

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

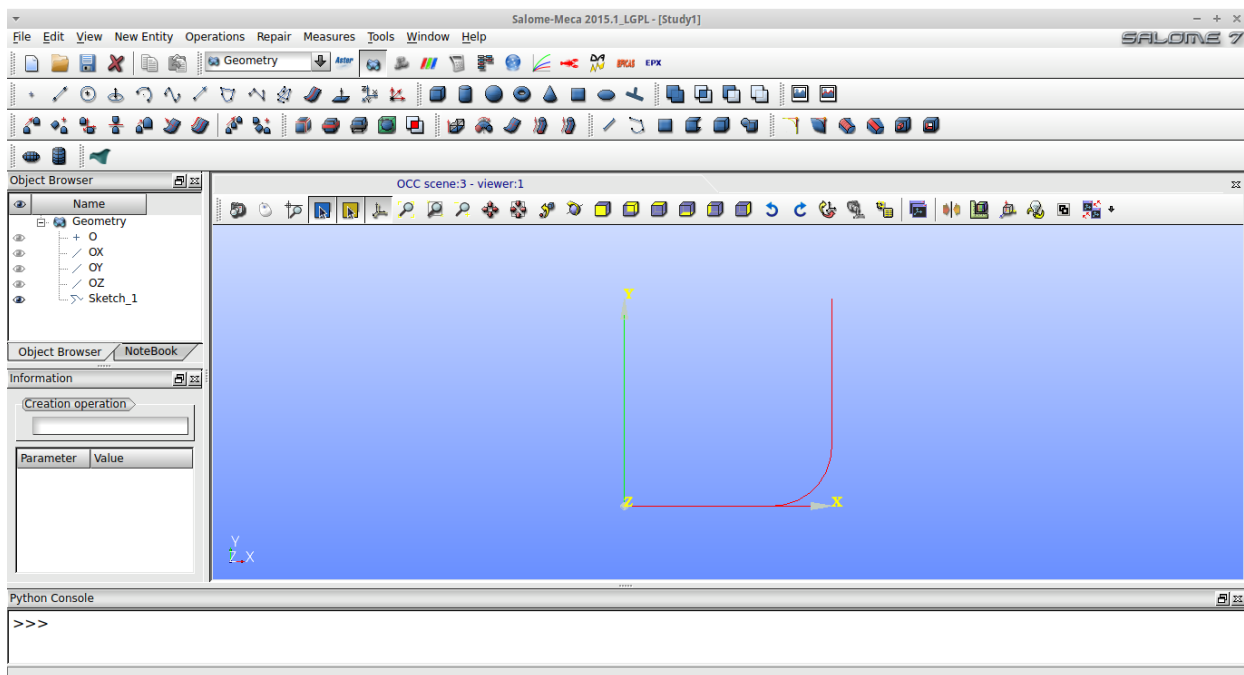
```
/opt/openfoam231/tutorials/incompressible/simpleFoam/pitzDaily
```

a nuestra carpeta creada para la tubería.

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

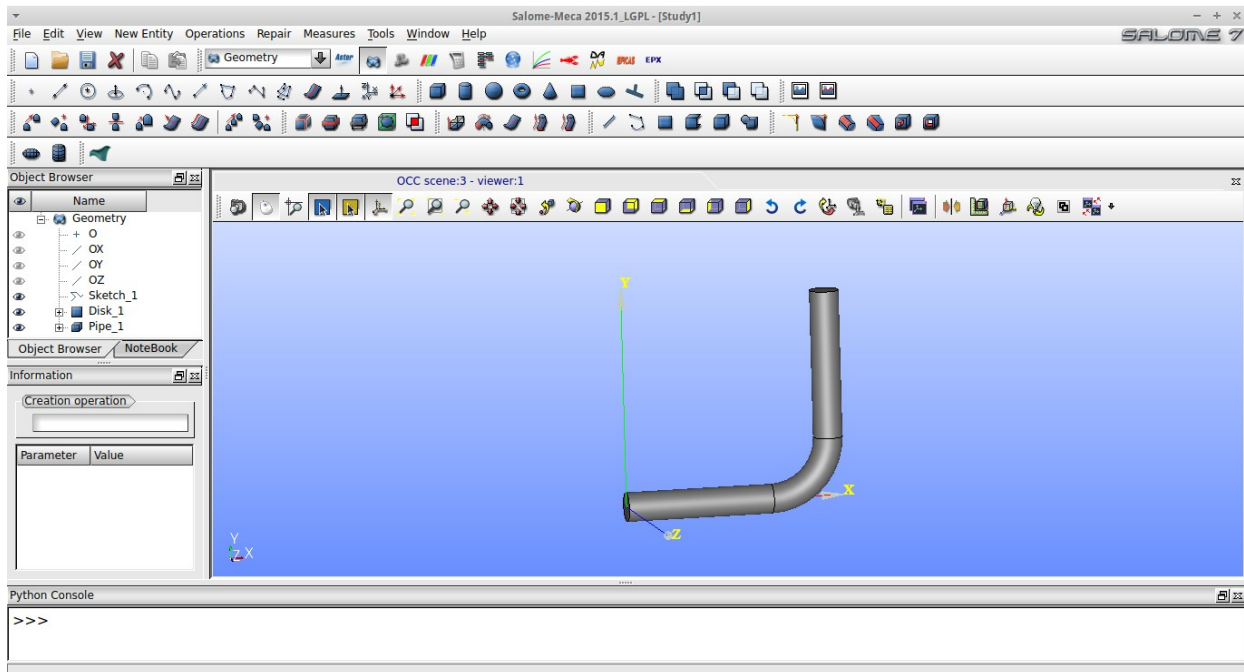
Hacemos un “2D sketch” con el menú New Entity -> Basic

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:

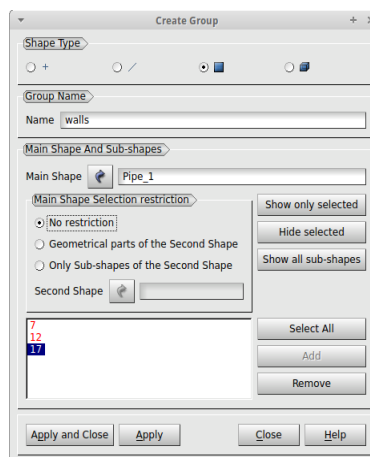


Hacemos un “explode” para dividirla en las diferentes caras que la componen. New Entity -> Explode (argumento Face).

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”.

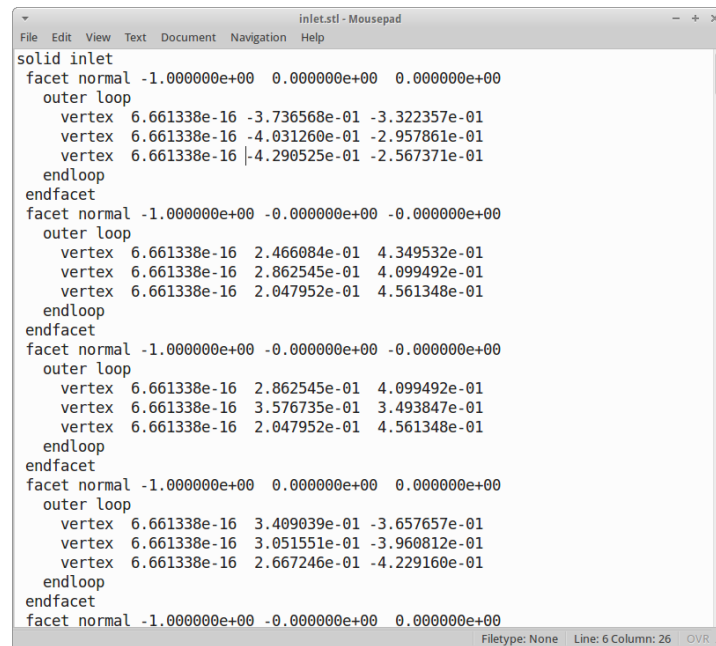


Por último, seleccionando la cara “inlet”, hacemos File -> Export -> STL, y, en el escritorio, lo salvamos como “inlet.stl”. Y hacemos igual con “outlet” y con el grupo “walls”. Podemos ahora grabar nuestra sesión con Salome y salir.

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”



```
inlet.stl - Mousepad
File Edit View Text Document Navigation Help
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
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
Filetype: None Line: 6 Column: 26 OVR
```

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. Borraremos 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|
|=====|
\*-----*- C++ -*-----*/
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)
);

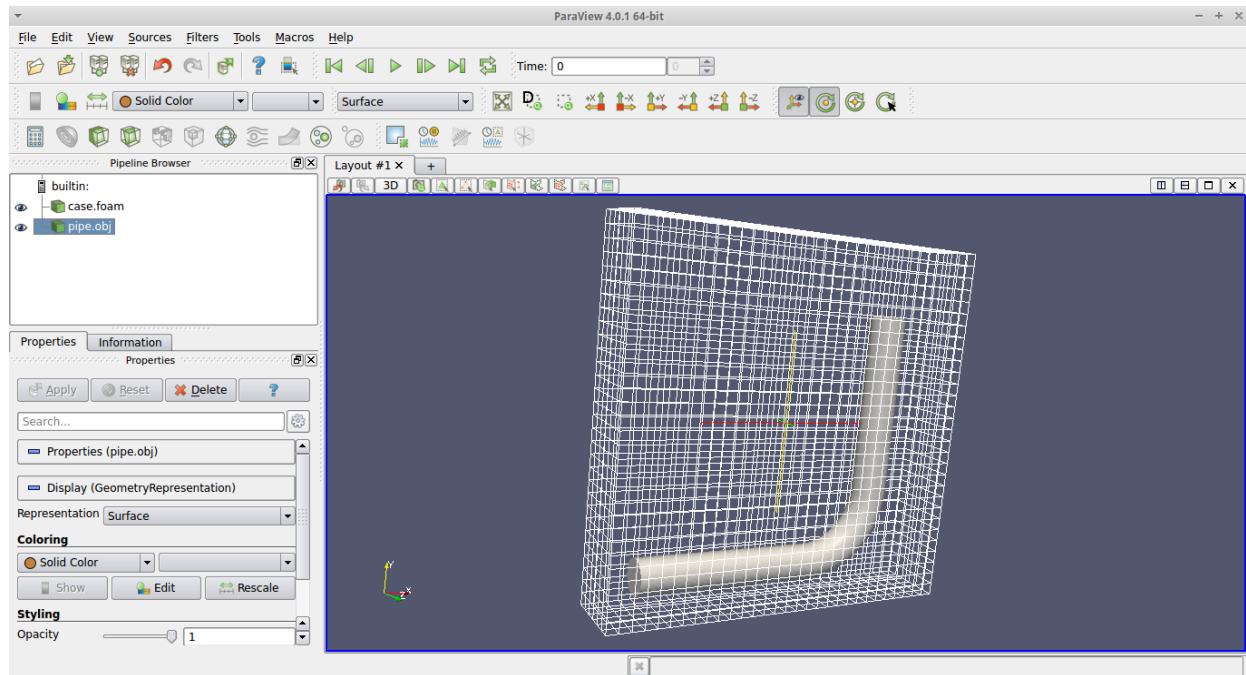
edges
(
);

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 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      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 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;

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

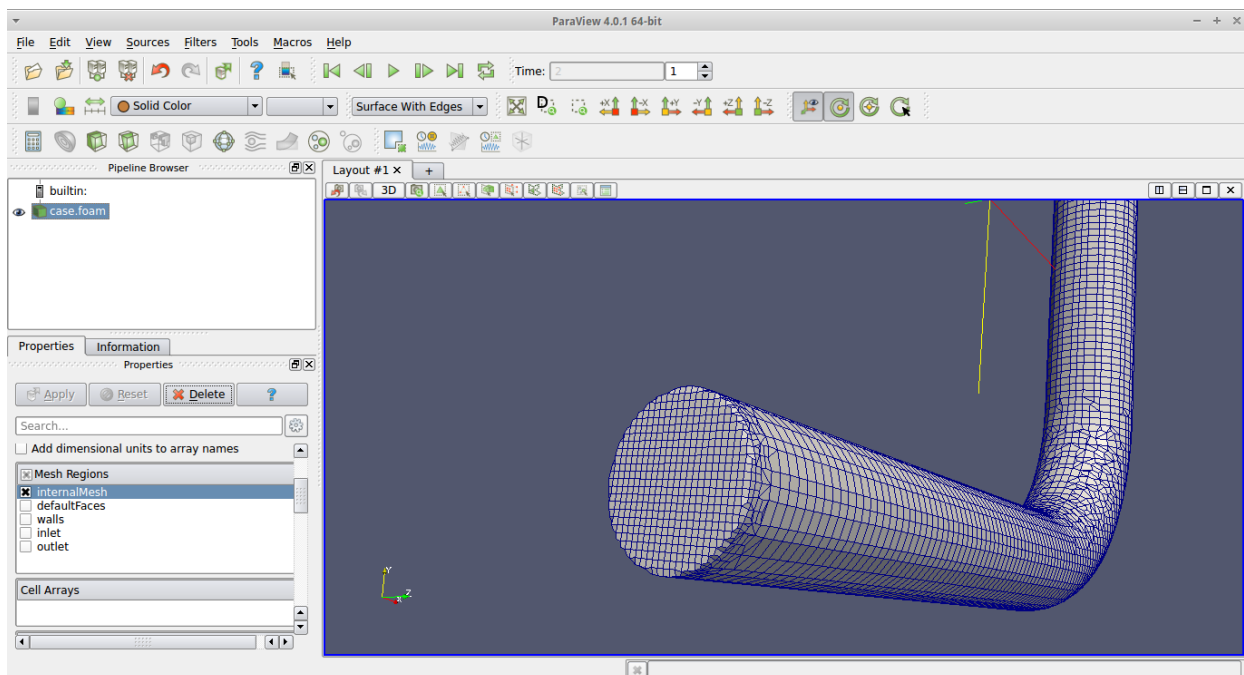
Ha de prestarse especial atención al “locationInMesh”. Determina si queremos mallar dentro (flujo interno) o fuera (flujo externo)

Y ya podeos ejecutar el mallador:

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

Nos crea dos carpetas, llamadas 1 y 2 (tiempos según 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 opcion `-overwrite`