featureEdges.nas";
    }

    trimFeatures
    {
        // Remove features with fewer than the specified number of edges
        minElem 0;

        // Remove features shorter than the specified cumulative length
        minLen 0.0;
    }

    subsetFeatures
    {
        // Use a plane to select feature edges
        // (normal)(basePoint)
        // Keep only edges that intersect the plane will be included
        plane (1 0 0)(0 0 0);

        // Select feature edges using a box
        // (minPt)(maxPt)
        // Keep edges inside the box:
        insideBox (0 0 0)(1 1 1);
        // Keep edges outside the box:
        outsideBox (0 0 0)(1 1 1);

        // Keep nonManifold edges (edges with >2 connected faces where
        // the faces form more than two different normal planes)
        nonManifoldEdges yes;

        // Keep open edges (edges with 1 connected face)
        openEdges yes;
    }

    addFeatures
    {

```



```

// Add (without merging) another extendedFeatureEdgeMesh
name          axZ.extendedFeatureEdgeMesh;

// Optionally flip features (invert all normals, making
// convex<->concave etc)
//flip          false;
}

// Output the curvature of the surface
curvature      no;

// Output the proximity of feature points and edges to each other
featureProximity      no;

// The maximum search distance to use when looking for other feature
// points and edges
maxFeatureProximity    1;

// Out put the closeness of surface elements to other surface elements.
closeness              no;

// Write options

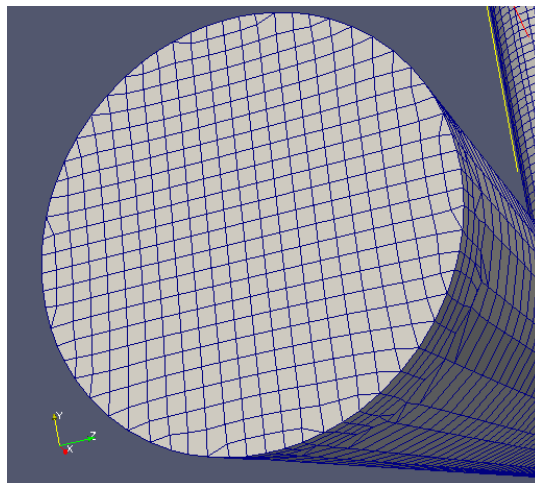
// Write features to obj format for postprocessing
writeObj              yes;

// Write surface proximity and curvature fields to vtk format
// for postprocessing
writeVTK              no;
}*/

// ***** //

```

Borramos los directorios 1 y 2 creados anteriormente y volvemos a ejecutar el “snappy”. Los que obtenemos ahora es mucho mejor...



Por último, hacemos un “checking” general de la malla

checkMesh

y verificamos que está todo en orden.

Al hacer “snappy” es habitual que nos encontremos con que los contornos usados con “blockMesh” existen aun, pero tienen tamaño “0”:

Checking patch topology for multiply connected surfaces...

Patch	Faces	Points	Surface topology
defaultFaces	0	0	ok (empty)
walls	5316	5696	ok (non-closed singly connected)
inlet	448	465	ok (non-closed singly connected)
outlet	448	465	ok (non-closed singly connected)

Si no incluimos este “defaultFaces” en nuestros ficheros de variables, tendremos un error. La otra opción es borrar este patch. Esto se puede hacer a mano, editando el fichero constant/polyMesh/boundary, y borrando la entrada correspondiente. No olvidar cambiar el número que indica la cantidad de contornos en el fichero, de 4 a 3. Y debemos asegurarnos que los contornos que sean “pared” estén definidos como tal.

```
4
(
  defaultFaces
  {
    type          wall;
    inGroups      1(wall);
    nFaces        0;
    startFace     31009;
  }
  walls
  {
    type          wall;
    inGroups      1(meshedPatches);
    nFaces        5316;
    startFace     31009;
  }
  inlet
  {
    type          patch;
    inGroups      1(meshedPatches);
    nFaces        448;
    startFace     36325;
  }
  outlet
  {
    type          patch;
    inGroups      1(meshedPatches);
    nFaces        448;
    startFace     36773;
  }
)
```

Existe un pequeño script escrito en python, e incluido en el paquete pyFoam, que hace esto de forma automática.

```
PyFoamClearEmptyBoundaries.py ./
```

## Preparación de la simulación

El solucionador que vamos a usar es el simpleFoam. Este es un simulador desarrollado para

- flujo estacionario
- flujo incompresible
- fluido newtoniano/ no-newtoniano (fluidos reológicos)
- flujo turbulento

No considera, por tanto, ni compresibilidad ni intercambio de calor. La solución encontrada será una solución estacionaria.

Vamos a preparar las condiciones de contorno en la carpeta 0. En cada uno de los fichero de esta carpeta, cambiamos el nombre del patch “upperWall” a “walls”, y borramos toda la información correspondiente a “lowerWall” y a “frontAndBack”.

Por motivos de seguridad, después de modificar la carpeta 0, hacemos una copia,

```
cp -r 0 0.org
```

Revisamos el diccionario system/controlDict, para asegurarnos que todo esta en orden. Hay definida un funcion “runTime” para calular lineas de corriente mientras se hace la simulación. Nosotros, de momento, omitimos esta función y la comentamos para que no haga el cálculo.

```
/*-----* C++ *-----*/
|=====|
| \ \ \ \ | F ield      | OpenFOAM: The Open Source CFD Toolbox
| \ \ \ \ | O peration  | Version: 2.3.1
| \ \ \ \ | A nd        | Web:      www.OpenFOAM.org
| \ \ \ \ | M anipulation|
|=====|
/*-----*/

FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       controlDict;
}
// *****

application     simpleFoam;

startFrom       latestTime;

startTime       0;
```

```

stopAt      endTime;
endTime     1000;
deltaT      1;
writeControl  timeStep;
writeInterval  50;
purgeWrite   0;
writeFormat   ascii;
writePrecision 6;
writeCompression off;
timeFormat    general;
timePrecision 6;
runTimeModifiable true;
/*functions
{
    streamLines
    {
        type      streamLine;

        // Where to load it from (if not already in solver)
        functionObjectLibs ("libfieldFunctionObjects.so");

        // Output every
        outputControl  outputTime;
        // outputInterval 10;

        setFormat      vtk; //gnuplot;//xmgr;//raw;//jplot;//csv;//ensight;

        // Velocity field to use for tracking.
        UName U;

        // Tracked forwards (+U) or backwards (-U)
        trackForward true;

        // Names of fields to sample. Should contain above velocity field!
        fields (p k U);

        // Steps particles can travel before being removed
        lifeTime 10000;

        // Number of steps per cell (estimate). Set to 1 to disable subcycling.
        nSubCycle 5;

        // Cloud name to use
        cloudName particleTracks;

        // Seeding method. See the sampleSets in sampleDict.
        seedSampleSet uniform; //cloud;//triSurfaceMeshPointSet;

        uniformCoeffs
        {
            type      uniform;
            axis      x; //distance;

```

$$\}^{\ast/}$$

El resto parece ser correcto.

Dado que el proceso de simulación es iterativo, es importante partir de un buen flujo inicial. Si simulamos ahora, partimos de un flujo 0, y conseguimos converger la solución en tan solo 83 iteraciones.

# simpleFoam

Sin embargo, la convergencia puede ser mejor si partimos de una campo inicial mejor. Esto se puede conseguir, por ejemplo, con la solución de un flujo potencial. El solver “potentialFoam” es muy rápido, porque no es iterativo; no resuelve las ecuaciones de Navier-stokes, sino simplemente la ecuación de Laplace, que es lineal. La solución es muy poco realista, pero es un muy buen punto de partida para un solver más complejo, como es simpleFoam.

Si ejecutamos

potentialFoam

nos encontraremos con el siguiente error:

```
--> FOAM FATAL IO ERROR:
keyword potentialFlow is undefined in dictionary
"/home/xubuntu/Desktop/pipeSnappyHexMesh/system/fvSolution"
```

que nos indica que necesita un subdiccionario para el solver potentialFoam en el diccionario fvSolution. Este subdiccionario lo podemos encontrar en cualquier caso con potentialFoam de los tutoriales. P.e., en

```
/opt/openfoam231/tutorials/basic/potentialFoam/pitzDaily/system/fvSolution
```

Lo copiamos y lo incluimos en el nuestro:

```

/*-----*-- C++ --*-----*/
=====
\\      F ield      OpenFOAM: The Open Source CFD Toolbox
\\      O peration   Version:  2.3.1
\\      A nd          Web:      www.OpenFOAM.org
\\      M anipulation

/*-----*-- C++ --*-----*/
FoamFile
{
    version      2.0;

```

```

    format      ascii;
    class       dictionary;
    location    "system";
    object      fvSolution;
}
// * * * * *

```

```

solvers
{
    p
    {
        solver      GAMG;
        tolerance    1e-06;
        relTol       0.1;
        smoother     GaussSeidel;
        nPreSweeps    0;
        nPostSweeps   2;
        cacheAgglomeration on;
        agglomerator   faceAreaPair;
        nCellsInCoarsestLevel 10;
        mergeLevels    1;
    }

    "(U|k|epsilon|R|nuTilda)"
    {
        solver      smoothSolver;
        smoother     symGaussSeidel;
        tolerance    1e-05;
        relTol       0.1;
    }
}

```

```

SIMPLE
{
    nNonOrthogonalCorrectors 0;

    residualControl
    {
        p      1e-2;
        U      1e-3;
        "(k|epsilon|omega)" 1e-3;
    }
}

```

```

potentialFlow
{
    nNonOrthogonalCorrectors 0;
}

```

```

relaxationFactors
{
    fields
    {
        p      0.3;
    }
    equations
    {
        U      0.7;
        k      0.7;
        epsilon 0.7;
        R      0.7;
        nuTilda 0.7;
    }
}

```

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

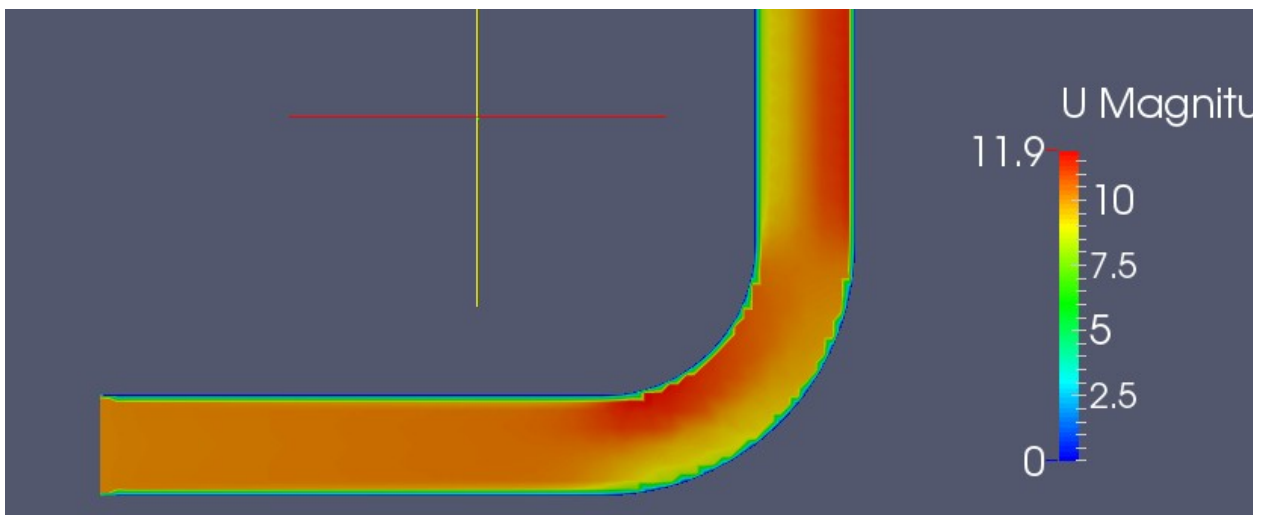
Para entender la importancia del parámetro `nNonOrthogonalCorrectors`, consultamos el documento `nonOrthogonality.pdf`.

Ahora tenemos un valor 0, pero dado que tenemos una non-orthogonality de  $48^\circ$ , vale la pena aumentar a 5 iteraciones.

Una vez hecha la simulación, calculamos el valor de la presión necesaria para obtener este flujo

```
patchAverage p inlet
```

lo cual nos da un valor de 14.3 Pa/Kg en la convergencia.



Observamos 2 cosas en esta figura:

1. El campo de velocidades “refleja” las celdas de la malla. Esto es una indicación de que la malla es demasiado “gruesa”. Hay que refinarla.
2. En el inlet el flujo es uniforme y necesita una cierta longitud para desarrollar una cierta distribución de velocidades más realista.

El segundo punto se puede resolver, aunque no de una forma sencilla, con `swak4foam` (<https://openfoamwiki.net/index.php/Contrib/swak4Foam>)

El primer punto es más sencillo. Tan solo tenemos que copiar el caso, rehacer la malla más fina, interpolar la solución y volver a ejecutar el solver.

```
cd ..
mkdir pipeSnappyHexMesh1
cp -r pipeSnappyHexMesh/0.org pipeSnappyHexMesh/constant/ \
pipeSnappyHexMesh/system/ pipeSnappyHexMesh1
cd pipeSnappyHexMesh1; foamClearPolyMesh; cp -r 0.org 0
```

Ahora tenemos 2 opciones para refinar:

1. Modificar el “blockMeshDict” y dejar intacto el “snappyHexMeshDict”
2. No tocar el “blockMeshDict” y modificar el “snappyHexMeshDict”

Optamos por la segunda opción. Editamos el “snappyHexMeshDict” y aumentamos en 1 los valores del nivel de refinamiento en las superficies y patches, y tambien en el “pipe.eMesh”.

```
blockMesh
snappyHexMesh -overwrite
pyFoamClearEmptyBoundaries.py ./
(remember also to change the type of “walls” from “patch” to “wall”)
mapFields -consistent -sourceTime 'latestTime' ../pipeSnappyHexMesh
simpleFoam
```

Ahora la simulacion es un poco más larga, y converge en la interacion 81. El valor ahora de la presión es de 13,5 Pa/Kg.

Deberiamos seguir relizando este proceso hasta que la presión converja.

Para tener un idea de lo buena que es nuestra malla en las paredes (donde se produce la pérdida de presión), es conviene usar la aplicación

yPlusRAS

que nos da un valor mínimo de 52, que es excesivo. Deberiamos seguir refinando la malla o, mejor aun insertar capas (tercera etapa del “snappy”)