Salome and cfMesh parametrization

From Talk:Sig Numerical Optimization

Thread summary: [Link to] [Edit]

I did a little research on combining cfMesh, Salome and OpenFoam to create completely free and Open-source parametric CFD simulation. So I want to share my experiences and maybe somebody can give me some advice to correct me or use part of what I did. I tried to create parametric Ahmed body simulations and I had some success, so I created a little tutorial. I will explain how to create parametric geometries with Salome and meshes with cfMesh.

I did a little research on combining cfMesh, Salome and OpenFoam to create completely free and Open-source CFD simulation. So I want to share my experiences and maybe somebody can give me some advice to correct me or use part of what I did. I tried to create parametric Ahmed body simulations and I had some success, so I created a little tutorial. I will explain how to create parametric geometries with Salome and meshes with *cfMesh*.

Contents

1 Salome geometry

First goal is to create completely automatic and parametric geometries. For that I used Salome and it's python scripting. For rest I used Bash scripting.

- Create working directory (eg. *ahmed*) and inside it create directory named *FoamSetup*. In that directory place all OpenFoam case needed files and directories ("0", "constant" and "system" directories; all files in them *controlDict*, *fvSolution*, *meshDict*, *RASProperties*, *p*, *U*, ...).
- Create python script template in Salome using GUI.
  - Run Salome and create geometries you need for simulation. Key is to GET SURFACES. VOLUMES ARE NOT NEEDED BUT YOU CAN USE THEM.
  - With volumes is easier to work with so use them to create flow domain
  - DO NOT DELETE AND REDO PARTS OF GEOMETRIES because numbering of faces will be wrong when python script is runned over and over. You will get bad geometries after.
  - Create groups of faces in a way you want to have patches in OpenFoam (*Inlet*, *Outlet*, ...).
  - Dump study to create python script.
- Edit created python script
  - Here is presented python script for creating Ahmed bodies with different slant angles. All edited parts (from originally dumped script) are commented:

```
### This file is generated automatically by SALOME v7.4.0 with dump python functionality

import sys
import salome
salome.salome_init()
theStudy = salome.myStudy
import salome_notebook
###
### GEOM component
###
import GEOM
from salome.geom import geomBuilder

# for solving math problems
import math
import SALOMEDS

# library for checking and creating folders
import os.path

# library for getting script path
import inspect

# get python script location/path
path = os.path.dirname(os.path.abspath(inspect.getfile(inspect.currentframe())))

geompy = geomBuilder.New(theStudy)
O = geompy.MakeVertex(0, 0, 0)
```

```
OX = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ = geompy.MakeVectorDXDYDZ(0, 0, 1)
O_1 = geompy.MakeVertex(0, 0, 0)
OX_1 = geompy.MakeVectorDXDYDZ(1, 0, 0)
OY_1 = geompy.MakeVectorDXDYDZ(0, 1, 0)
OZ_1 = geompy.MakeVectorDXDYDZ(0, 0, 1)
Box_1 = geompy.MakeBoxDXDYDZ(9000, 4000, 2000)
geompy.TranslateDXDYDZ(Box_1, -2000, -2000, 0)
listSubShapeIDs = geompy.SubShapeAllIDs(Box_1, geompy.ShapeType["FACE"])
listSubShapeIDs = geompy.SubShapeAllIDs(Box_1, geompy.ShapeType["FACE"])
listSubShapeIDs = geompy.SubShapeAllIDs(Box_1, geompy.ShapeType["FACE"])
listSubShapeIDs = geompy.SubShapeAllIDs(Box_1, geompy.ShapeType["FACE"])
Inlet = geompy.CreateGroup(Box_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(Inlet, [3])
Outlet = geompy.CreateGroup(Box_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(Outlet, [13])
Bottom = geompy.CreateGroup(Box_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(Bottom, [31])
Slip = geompy.CreateGroup(Box_1, geompy.ShapeType["FACE"])
geompy.UnionIDs(Slip, [23, 27, 33])
Box_2 = geompy.MakeBoxDXDYDZ(1044, 389, 288)
geompy.TranslateDXDYDZ(Box_2, 0, -194.5, 50)
Fillet_1 = geompy.MakeFillet(Box_2, 100, geompy.ShapeType["EDGE"], [5, 8, 10, 12])

# Instead of pure dimensions put variable that is going to be changed/parametrized. In this case it is "Slant_angle".
# Dimensions needed for chamfer are 2 lengths or angle in radians and length.
  # Here I have angle and length and I replaced them with equations with one unknown – "Slant_angle".
Chamfer_1 = geompy.MakeChamferEdgeAD(Fillet_1, math.cos(Slant_angle*(math.pi/180))*222, Slant_angle*(math.pi/180), 46, 54)

# Get Id's of all faces that make ahmed body; in this case ahmed body is called "Chamfer_1" (Salomes automated naming could change faces numbers)
Ids = geompy.SubShapeAllIDs(Chamfer_1,geompy.ShapeType["FACE"])

Ahmed = geompy.CreateGroup(Chamfer_1, geompy.ShapeType["FACE"])

# Create group Ahmed from collected faces above
geompy.UnionIDs(Ahmed, Ids)

geompy.addToStudy( O, 'O' )
geompy.addToStudy( OX, 'OX' )
geompy.addToStudy( OY, 'OY' )
geompy.addToStudy( OZ, 'OZ' )
geompy.addToStudy( O_1, 'O' )
geompy.addToStudy( OX_1, 'OX' )
geompy.addToStudy( OY_1, 'OY' )
geompy.addToStudy( OZ_1, 'OZ' )
geompy.addToStudy( Box_1, 'Box_1' )
geompy.addToStudyInFather( Box_1, Inlet, 'Inlet' )
geompy.addToStudyInFather( Box_1, Outlet, 'Outlet' )
geompy.addToStudyInFather( Box_1, Bottom, 'Bottom' )
geompy.addToStudyInFather( Box_1, Slip, 'Slip' )
geompy.addToStudy( Box_2, 'Box_2' )
geompy.addToStudy( Fillet_1, 'Fillet_1' )
geompy.addToStudy( Chamfer_1, 'Chamfer_1' )
geompy.addToStudyInFather( Chamfer_1, Ahmed, 'Ahmed' )
if salome.sg.hasDesktop():
  salome.sg.updateObjBrowser(1)

# Export the geometries as .stl files to folder defined in variable "path" at beginning
geompy.Export(Ahmed, "%s/Ahmed.stl" % path, "STL_ASCII")
geompy.Export(Bottom, "%s/Bottom.stl" % path, "STL_ASCII")
geompy.Export(Inlet, "%s/Inlet.stl" % path, "STL_ASCII")
geompy.Export(Outlet, "%s/Outlet.stl" % path, "STL_ASCII")
geompy.Export(Slip, "%s/Slip.stl" % path, "STL_ASCII")
```

2 BASH script for installing temporary Salome session

- In created working directory, create Bash script named "salomePythonScriptReader" (or whatever you wish to call it)
- Add first part of script which installs temporary Salome session to working folder:

```
#!/bin/bash
# how to run it: ./[PATH]/salomePythonScriptReader ./[PATH]/PYTHON_SCRIPT_NAME.py
# in ahmed example to start script you need to execute it: ./ahmed/salomePythonScriptReader ./ahmed/ahmed.py

# get path to base case folder and python script (/home/...../ahmed)
scriptPath=$(cd `dirname "$1"` && pwd) # path to salomePythonScriptReader script
casePath=$(cd `dirname "${BASH_SOURCE[0]}"` && pwd) # path to ahmed.py python script
pythonScriptNameExt=$(basename $1) # get name of python script. It will be ahmed.py
pythonScriptName=`echo "$pythonScriptNameExt" | cut -f1 -d'.'` # remove file extension from name of python script. It will be just ahmed

# remove directory SalomeSession if it exist in working directory
if [ -d "$casePath/SalomeSession" ]; then
    rm  -r $casePath/SalomeSession
fi

# to install Salome TUI session in working directory you need path to files "appli_gen.py" and "config_appli.xml".
# next 2 lines of code searches user directory ~/ for Salome installation.
# ON FIRST RUN IT WILL TAKE SOME TIME (MINUTE OR TWO) afterwards only a second or two!
# path to "appli_gen.py" python script and "config_appli.xml" can be manually setup by commenting "find" command and entering path on your own.
appliPath=$(find ~/ \( ! -wholename *Trash* \) -wholename *bin/salome/appli_gen.py -print -quit)
configPath=$(dirname $appliPath)

# start Salomes python script to install temporary Salome session to working folder
python $appliPath --verbose --prefix=$casePath/SalomeSession --config=$configPath/config_appli.xml

#start Salome session in TUI
source ${casePath}/SalomeSession/runAppli -t
```

3 BASH script for running created temporary Salome session in loop

After the part for installing temporary Salome session to working directory we need to change and run python script over and over (in a loop). Append this part to created bash script to run session and python scripts in parametric way:

```
# uncomment this block for PARAMETRIC salome session (: <<'PARAMETRIC' is just bash block commenting)
#: <<'PARAMETRIC'

# "Slant_angle" is variable in python script "ahmed.py" (created above as example) that is changing.
  #"pythonValues" is array of values which will be substituted in python script.
pythonVariable="Slant_angle"
pythonValues=(25 35) # for this example it will be array of slant angles for ahmed body.

# for loop:
  # reads python script and finds string "Slant_angle"
  # changes string "Slant_angle" to values defined in "pythonValues" (25, 35)
  # runs Salome session executing changed python script
for i in "${pythonValues[@]}"
do
    # remove case directories if they exist
    if [ -d "${pythonScriptName}_$i" ]; then
        rm  -r ${pythonScriptName}_$i
    fi

    # create case directories with names as combination of python script name and "pythonValue". In this example directories will be "ahmed_25", "ahmed_35"
    mkdir -p $casePath/${pythonScriptName}_$i/constant/triSruface

    # copy OpenFoam case files and directories, from "FoamSetup" directory, to each generated case directory
    cp -r $casePath/FoamSetup/* $casePath/${pythonScriptName}_$i

    # copy new Salome python script to case directory. In this example ahmed.py will be copied and named ahmed_25.py
    cp $scriptPath/$pythonScriptNameExt $casePath/${pythonScriptName}_$i/${pythonScriptName}_$i.py

    # change string "Slant_angle" with array values (25, 35)
    sed -i "s/$pythonVariable/$i/g" $casePath/${pythonScriptName}_$i/${pythonScriptName}_$i.py

    # run Salome session and python script
    cd $casePath/SalomeSession
    ./runSession python $casePath/${pythonScriptName}_$i/${pythonScriptName}_$i.py

    # move generated .stl files to directory "triSurface"
    mv $casePath/${pythonScriptName}_$i/*.stl $casePath/${pythonScriptName}_$i/constant/triSruface
done

# terminate salome sessions (with "killSalome.py" python script located in the same directory as "appli_gen.py") and delete directory "SalomeSession"
killPath=$configPath
python $killPath/killSalome.py
rm  -r $casePath/SalomeSession

# end part of bash block commenting
#PARAMETRIC
```

4 BASH script for running created temporary Salome session only ones

This part is for nonparametric running of python script. It does same thing like parametric one except it only goes trough script ones. Append it to bash script instead of parametric one if want to use it (or insert both of them and comment out one you don't want to use). Since code is similar to parametric one it is not as thoroughly explained.

```
# uncomment this block for NONPARAMETRIC salome session
#: <<'NON_PARAMETRIC'

# run Salome session and read python script
if [ -d "case_${pythonScriptName}" ]; then
    rm  -r case_${pythonScriptName}
fi
mkdir -p $casePath/case_${pythonScriptName}/constant/triSurface
cp -r $casePath/FoamSetup/* $casePath/case_${pythonScriptName}

# run python script
cd $casePath/SalomeSession
./runSession python ${scriptPath}/${pythonScriptNameExt}

# move .stl files to "triSurface" directory
mv ${scriptPath}/*.stl ${casePath}/case_${pythonScriptName}/constant/triSurface

# terminate salome sessions and delete directory "SalomeSession"
killPath=$configPath
python $killPath/killSalome.py
rm  -r $casePath/SalomeSession

#NON_PARAMETRIC
```

5 Preparation of created boundary .stl files for cfMesh

After creating geometry boundary .stl files it is time to mesh the domain. Although it is not implemented in bash script it can be added very easily, because even meshing can be done automatically now thanks to Creative Fields, Ltd. and their free cfMesh.

First we need to prepare created .stl files for meshing with cfMesh. Each .stl file represent one patch, so to create closed domain all patches have to be combined in to one “big” .stl file. One thing you have to be careful about is to set patches names in new combined .stl file.

Example is shown for Bottom.stl file in Ahmed geometry. Bottom.stl file looks like this:

```
solid
 facet normal  0.000000e+00  0.000000e+00 -1.000000e+00
   outer loop
     vertex -2.000000e+03 -2.000000e+03  0.000000e+00
     vertex -2.000000e+03  2.000000e+03  0.000000e+00
     vertex  7.000000e+03  2.000000e+03  0.000000e+00
   endloop
 endfacet
 facet normal  0.000000e+00  0.000000e+00 -1.000000e+00
   outer loop
     vertex  7.000000e+03 -2.000000e+03  0.000000e+00
     vertex -2.000000e+03 -2.000000e+03  0.000000e+00
     vertex  7.000000e+03  2.000000e+03  0.000000e+00
   endloop
 endfacet
endsolid
```

When all .stl files are combined together, part for patch Bottom should look like this:

```
….
 endfacet
endsolid Inlet
solid Bottom
 facet normal  0.000000e+00  0.000000e+00 -1.000000e+00
   outer loop
     vertex -2.000000e+03 -2.000000e+03  0.000000e+00
     vertex -2.000000e+03  2.000000e+03  0.000000e+00
     vertex  7.000000e+03  2.000000e+03  0.000000e+00
   endloop
 endfacet
 facet normal  0.000000e+00  0.000000e+00 -1.000000e+00
   outer loop
     vertex  7.000000e+03 -2.000000e+03  0.000000e+00
     vertex -2.000000e+03 -2.000000e+03  0.000000e+00
     vertex  7.000000e+03  2.000000e+03  0.000000e+00
   endloop
 endfacet
endsolid Bottom
solid Outlet
….
```

For that purpose I have written little bash script I called "stlCombine" which combines .stl files and outputs "mergedStl.stl" file.

```
#!/bin/bash
# how to run it: ./stlCombine [PATH TO DIRECTORY WITH .stl FILES] [PATH TO OUTPUT DIRECTORY]
# example with ahmed is ./stlCombine ./ahmed/ahmed_25/constant/triSurface ./ahmed/ahmed_25

# get .stl files location/path specified with running script (from example ./ahmed/ahmed_25/constant/triSurface)
filePath=$(cd $1 && pwd)
# get merged file output location/path specified with running script (from example ./ahmed/ahmed_25)
outputPath=$(cd $2 && pwd)

# check if file mergedStl.stl exists. If it does delete it.
if [ -f $filesPath/mergedStl.stl ]
then
    rm $filesPath/mergedStl.stl
fi

# remove all temporary files from input directory (where .stl files are located)
if [ -d "$filePath/*~" ]; then
    rm $filePath/*~
fi

# add patch name:
  # change first and last line of every file in input location from solid to solid [.stl FILE NAME] and from endsolid to endsolid [.stl FILE NAME]
  # merge them together
# example is from "solid" to "solid Bottom" and from "endsolid" to "endsolid Bottom"
for file in $filePath/*.stl
do
    # get .stl file name
    fileName=`basename $file`
    # remove .stl file extension
    patchName=`echo "$fileName" | cut -f1 -d'.'`

    # append line solid [.stl FILE NAME] (eg. solid Bottom ) to file mergedStl.stl
    echo "solid $patchName" >> $outputPath/mergedStl.stl

    # copy all lines except first and last one from loaded .stl file and append them to mergedStl.stl
    tail -n +2 $file | head -n -1 >> $outputPath/mergedStl.stl

    # append line endsolid [.stl FILE NAME] (eg. endsolid Bottom ) to file mergedStl.stl
    echo "endsolid $patchName" >> $outputPath/mergedStl.stl
done
```

6 Running cfMesh

Setting up mesh dictionary (meshDict) for cfMesh is easy and very good explained in its documentation (cfMesh documentation (http://www.c-fields.com/technical-area/downloads/documentation-cfmesh#C1)) and tutorials. Running stlCombine and cfMesh (after generating geometries and case directories with explained bash script "salomePythonScriptReader") is simple and those few commands can be appended to script "salomePythonScriptReader". Now we have completely automatized geometries and mesh generation. If you decide to go step further, It is possible to append OpenFoam commends to bash script "salomePythonScriptReader" and run simulations on created cases.

7 Tips

When I created first few meshes, cfMesh created hole where Ahmed body is. Something like this: cfMesh error image (http://s11.postimg.org/noimu5nn7/cf_Mesh_error.png)

My first guess was, that happened because patch Ahmed in .stl file had face vectors pointing in wrong direction (because they were created from another volume in Salome). Vectors are defined in .stl file as "facet normal [VECTOR VALUES]". It can be seen in Bottom.stl in 2. line ("facet normal 0.000000e+00 0.000000e+00 -1.000000e+00"). At that point I have written another bash script that rotates face vectors in .stl file. After I finished script I realized it is not necessary. Face vectors weren't problem to cfMesh. The problem was mesh was set to coarse. Afterwards I realized **IF cfMesh DOES SOMETHING THAT YOU DIDN'T WANT (LIKE THAT HOLE I SHOWED OR DOESN'T REFINE MESH IN SOME SPECIFIED AREA) PROBLEM IS IN meshDict AND REFINEMENT SETTINGS**. If somebody has some other experiences please do share. :)

Still I will put here that vector rotating script. Maybe someone will find it handy for something else.

```
#!/bin/bash
# how to run it: ./vectorRotation [PATH TO .stl FILE] (eg. ./vectorRotation ./mergedStl.stl)

# get file name and location
filePath=$(cd `dirname "$1"` && pwd)
# get file name with extension
fileNameExt=$(basename $1)
# get the file name without extension
fileName=`echo "$fileNameExt" | cut -f1 -d'.'`

# remove inverted file if it already exist
if [ -f $filePath/inverted$fileNameExt ]; then
    rm $filePath/inverted$fileNameExt
fi

# read file line by line
while read line
do
    # check if line has string facet normal
    if  "$line" == *"facet normal"*
    then
        # if it has, extract vector coordinates
        x[1]=$(echo "$line" | awk -F' ' '{print $3}')
        x[2]=$(echo "$line" | awk -F' ' '{print $4}')
        x[3]=$(echo "$line" | awk -F' ' '{print $5}')
        # rotate vector coordinates
        i=0
        # go trough x array created one step above
        for num in "${x[@]}"
        do
            i=$[$i +1]
            # check if number is negative
            if  $num == -*
            then
                # if truth remove first number character (that is minus sign)
                x[i]=${x[$i]#?}
            else
                # if false prepend minus sign to number
                x[i]="-${num}"
            fi
        done
        # reconstruct line with new numbers
        ortLine=" facet normal "
        # append numbers to first string facet normal
        for string in "${x[@]}"
        do
            ortLine="$ortLine $string"
        done;
        # push reconstructed line to inverted .stl file
        echo "$ortLine" >> $filePath/inverted$fileNameExt
    else
        # if the line doesn't have string facet normal push the line to inverted .stl file
        echo "$line" >> $filePath/inverted$fileNameExt
    fi
done < $1

# remove .stl extension from original file (so it is invisible to other scripts)
mv $filePath/$fileNameExt $filePath/$fileName
```

Kruno (talk) 11:54, 18 July 2014

More

Hi Kruno, great work, thanks for the effort.

Could you provide a minimal working case that does what you want ? I think I can help with Salome and cfmesh, but I'd like to see the whole thing as one block instead of several parts.

Cheers.

Ben

Bennn (talk) 18:19, 11 August 2014

Parent More
