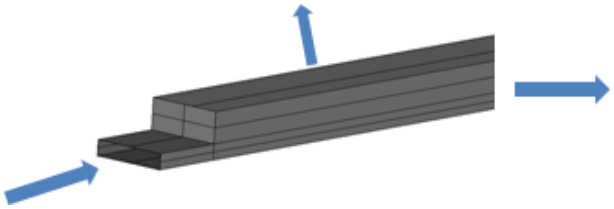
One channel: Flow through a channel and porous layer with one or two outlets

Creating geometry and mesh for an existing OpenFOAM CFD fuel cell model by using the software SALOME

Version 3.0
June 17, 2016



Uwe Reimer
Forschungszentrum Jülich GmbH, Institute of Energy and Climate Research,
Electrochemical Process Engineering, 52425 Jülich/ Germany

JÜLICH
FORSCHUNGSZENTRUM

Contents

1 Introduction **3**

2 The script *OneChannel.py* **4**

2.1 Declaration of input variables 5

2.1.1 Internal parameters 6

2.1.2 Names and lists for GROUP definition 7

2.2 Functions 7

2.2.1 DefBlock 7

2.2.2 CreateCell 7

2.2.3 CreateGroups 8

2.2.4 CreateMesh 10

1 Introduction

The main goal of this documentation is to give an example how to generate a geometry and the corresponding mesh for a single channel to be used in computational fluid dynamics (CFD) simulation. Geometry and meshing tools are usually provided by commercial software packages but lacking in Open Source codes like OpenFOAM[1]. The present example shows how to use another Open Source tool SALOME[2] in order to generate the required mesh.
This example is used for a beginners tutorial for simulations uding OpenFOAM is used[3]. The geometry consists of a single channel with a porous layer on top. The gas enters at the inlet and exits at the outlet and (optional) through a membrane located at the top of the porous transport layer. Additional lengths for inlet and outlet of the channel an be defined as shown in Figure 1.

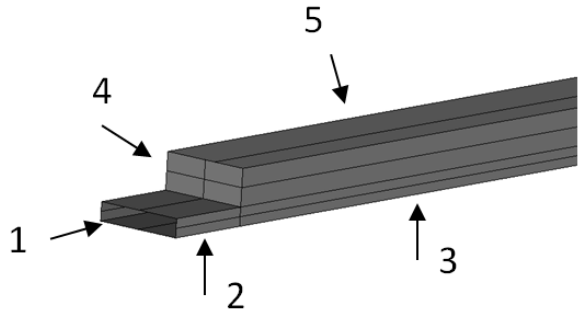


Figure 1: Single channel with boundaries; 1 = inlet, 2 = additinal inlet length, 3 = channel, 4 = poous transport layer, 5 = membrane (second outlet)

Sometimes, an additional requirement in CFD simulation is to systematically vary parameters – as for example to check for grid independence or to search for an optimum in channel height or width. Therefore, an automtated procedure is of advantage that generates the corresponding mesh. One solution is to use the scripting ability of the *Salome* software. The scripting language is PYTHON. The overall process of this example includes the following steps.

1. start the software *Salome*
2. generate geometry and mesh by using the script *OneChannel.py*
3. export the mesh to OpenFOAM

[1] http::://www.openfoam.com
[2] http::://www.salome-platform.org
[3] http://openfuelcell.sourceforge.net/

2 The script *OneChannel.py*

The script generates the geometry based on simple building blocks (cuboids). The PYTHON script generates an internal data structure for the required geometry and creates a 'copy' of it by calling the corresponding access functions of SALOME. The following steps are necessary to run the script. (This script was tested with SALOME 7.4.0.)

1. Open any text editor and adjust the input parameters in 'OneChannel.py'
2. Start SALOME
3. Start a new session ('File' → 'New' or 'CTRL' + 'N')
4. Load the script ('File' → 'Load Script' or 'CTRL' + 'T')

The script runs automatically and the geometry and mesh are generated. If the script is finished it prints ' ... done.' in the text console (located at the bottom of the SALOME windows).
The script consists of the following blocks which are explained in the following sections.

Header information This invokes the SALOME modules for geometry and mesh generation.

Declaration of input variables This contains all parameters which can be adjusted by the user.

Declaration of functions Here the internal functions are defined.

Main area Here the script is actually run. The functions are called step by step in order to generate the geometry, create the surface and volume groups and finally to generate the mesh.

2.1 Declaration of input variables

The input variables are arranged in four blocks. The first two blocks contain parameters to specify geometry and mesh. These are for user interaction. The second two blocks contain internal variables which are inherently linked to the code. These parameters should only be changed by advanced users.

The input parameters are collected within an ordered list. This provides an interface for future versions where this data could be imported form a text file or some other user interface. The geometry parameters are shown below.

```
#######################################
## GEOMETRY
#######################################
myinput.append(0.0)     # origin coordinate x   / 0
myinput.append(0.0) # origin coordinate y       / 1
myinput.append(0.0) # origin coordinate z       / 2

myinput.append(5.0e-3)          # channel width x in m                  / 3
myinput.append(85.0e-3)         # channel length y in m                 / 4
myinput.append(1.0e-3)          # channel height z in m                 / 5
myinput.append(2.0e-3)          # porous layer height z in m    / 6

myinput.append(True)            # (True) or (False) to have additional inlet and
    outlet / 7
myinput.append(5.0e-3)          # additional length of inlet  in m      / 8
myinput.append(5.0e-3)          # additional length of outlet  in m     / 9
```

The mesh parameters are shown below.

```
#######################################
## MESH
#######################################
myinput.append(5)                       # mesh: (half of) number of segments per
    channel width  / 10
myinput.append(3)                       # mesh: (half of) number of segments per
    channel height / 11
myinput.append(False)           #(True) or (False) mesh: use mesh gradient in
    channel   / 12
myinput.append(10)                      # mesh: gradient value
    /13             ( factor for refinement (small numbers like 4 will give many
    cells with little gradient / large numbers like 40 will give nice gradient) )
myinput.append(4.0)             # mesh: define number of segments for length of
    channel as multiplier of the size of segments per channel width  / 14
myinput.append(3)                       # mesh:(half of) number of segments for
    height of porous layer  / 15
myinput.append(10)                      # mesh: gradient value  for height of
    porous layer              /16
```

Important: Number of segements for channel width (and height) are defined by half of their actual value in the input section (see code listing).

If the parameter for mesh gradient in channel is set to 'True' a gradient is assumed from middle of channel towards the walls. Figure 2 (left) shows an example of a channel without mesh gradient.

Different resolution can be set for the height of the channel and height of the porous layer.
Figure 2 (right) shows an example with mesh gradient towards the walls. The gradient is defined as a geometric progression. Again, different gradient can be chosen for channel and porous transport layer.

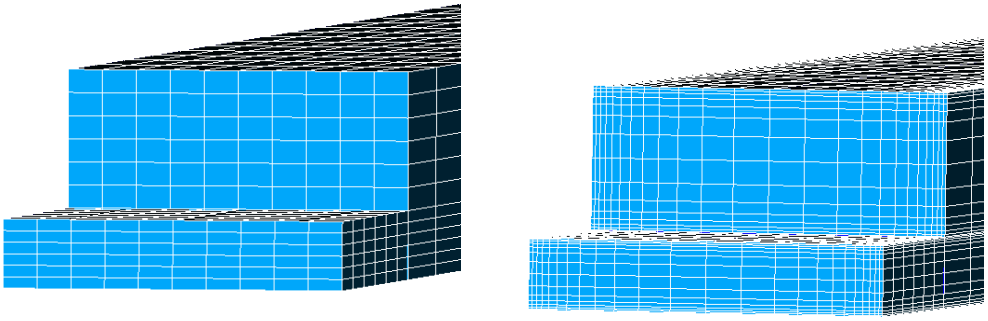


Figure 2: Left: simple mesh / Right: mesh with gradient

2.1.1 Internal parameters

Here, three lists are defined. The list 'myboxes' contains all box objects in the order they are created. For each box the list 'myboxpts' contains sublists of the eight corner points. The list 'allgroups' collects all groups which are defined within the function 'CreateGroups'. (This is needed to generate the corresponding groups for the mesh.)

```
################################
### internal parameters
################################
myboxes = []     # container for all boxes
myboxpts = []    # container for all box points
allgroups = []   # container for all generated groups
```

2.1.2 Names and lists for GROUP definition

The definition of volume groups, face groups and their names is specific for the applied CFD model in OpenFOAM.

```
##################################
### Names and lists for GROUP definition
##################################
## create an empty list for each group
# volume groups
fluid = []
myfluidnames = ['fluid', 'porous']
# face groups
fwall = []  # fluid wall
fpwall = [] # porous wall
finlet = []
foutlet = []
fmembrane = []
myfacegroups = [fwall, finlet, foutlet, fmembrane, fpwall]
myfacenames = ['wall', 'inlet', 'outlet', 'membrane', 'pwall']
```

2.2 Functions

2.2.1 DefBlock

This function generates 8 vertices of a given block. Additionally to the three coordinates an offset for the origin can be defined (x0, y0, z0). The points (corners) are collected in an ordered list and can be accessed by the index as shown in Figure 3.

```
def DefBlock(mx, my, mz, x0, y0, z0, mybox):
        #: compute 8 points (8 corners) and add them to a list -> mybox (mybox
            must be an empty list !!)
        # x-vector = mybox[0], mybox[1]
        # y-vector = mybox[0], mybox[3]
        # z-vector = mybox[0], mybox[6]
        global geompy

        pt = geompy.MakeVertex ( x0, y0, z0 )
        mybox.append(pt)

        pt = geompy.MakeVertex ( (x0 + mx), y0, z0 )
        mybox.append(pt)

        ...
```

2.2.2 CreateCell

This function generates the geometry by building it up from blocks. The function **CreateCell** makes a geometry with additional inlet and outlet region and the function **CreateCell2** makes a geometry without additional inlet and outlet region. The order for **CreateCell** is shown in Figure 4. For each box the function 'DefBlock' is called to generate the vertices. As a result the list 'myboxes' contains

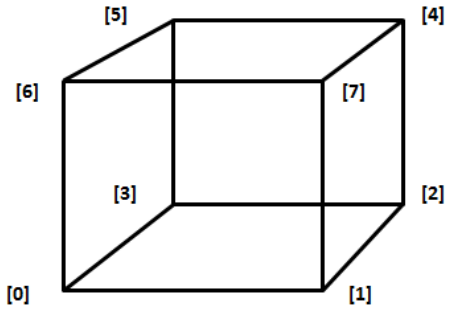


Figure 3: Box with corners numbered according to the index of the list of points

all boxes in the order they have been created. The list 'myboxpts' contains sublists with the vertices of these boxes (arranged in the same order).

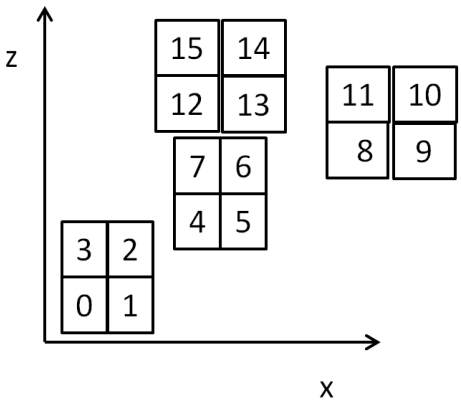


Figure 4: Order of stacking of boxes.(First set is for inlet relaxation, second set is main volume, last set is outlet relaxation zone.) Boxes 12, 13, 14 and 15 belong to the porous transport layer.

2.2.3 CreateGroups

This function creates the volume and face groups which are required by a specific model in OpenFOAM. The function **CreateGroups** is called for a geometry with additional inlet and outlet region and the function **CreateGroups2** is called in case of a geometry without additional inlet and outlet region. The function consists of two blocks. The first block defines the volume groups and the second block the face groups. Group operations must be performed on the united geometry (partition) and not on the original boxes (otherwise they will not be accessible for the mesh). By generating the partition the boxes of the united geometry have been internally re-numbered by SALOME. This is common for such type of software. The new generated objects can be retrieved by means of an access function, which now have the odering as showed in Figure 5. The first step is to obtain this ordered

list of the subshapes (boxes) and rearrange it into a list with the order defined in function 'CreateCell' (see Figure 4).

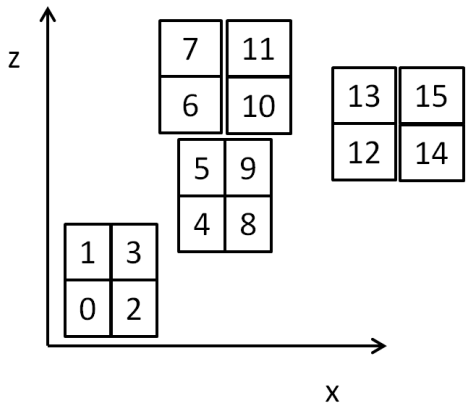


Figure 5: Order of subshapes as returned by the access function of SALOME.

```
        ##################################################################
        # Sub-shapes will be sorted by coordinates of their gravity centers
        # this list of subshapes is sorted:  increasing z / increasing x /
            increasing y
        mylist = geompy.SubShapeAllSortedCentres(mychannel, geompy.ShapeType["
            SOLID"])
        """
        ##visualization
        i = 0
        for a in mylist:
            geompy.addToStudy(a, ('box-' + str(i) ) )
            i = i + 1
        """
        # here we assign face groups
        # we have to re-sort it to fit the correct order created in function
            CreateCell
        newboxes = []                    # list to collect subshapes in the correct
             order (according to function CreateCell )
        newboxes.append(mylist[0])
        newboxes.append(mylist[2])
        newboxes.append(mylist[3])
        newboxes.append(mylist[1])
        newboxes.append(mylist[4])
        newboxes.append(mylist[8])
        newboxes.append(mylist[9])
        newboxes.append(mylist[5])
        newboxes.append(mylist[12])
        newboxes.append(mylist[14])
        newboxes.append(mylist[15])
        newboxes.append(mylist[13])
        newboxes.append(mylist[6])
        newboxes.append(mylist[10])
```

```
        newboxes.append(mylist[11])
        newboxes.append(mylist[7])
```

After that the geometry is parsed with the loops which are defined in function 'CreateCell'. The first block assigns volume groups and the second block face groups. In order to access the correct faces the list of box vertices from function 'DefBlock' are used.

```
        m = myboxpts[i]
        front = geompy.GetFaceByPoints(newboxes[i], m[0], m[1], m[6], m[7] )
        back = geompy.GetFaceByPoints(newboxes[i], m[2], m[3], m[4], m[5] )
        bottom = geompy.GetFaceByPoints(newboxes[i], m[0], m[1], m[2], m[3] )
        top = geompy.GetFaceByPoints(newboxes[i], m[4], m[5], m[6], m[7] )
        left = geompy.GetFaceByPoints(newboxes[i], m[0], m[3], m[5], m[6] )
        right = geompy.GetFaceByPoints(newboxes[i], m[1], m[2], m[4], m[7] )

        ...

        myfacegroups[0].append(bottom)
        ...
```

2.2.4 CreateMesh

This function generates the mesh. The function **CreateMesh** is called for a geometry with additional inlet and outlet region and the function **CreateMesh2** is called in case of a geometry without additional inlet and outlet region. The following steps are performed.

1. Initialize all objects with 1 segment. This generates a very basic wire frame.
2. Define number of segments for length of channel and assign this value to the edge of the first box (bottom left corner of the geometry).
3. Define number of segments for width of channel and assign it to the width of every channel.
4. Define number of segments for height of channel, interconnect, GDL, electrolyte and assign them to the correct edges.
5. Copy all groups from the geometry into the mesh.

The respective number of segments are assigned only to some key egdes. After each step all affected edges are initialized by calling the automatic propagation method.