



# TreeFoam 操作マニュアル

ver 3.31-250809

TreeFoam は、OpenFoam を GUI で操作できるツール。

OpenFOAM は、基本的に CUI ベースで操作する為、操作性（生産性）が悪く、初心者には敷居が高い。これを少しでも改善する為に、OpenFOAM が GUI 上で操作できる様に工夫したものが、TreeFoam。

このマニュアルは、事例を多用してまとめている為、実際に試す事ができ理解が深まる。特に 6 項の基本的な操作方法の例については、tutorials の計算方法をまとめたもので、これをそのまま試す事で、TreeFoam の殆どが理解できるものと思う。

このマニュアルは、  
OpenFOAM-2.4、3.0、4.0、5、6、7、8、9、10、11、12  
13、v1806、v1906、v1912、v2006、v2012、v2106、v2112、  
v2206、v2306、v2406、v2506  
および、TreeFoam-3.31 でまとめている。

25/08/09 藤井

## 変更経歴

ver 2.25-150308	新規作成
ver 2.32-150726	wx.version-3.0 (ubuntu15.04) に対応。 topoSetEditor のコマンドを OF-2.3 に対応。 繰り返し処理用の resulyTyoe 「sets」「zones」を追加 runParallel に preserve の設定を追加 createBaffles220 を修正 (空 patch 作成部分を削除) これにより、topoSetEditor、並列処理、内部 patch に関する部分のみ修正。 paraFoam による multiRegionCase の確認方法を追加 インストール方法を修正
ver 2.33-150809	TreeFoam のバージョンの記録方法を変更。 multiRegion の材料設定方法を追加。これに伴いマニュアルの一部を修正。
ver 2.34-150918	mesh 作成の区分に wall を追加。
ver 2.35-151010	mesh 作成の区分に empty, symmetry, symmetryPlane, wedge を追加。 cfMesh による mesh 作成を追加。 gridEditor に cell クリア (空白 cell 作成) を追加。 「#includeEtc」に対応。
ver 2.36-151212	ubuntu-15.10、OpenFOAM-3.0.0 への対応。 setFields、mapFields に新たなボタンを追加。 helyxOS を操作マニュアルから削除。 操作マニュアルは、基本的に OF-2.4 で作成しているが、ここに OF-3.0.0 の場合を追加した。
ver 2.37-160602	sshfs によるサーバ接続、login シェルを追加。
ver 2.40-161106	snappyHexMesh の並列処理追加。 server 間、server と local 間の file や folder のコピー追加。 FOCUS 用の Job 管理ツールを追加。
ver 2.40-170226	FOCUS Job 管理ツールに実行している Job の「log 表示」、「plotWatcher」のボタンを追加。
ver 2.41-170424	gridEditor を server 用に対応させる。
ver 2.42-170809	複数のサーバに接続できる様に修正。名古屋大学 CX400 用の Job 管理ツール追加。
ver 2.43-170928	OF-5.0 への対応。名古屋大学 FX100 用の Job 管理ツール追加。
ver 2.44-180624	folder コピー貼り付け時、desktop 名を確認する様に修正。 ubuntu-18.04 対応 (警告発生する為。) folder 構成読み取りアルゴリズムを見直し、処理を高速化 (local、server 共)
ver 2.45-190303	multiRegion の case の処理方法を修正。 TreeFoam の log 表示用 textBox の表示処理を wxPython の version に合わせる。
ver 3.01-200502	全スクリプトを GTK+3 (一部 Qt4)、python3 用書き換え。 TreeFoam の機能を一部削除。directory 読み込み方法見直し、高速化。
ver 3.02-200603	gridEditor に patchViewer を追加。 patchViewer (3D viewer) を使うことによって、形状を確認しながら境界条件の設定が可能。
ver 3.03-200706	PySide, PySide2 上で TreeFoam が動かなくなった事を修正。(バグ修正) patchViewer 上に、左右反転、回転のボタンを追加。 TreeFoam インストール方法の詳細を追記。
ver 3.04-200802	PySide, PySide2 上の gridEditor で cell の copy&Paste ができなかった事を修正。
ver 3.05-201125	newCase 作成後、newCase が選択表示される様に修正。 OF-v2006 に対応。 v2006 では「List<scalar> 0();」「List<scalar> 0;」、「"\$:regionWall.T"」が出力され、gridEditor で読み込み時エラーが発生していた。

ver 3.06-210622	OF-8 対応。multiRegionCase の書式が変更、「\$!output/T」の書式追加。
ver 3.07-210729	stl ファイルの編集に移動と回転を追加し、その形状を vtk 表示させる様に修正。 9-1-1.項を全面書き換え。
ver 3.08-211105	topoSetEditor から起動する meshViewer を追加。 (patch, zones, sets, stl の形状が確認できる。)
ver 3.09-220108	メッシュ形状を速やかに確認できる meshViewer を追加。(paraView よりも早く起動、確認できる。) vtk-9.1.0 対応。
ver 3.10-220212	OF-9, OF-v2106, OF-v2112 への対応。 popupMenu に「コピー & mesh 貼り付け」「コピー & mapFields 貼り付け」を追加。
ver 3.11-220514	meshViewer に stlViewer を link。「case 貼り付け」を case 内の全 folder に対して可能な様に修正。
ver 3.12-220723	OF-v2206, OF-10 への対応。
ver 3.13-220906	stl ファイル check を追加。gridEditor のバグ修正。
ver 3.14-230308	流体-構造連成解析を追加。folderCopyPaste 方法を修正。
ver 3.15-230325	連成計算時の restartFile 保存方法を修正。
ver 3.16-230530	流体-固体の熱連成、固体の熱ひずみを追加。
ver 3.17-230706	流体構造連成、流体固体熱連成のアルゴリズム見直し、高速化。
ver 3.18-230805	OF-9、OF-v2206 の連成解析に OF-10 を追加。熱連成のバグ修正。 OF-v2306 への対応。
ver 3.20-231101	流体構造連成、流体固体熱連成のアルゴリズム見直し、OF-Fistr 同時計算に加え、交互計算を追加。
ver 3.21-231130	FrontISTR との連成計算にて、FrontISTR は、thread 並列が前提だったが、process 並列ができる様に変更。
ver 3.22-240225	OF-11 への対応。PyFoam の install 方法追記。
ver 3.23-240503	FrontISTR との連成計算にて、OF の時間の有効桁数を Fistr 側にも反映。
ver 3.24-240723	OF-12, OF-v2406 への対応。
ver 3.25-240831	Qt6 (PySide6) への対応。
ver 3.26-241007	stlViewer に outline (featureEdge) を追加。
ver 3.27-241211	mesh 作成用 dialog (snappyHexMesh, cfMesh) を追加。
ver 3.28-250210	meshViewer に internalMesh の clip 機能を追加
ver 3.29-250407	pyFoam 関連を TreeFoam 外に移動。snappyHexMesh の locationInMesh 修正方法変更
ver 3.30-250722	OF-13, OF-v2506 への対応。
ver 3.31-250809	remeshAndMapping.py コマンド追加。tutorials 取得時のバグ修正。

## 目次

1. TreeFoamとは.....	6
2. インストール方法.....	7
2-1. 動作環境.....	7
2-2. インストール.....	7
2-2-1. 必要 package 一覧とそのインストール.....	7
2-2-2. python-3.10 の仮想環境の設定と pyFoam のインストール(ubuntu24.04).....	8
2-2-3. deb パッケージから TreeFoam をインストール.....	9
2-2-4. ソースから TreeFoam をインストール.....	10
2-3. configTreeFoam の設定内容.....	11
3. 起動方法.....	14
3-1. 通常の起動方法.....	14
3-2. 起動しない場合のエラー内容の確認.....	14
3-3. configTreeFoam の修正.....	15
4. TreeFoam の起動画面.....	16
5. 基本的な操作方法.....	17
5-1. メニュー構造とその内容.....	17
5-1-1. メニューバー、ツールバー.....	17
5-1-2. ポップアップメニュー.....	20
5-1-3. ダブルクリックによる操作.....	23
6. 基本的な操作方法の例.....	24
6-1. 天井駆動のキャビティ流れ (cavity) の操作例.....	24
6-1-1. myTutorials フォルダ作成.....	24
6-1-2. tutorials の「cavity」を「myTutorials」フォルダにコピー.....	25
6-1-3. blockMesh 作成.....	27
6-1-4. icoFoam の実行.....	27
6-1-5. paraFoam による結果の確認.....	28
6-1-6. 境界条件を変更する場合.....	29
6-1-7. constant、system フォルダの内容確認.....	31
6-1-8. controlDict の内容確認.....	32
6-2. ダムの決壊 (damBreak) の操作例.....	33
6-2-1. tutorials の「damBreak」を「myTutorials」フォルダにコピー.....	33
6-2-2. blockMesh の作成.....	36
6-2-3. setFields で値をセット.....	37
6-2-4. 境界条件の確認.....	38
6-2-5. interFoam の実行.....	39
6-2-6. 結果の確認.....	40
6-2-7. 並列計算.....	41
7. メッシュ作成の例.....	45
7-1. snappyHexMesh による通常メッシュの作成.....	45
7-1-1. case の作成.....	45
7-1-2. モデル形状.....	46
7-1-3. メッシュ作成用データ入力.....	49
7-1-4. メッシュ作成.....	50
7-1-5. レイヤ作成.....	51
7-2. snappyHexMesh による faceZone や cellZone を含むメッシュ作成の例.....	54
7-2-1. メッシュ作成用 case の作成.....	54
7-2-2. モデル形状.....	54
7-2-3. メッシュ作成用データ入力.....	55
7-2-4. メッシュ作成.....	56
7-2-5. 解析用 case の作成.....	57
7-2-6. setFields で値をセット.....	58
7-2-7. データセット状態の確認.....	62
7-2-8. baffle (内部パッチ) 作成.....	64
7-3. cfMesh による通常メッシュの作成.....	70
7-3-1. case の作成.....	70
7-3-2. レイヤ付きメッシュ作成用の csv ファイル作成.....	71
7-3-3. メッシュ作成.....	73
7-4. salome で作成したメッシュを FOAM 形式に変換する例.....	77
7-4-1. case の作成.....	77
7-4-2. salome によるメッシュ作成.....	77
8. TreeFoam 内の主なアプリケーション.....	81
8-1. gridEditor.....	81
8-1-1. 起動画面.....	81
8-1-2. gridEditor の起動.....	82
8-1-3. メニュー構造と内容.....	82



---

8-1-4.	field 内変数や patchGroup、include 文の扱い.....	88
8-1-5.	binary 形式の扱い.....	95
8-2.	topoSetEditor.....	99
8-2-1.	topoSet のコマンド構造.....	99
8-2-2.	topoSetEditor の画面.....	100
8-2-3.	topoSet コマンドの内容.....	100
8-2-4.	topoSet コマンドの抽出について.....	101
8-2-5.	topoSetEditor の操作例.....	102
8-2-6.	繰り返しの Action について.....	106
8-2-7.	組み合わせ (combined) Action について.....	110
8-3.	meshViewer.....	113
8-3-1.	internalMesh の表示.....	114
8-3-2.	patch の表示.....	115
8-3-3.	sets、zones の表示.....	116
8-3-4.	multiRegion の表示.....	116
8-3-5.	stl の表示.....	116
8-4.	FrontISTR との連成解析.....	118
8-4-1.	連成計算方法.....	118
8-4-2.	具体例.....	120
8-4-3.	熱連成計算の検証.....	130
8-4-4.	OpenFOAM、FrontISTR の他バージョンへの適用.....	131
9.	応用例.....	132
9-1.	ファイルの操作・編集.....	132
9-1-1.	stl ファイルの編集.....	132
9-1-2.	binary 形式ファイルの扱い方.....	146
9-2.	gridEditor の表示.....	155
9-2-1.	列 (field) の表示.....	155
9-2-2.	行 (patch 名など) の表示.....	159
9-2-3.	セル (patch 内容など) の表示.....	162
9-2-4.	空 patch (face 数が「0」の patch) の作成、削除.....	164
9-2-5.	空白セルを zeroGradient で埋める.....	166
9-2-6.	internalField をクリア.....	168
9-2-7.	cell データを editor で編集 (「...」付きデータの編集).....	169
9-3.	field へのデータセット.....	173
9-3-1.	setFields によるデータセット.....	173
9-3-2.	mapFields によるデータセット.....	179
9-4.	内部 patch の作成.....	186
9-4-1.	cyclic、mapped、baffle の patch 作成方法.....	186
9-4-2.	cyclic、baffle を含む mesh の並列計算方法.....	196
9-5.	multiRegion の case.....	200
9-5-1.	case 作成例.....	200
9-6.	計算サーバを接続する場合.....	231
9-6-1.	サーバ接続の為の準備.....	231
9-6-2.	サーバ接続とサーバのマウント.....	232
9-6-3.	サーバ切断とサーバのアンマウント.....	234
9-6-4.	サーバとローカル間の folder コピー方法.....	234
9-6-5.	FOCUS の Job 管理.....	235

---

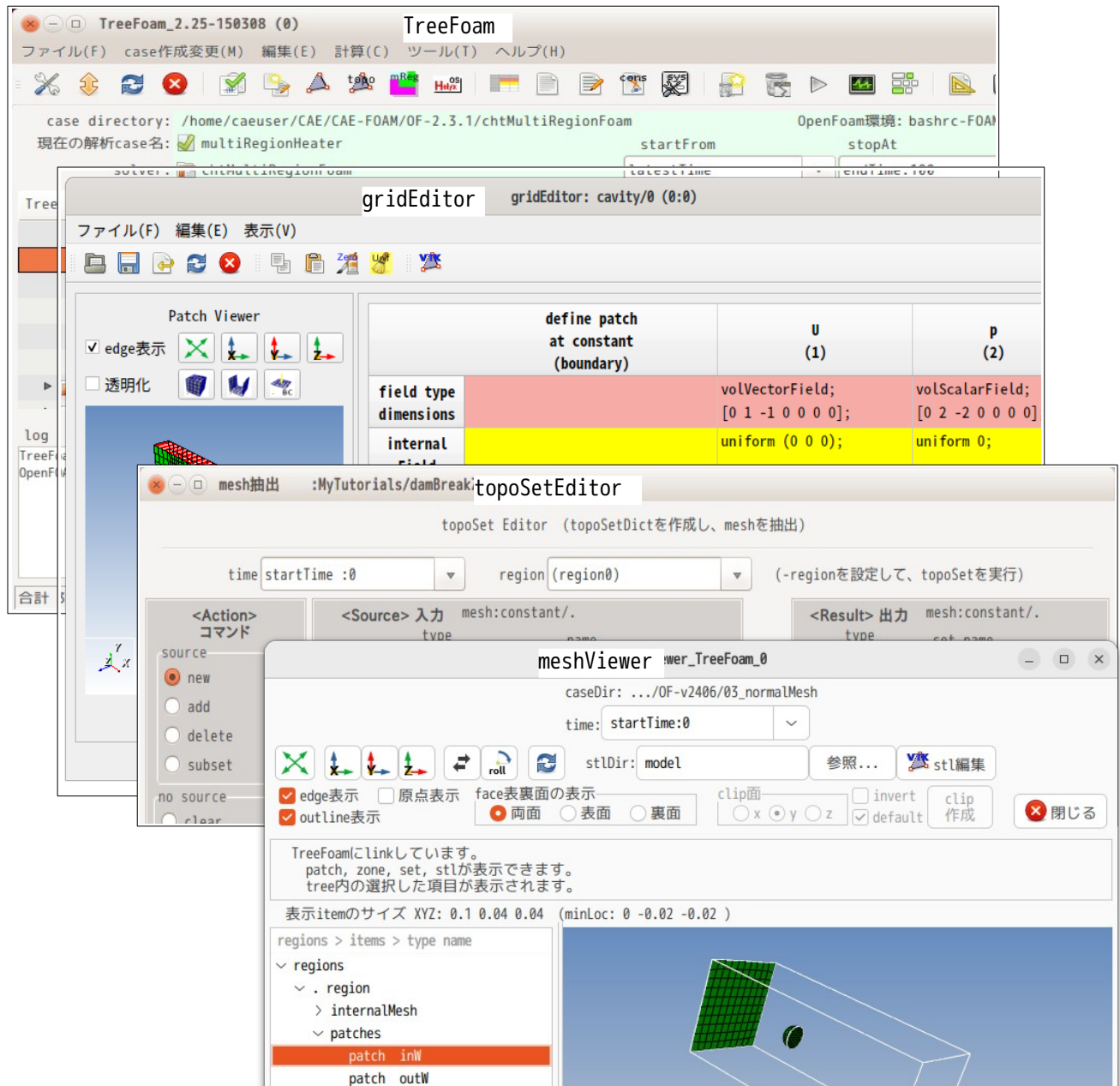
## 1. TreeFoam とは

OpenFOAM を使いやすくする為の GUI ツールで、代表的には以下の機能を備えている。これら機能が GUI 上で行えるので、直感的に操作でき、case フォルダの管理もしやすくなる。

- 1) case がツリー表示される。
- 2) ツリー上に solver 名や計算結果フォルダ (数字フォルダ) の数等の case の概略が表示される。
- 3) FreeCAD、salome、paraView のランチャを備えている。
- 4) Dict ファイルを意識せずに、snappyHexMesh や cfMesh でメッシュが作成できる。
- 5) gridEditor を使って OpenFOAM の境界条件が、形状を確認しながら表形式で編集できる。
- 6) topoSetEditor を使って、cell の抽出、加工が容易にできる。
- 7) meshViewer を使って、メッシュ形状を速やかに確認できる。
- 8) TreeFoam や gridEditor は、マルチタスクに対応している為、複数起動して、お互いに copy & paste が可能。また TreeFoam と gnome (nautilus) 間で、ファイルの copy & paste も可能。
- 9) 計算サーバを接続して、TreeFoam 上でサーバ内の Job 管理ができる。(FOCUS のみ)
- 10) OpenFOAM と FrontISTR による流体-構造連成計算ができる。

TreeFoam で使用しているアイコンは、gnome に標準で装備されているアイコンと salome、paraView のアイコンを TreeFoam/icons フォルダ内にコピーして使っている。

利用しているユーティリティプログラムは、pyFoam と OpenFOAM 標準のユーティリティを使っている。



## 2. インストール方法

TreeFoamは、Ubuntu Gnome デスクトップ上で Gtk3 (一部 Qt) の環境下で、python3 と bash シェルスクリプトを使って開発しているので、この環境下であれば、基本的に作動する。  
Qtについては、PyQt4, PyQt5, PyQt6 (又は PySide, PySide2, PySide6) に対応させている。

### 2-1. 動作環境

以下の環境が整えば、TreeFoamを定められた folder にコピーするだけで作動する。また、これ以外に TreeFoam の操作対象である OpenFOAM、paraView が最低限インストールされている事が前提になる。  
また、TreeFoam の機能をフルに使うのであれば、他に pyFoam、さらに Salome や FreeCAD が動作する環境が望まれる。

OS	Ubuntu-11.04 以上で Gnome デスクトップ環境。 TreeFoamは、ファイルマネージャと端末を使っている。これらはデフォルトで ファイルマネージャ:nautilus、端末:gnome-terminal が設定されている。 (configTreeFoamを修正することで、他の fileManager、端末も使用可。)
OpenFOAM	OpenFOAM-2.3.0 以上 このマニュアルは、ver-2.4、3.0、5.0、6.0、7.0、8.0、9、10、11、12、13、 v1806、v1906、v1912、v2006、v2012、v2106、v2112、v2206、v2306、v2406、 v2506 用で書いている。 バージョンによって、操作が異なるので GUI を変えている。
pyFoam	Ubuntu-24.04 の場合、python の仮想環境を準備する。(2-2-2 項参照) Ubuntu-22.04 以前の場合、pip install PyFoam で最新版をインストールする。
ParaView	
必要アプリ	python-3.0 以上 nautilus, gnome-terminal, gedit Gtk3(Glade), Qt(PyQt4, PyQt5, PyQt6, PySide, PySide2 or PySide6) ssh、scp、sshfs (サーバ接続の為にコマンド) xclip, wl-clipboard (ファイルマネージャと TreeFoam 間の copy&paste 用) office, VTK

### 2-2. インストール

インストール方法は、deb パッケージからインストールする方法と、ソースをインストール場所に直接コピーする方法がある。  
尚、計算サーバを接続する為には、「9-6-1. サーバ接続の為に準備」に従ってセットする必要がある。

#### 2-2-1. 必要 package 一覧とそのインストール

TreeFoamは、ubuntu の Gnome デスクトップ環境で使用する事を前提に作成している。この理由は、TreeFoam上で使用している Gtk3 の環境が整っている為。  
しかし、WSL 上の ubuntu に TreeFoam をインストールする事を考えると、Gnome デスクトップが準備されていないので、まずこれらの準備が必要なる。この為、ここでインストールに必要な package の一覧を示しておく。

<基本 package>		
package 名	内容	備考
python3	python	Gnome デスクトップでは、インストール済
gnome-terminal	端末	↑
gedit	editor	↑ (TreeFoam-3.24 から gedit への依存を削除)
nautilus	fileManager	↑

この部分は、Gnome デスクトップ環境であれば、既にインストールされているもの。  
WSL 上の ubuntu の場合は、これらをインストールする事によって、Gtk3 関連のライブラリもインストールできる。

<VTK の package>		
package 名	内容	備考

python3-vtk7 python3-vtk9	VTK ↑	python3 の vtk モジュール (ubuntu-21.04 以降 の場合)
------------------------------	----------	--

<Qt 関連の package>

package 名	内容	備考
python3-pyqt6	PyQt6	
-	PySide6	
python3-pyqt5	PyQt5	PyQt6, PyQt5, PyQt4, PySide6, PySide2, PySide の いずれかを install
python3-pyside2	PySide2	
python3-pyqt4	PyQt4	
python3-pyside	PySide	

Qt は、上記いずれかのパッケージ 1 ヶをインストールする。  
PySide6 については、deb パッケージが無く、pip でインストールする。

WSL 上の ubuntu 環境では、上記をインストールすれば、一応 TreeFoam が起動するが、日本語が表示できず文字化けするので、さらに日本語環境を整える必要がある。

ubuntu22.04 以前は、pip を使ってパッケージをインストールしても、問題なくインストールできたが、ubuntu24.04 から、pip コマンドが pipx コマンドに代わり、動作も変わっているので、ubuntu24.04 以降は、pip を使わずに deb パッケージでインストールすることを推奨する。(ubuntu23.04, 23.10 は、未確認。)

ubuntu22.04 以前から pip を使ってインストールしていた (できる) whl パッケージは、vtk、pyFoam があるが、ubuntu24.04 上では、これらパッケージのインストールは、工夫が必要になる。

vtk を ubuntu24.04 にインストールする場合は、pip でインストールせず、以下の方法でインストールする事を推奨する。最初に paraview の deb パッケージをインストールする。これにより、vtk-9.2 もインストールされる。(vtk の deb パッケージは、vtk-9.1 であり、これをインストールすると、deb パッケージの paraview の vtk と競合するので、paraview の deb パッケージ版が動かなくなる為。)

pyFoam を ubuntu24.04 にインストールする場合は、python の仮想環境を準備し、この仮想環境上に pyFoam をインストールする。(2-2-2 項を参照)  
この理由は、pyFoam が python-3.11 までしか対応しておらず (25/07/11 現在)、ubuntu24.04 は、python-3.12 が標準。python-3.12 では、pyFoam のエラーが発生し、動かない状態が発生している。これを回避するためには、python3.10 の仮想環境を準備するしかない。

## 2-2-2. python-3.10 の仮想環境の設定と pyFoam のインストール(ubuntu24.04)

ubuntu24.04 上では標準の状態では、pyFoam (plotWatcher) が起動しないので、python-3.10 の仮想環境を作成する必要がある。(python-3.10 は、ubuntu22.04 で使っていた環境と同じ環境になる。)

仮想環境の作成方法は、以下の方法による。

必要なパッケージをインストール

```
$ sudo apt install build-essential libbz2-dev libdb-dev libreadline-dev libffi-dev
libgdbm-dev liblzma-dev libncursesw5-dev libsqlite3-dev libssl-dev zlib1g-dev uuid-
dev tk-dev
```

pyenv をインストール (git リポジトリを「~/pyenv」に clone する。)

```
$ git clone https://github.com/pyenv/pyenv.git ~/pyenv
```

「.bashrc」の最後に以下を追記して「~/pyenv」に PATH を通す

```
----- 追記内容 -----
export PYENV_ROOT="$HOME/.pyenv"
command -v pyenv >/dev/null || export PATH="$PYENV_ROOT/bin:$PATH"
eval "$(pyenv init -)"
-----
```

「.bashrc」への追記内容を反映させる

```
$ source ~/.bashrc
$ pyenv -version          #pyenv 作動確認
pyenv 2.4.7
```

インストール可能な python を確認する

```

$ pyenv install -l
Available versions:
  2.1.3
  2.2.3
  :
  3.10.13
  3.10.14          #python-3.10 の最終版
  3.11.0
  3.11-dev
  :

python-3.10 の最終版「3.10.14」をインストール
$ pyenv install 3.10.14

仮想環境フォルダを作成
$ mkdir ~/local_python
$ cd local_python
$ pyenv local 3.10.14          #python3.10 の仮想環境が完成する

```

以上の操作で、python3.10 の仮想環境が構築できたことになる。  
引き続き、仮想環境の python3.10 の pip を使って「pyFoam」をインストールする。

```

$ cd ~/local_python
$ python3 -m venv ~/local_python
$ source ~/local_python/bin/activate          #仮想環境をセット
$ python3 -m pip install PyFoam              #python3.10 を起動して pyFoam をインストール
$ deactivate                                  #環境を元に戻す

```

上記の操作で、~/local\_python フォルダに pyFoam がインストールされた事になる。  
仮想環境 python の起動スクリプトを作成する。  
以下のスクリプトを「~/bin/python3.10」として保存して、PATH が通るようにする。

```

----- python3.10 起動スクリプト -----
#!/bin/bash
cd ~/local_python
source ~/local_python/bin/activate          #環境をセット
python3 $1 $2 $3 $4 $5 $6 $7 $8 $9         #python3.10 を起動
deactivate                                  #実行後、環境を戻す
-----

```

以上の設定で、以下の様に「python3.10」を入力することで、python3.10 が起動する。

```

$ python3.10
Python 3.10.14 (main, Jul 21 2024, 09:47:56) [GCC 13.2.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> exit()

```

この設定で、plotWatcher の起動は、以下の様な設定で起動できることになる。(ハッチング部)

```

----- python の仮想環境上での plotWatcher 起動スクリプト例 -----
pyFoamDir=~/.local_python
logFile=$1
python3.10 $pyFoamDir/bin/pyFoamPlotWatcher.py $logFile > /dev/null
-----

```

従来の起動方法と大差なく作成する事ができる。

### 2-2-3. deb パッケージから TreeFoam をインストール

以下の deb パッケージからインストールする。

```

Treefoam_3.31.250809_all.deb          TreeFoam の本体
treefoam-doc_3.31.250809_all.deb      ドキュメント (ヘルプファイル)

```

これらのファイルを使って、端末から以下の様に入力してインストールする。

```

$ sudo dpkg -i ./treefoam_3.31.250809_all.deb
$ sudo dpkg -i ./treefoam-doc_3.31.250809_all.deb

```

Gnome デスクトップ環境でインストールするのであれば、上記コマンド入力で問題なくインストールできるが、Gnome デスクトップ環境でない状態で、上記コマンドを入力すると、gnome-terminal, nautilus がインストールされていない場合、インストールできない旨のエラーメッセージが発生する。この場合には、予めイ

インストールするか、treefoamを以下の様に apt-get でインストールする。

```
$ sudo apt-get install ./treefoam_3.31.250809_all.deb
```

apt-get は、依存 package を確認し、依存 package も同時にインストールしてくれる。

TreeFoam のインストール場所は、「/opt」であり、各ユーザの TreeFoam 設定ファイルは、「~/TreeFoamUser」フォルダ内に設定される。（~/TreeFoamUser フォルダは、TreeFoam 起動時に毎回チェックし、無ければ TreeFoam が作成し、ここに TreeFoam の設定ファイルをコピーする。）  
任意の場所にインストールしたい場合は、2-2-4 項の方法でインストールする。

このインストール方法では、インストール作業は楽になるが、前記した様に、インストール場所や TreeFoam の設定ファイルの保存場所は決まっており、これらの場所を変更する事はできない。

また、DEXCS2023 以前の DEXCS に DEXCS2024 用の TreeFoam (TreeFoam-3.27 以降) をインストールする場合は、インストール後に TreeFoam の修正が必要になる。  
この理由は、TreeFoam 内の関数が DEXCS 側 (FreeCAD の AppImage) から呼び出されており、DEXCS2024 から FreeCAD の GUI パッケージが pySide2→Qt5 に変更されている為、DEXCS2024 用の TreeFoam 側もこれに対応させている。（以下のハッチング部分）

----- /opt/TreeFoam/python/pyTreeFpm.py の内容 (DEXCS2024 用) -----

```
def command(self, comm):
    """ command を実行する。終了するまで待ち、終了後に戻る。
    終了を待たずに戻る場合は、command の最後に「&」を追加する。
    DEXCS 用に修正。（AppImage で実行） """
    os.chdir(self.caseDir)
    #print('comm = ',comm)
    if comm[0] == '.':
        #from PySide2 import QtCore          #DEXCS22.04 以前用
        from PyQt5 import QtCore           #今回の DEXCS24.04 用
        process = QtCore.QProcess()
        working_dir = self.caseDir
    :
```

以上により、DEXCS2023 以前の DEXCS に、今回の DEXCS 版 TreeFoam をインストールする場合は、この部分を以下の赤字の様に修正する必要がある。

----- /opt/TreeFoam/python/pyTreeFpm.py の内容 (DEXCS2023 以前用) -----

```
def command(self, comm):
    """ command を実行する。終了するまで待ち、終了後に戻る。
    終了を待たずに戻る場合は、command の最後に「&」を追加する。
    DEXCS 用に修正。（AppImage で実行） """
    os.chdir(self.caseDir)
    #print('comm = ',comm)
    if comm[0] == '.':
        from PySide2 import QtCore          #DEXCS22.04 以前用
        #from PyQt5 import QtCore           #今回の DEXCS24.04 用（コメントアウトする）
        process = QtCore.QProcess()
        working_dir = self.caseDir
    :
```

TreeFoam をインストールした後は、「~/TreeFoamUser/configTreeFoam」の内容を各自の環境に合わせないと、TreeFoam が起動しても、使う事ができない。最低限、OpenFOAM、paraFoam、editor の設定は必要になる。設定方法は、2-3、3-3 項を参照。

アンインストールする場合は、synaptic を起動して削除するか、端末を起動して以下の様に入力してアンインストールする。

```
$ sudo dpkg -r treefoam
$ sudo dpkg -r treefoam-doc
```

## 2-2-4. ソースから TreeFoam をインストール

ソースからのインストール方法は、圧縮ファイルを解凍し、インストール場所に TreeFoam をコピーするのみで済む。ただし、DEXCS2304 以前の DEXCS に DEXCS2024 用の TreeFoam (TreeFoam-3.27 以降) をインストールする場合は、インストール後、TreeFoam を修正する必要がある。詳細は、前項を参照。

TreeFoam のフォルダ構成は以下。

\$HOME	ホーム Dir
TreeFoam	ホームフォルダ直下に TreeFoam をコピー
app	TreeFoam 上から起動するアプリの起動用スクリプト
bin	実行ファイル
data	TreeFoam の各モジュールのデータ、stl ファイル等
frontIstr	流体-構造連成解析関連のデータ、スクリプト
help	help ファイル
icons	アイコンファイル
python	TreeFoam の python モジュール
glade	Glade で作成した TreeFoam の GUI データ
Qt	Qt4Designer で作成した ui ファイル

また、TreeFoam は、以下の環境変数を準備しており、これらを使って path の設定を行っている。

\$TreeFoamPath	#インストール場所
\$TreeFoamUserPath	#user 用の data や temp ファイルの保存場所

これらの環境設定は、実行シェルスクリプト「treefoam」内で環境設定するので、どこにインストールしても、ソースを修正することなく、実行できる。(treefoam スクリプト内で、スクリプト自身の directory を調べ、環境設定を行っている。)

実行例：

```
$ <インストール場所>/treefoam
```

実行後、\$TreeFoamUserPath フォルダ内には、「app」、「data」、「temp」フォルダが作成され、最低限の必要な設定ファイルがコピーされる。TreeFoam 実行時には、ここの内容が参照される。

TreeFoam をインストールした後は、「\$TreeFoamUserPath/configTreeFoam」の内容を各自の環境に合わせないと、TreeFoam が起動しても、使う事ができない。最低限、OpenFOAM、paraFoam、editor の設定は必要になる。設定方法は、次項を参照。

### 2-3. configTreeFoam の設定内容

\$TreeFoamUserPath/configTreeFoam が TreeFoam の環境を決めているので、ここの内容のみ各自の環境に合わせれば、TreeFoam が正常に作動し、OpenFOAM が操作できる事になる。configTreeFoam 内での設定項目と設定内容は、以下の通り。

- 1) language  
TreeFoam 上で使用する言語を設定する。(設定は、Japanese or English のみ)  
TreeFoam は、国際化されているので、ここを English に設定して再起動すると、表示内容が全て英語表示になる。  
インストール後の初回起動時は、TreeFoam が linux の言語設定を調べ、言語が「ja\_JP.UTF-8」の場合(正確には頭2文字が「ja」の場合)、ここの設定を Japanese、それ以外は English の設定で起動する。(\$TreeFoamUserPath フォルダ有無で初回起動の確認を行う。)  
2回目以降の起動時は、configTreeFoam 内の language 設定を確認し、Japanese ならば日本語、English ならば英語表示で TreeFoam を起動する。
- 2) logFile  
TreeFoam の logFile を作成するかどうかを決定する(設定は、yes or no のみ)  
yes の場合、logFile を作成し、その log が TreeFoam 下部のテキストボックスに逐次表示される。  
no の場合、logFile は作成せず、端末に log が表示される。  
尚、TreeFoam 下部のテキストボックス中の log テキストは、行数の制限(200 行)を設けており、必要以上に log をため込まない。(folder 選択時に文字数を確認し、200 行以上を削除している。)全ての log 内容は、「\$TreeFoamUserPath/temp」フォルダ内に log ファイルがあるので、これで確認できる。
- 3) OFversion  
TreeFoam 起動時と TreeFoam 上から OpenFOAM 環境設定を変更した時、TreeFoam が環境変数「\$WM\_PROJECT\_VERSION」を読み込み、その内容を TreeFoam がここに書き込む。
- 4) rootDir  
TreeFoam 上で表示される Tree 構造の最上位の dir を記述する。  
この内容は、指定が無い、または存在しない dir の場合、起動時に \$HOME に設定される。  
ここの設定は、TreeFoam が終了する度に書き直される。
- 5) workDir  
TreeFoam が表示している Tree 構造内で選択されている dir (解析 caseDir) が書き込まれる。



実在しない dir の場合は、\$HOME を書き込む。  
この設定は、TreeFoam が終了する度にこの内容が書き直される。

- 6) `bashrcFoam`  
OpenFOAM 起動用のスクリプトを記述する。  
この中には、起動用だけでなく、必要な箇所に PATH、PYTHONPATH を通しておく。  
以下の例に従って、環境に合わせておく。  
例: `$TreeFoamUserPath/app/bashrc-FOAM-13.0`
- 7) `paraFoam`  
`paraFoam` 起動用のスクリプトを記述する。  
以下の例を参照。  
例: `$TreeFoamUserPath/app/runParaFoam-13.0`
- 8) `plotWatcher`  
`plotWatcher` の起動用スクリプトを記述する。  
`pyFoam` の python スクリプトが python-3.12 以降 (ubuntu24.04) では動かない為、python-3.10 の  
仮想環境上で起動するスクリプトを作成して起動する。以下の例を参照  
例: `$TreeFoamUserPath/app/runPlotWatcher-venv` #仮想環境で起動させるスクリプト
- 9) `salomeMeca`  
`salome` の起動用スクリプトを記述する。  
以下の例を参照  
例: `$TreeFoamUserPath/app/runSalomeMeca-2023.1.0`
- 10) `CAD`  
使用する CAD の起動用スクリプトを記述する。  
以下の例を参照  
例: `freecad`
- 11) `editor`  
使用する editor の起動用スクリプトを記述する。  
editor の設定は、裏で動く設定にしない。この設定にしないと、TreeFoam 上で binary ファイル  
の編集ができなくなる。(9-1-2 項参照。)  
例: `gedit --standalone`

インストールに当たっては、以下の例の様に設定する。(設定不要項目もある。)  
OpenFOAM が使える設定にする為には、「`bashrcFoam`」、「`paraFoam`」、「`editor`」の設定は、最低限必要  
になる。

```
#
# TreeFoam の設定
# -----
#

#使用する言語                                #「Japanese」に設定
language Japanese

#logFile 作成有無                            #「yes」に設定
logFile yes

# OpenFOAM のバージョン                      #設定不要
OFversion 13

# rootDir の設定                             #設定不要
rootDir /home/caeuser

# 選択されている現在の case の設定          #設定不要
workDir /home/caeuser/CAE/CAE-FOAM/OF-13/cavity

# FOAM 端末の環境設定ファイル                #実際の作動環境に合わせておく
# OpenFOAM の他、必要な箇所に PATH、PYTHONPATH を通しておく。
bashrcFOAM $TreeFoamUserPath/app/bashrc-FOAM-13.0

# paraFoam の起動                            #実際の環境に合わせておく
paraFoam $TreeFoamUserPath/app/runParaFoam-13.0

#plotWatcher の実行コマンド
```



```
# python の仮想環境で起動する場合は、「runPlotWatcher-venv」を選択
plotWatcher $TreeFoamUserPath/app/runPlotWatcher-venv

# SalomeMeca の起動
salomeMeca $TreeFoamUserPath/app/runSalomeMeca-2023.1.0

# CAD の起動
CAD freecad

# editor の設定
# editor が close するまで、待つ設定にする。
editor gedit --standalone
```

さらに、fileManager (nautilus)、terminal (gnome-terminal)、office (looffice) の設定も変更することができる。(詳細は、configTreeFoam ファイル内を参照)

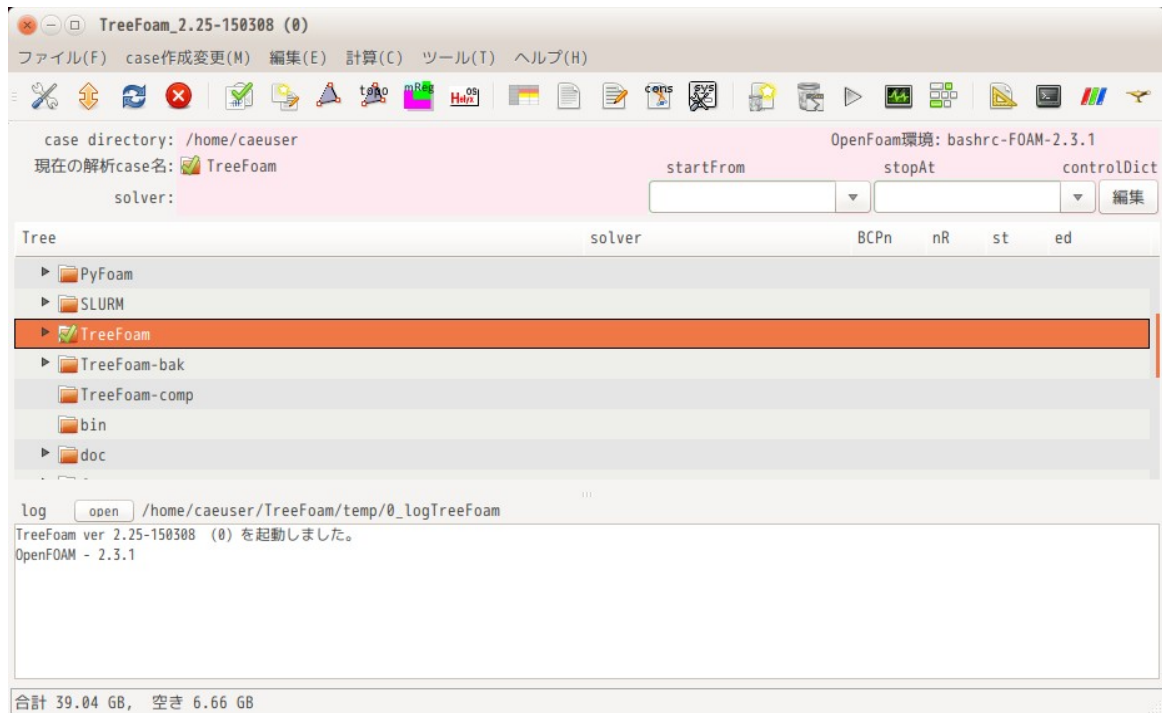
configTreeFoam の修正は、editor で編集しても構わないが、TreeFoam 起動させた後、3-3 項の方法で行った方が、スムーズに行う事ができる。

### 3. 起動方法

#### 3-1. 通常の起動方法

「/opt/TreeFoam/treefoam」を実行すると TreeFoam が起動するので、ランチャを作る場合は、「treefoam」を起動するように設定する。  
 尚、deb パッケージでインストールした場合は、/usr/share/applications フォルダ内に「TreeFoam.desktop」ファイルがコピーされており、これを実行しても起動する。またこのファイルをランチャに登録する事ができる。  
 また、~/.local/share/applications フォルダ内に同じファイル名「TreeFoam.desktop」があると、それが優先されるので、削除しておく。

インストール後、初回の TreeFoam 起動時は、以下の画面が現れる。



#### 3-2. 起動しない場合のエラー内容の確認

TreeFoamは、logをTreeFoamのテキストボックスに表示させている都合上、TreeFoamの画面が現れる前にエラーが発生してしまうと、何も表示されず止まってしまう。(エラーメッセージが表示されない。)  
 TreeFoamは、以下の順番で起動している為、

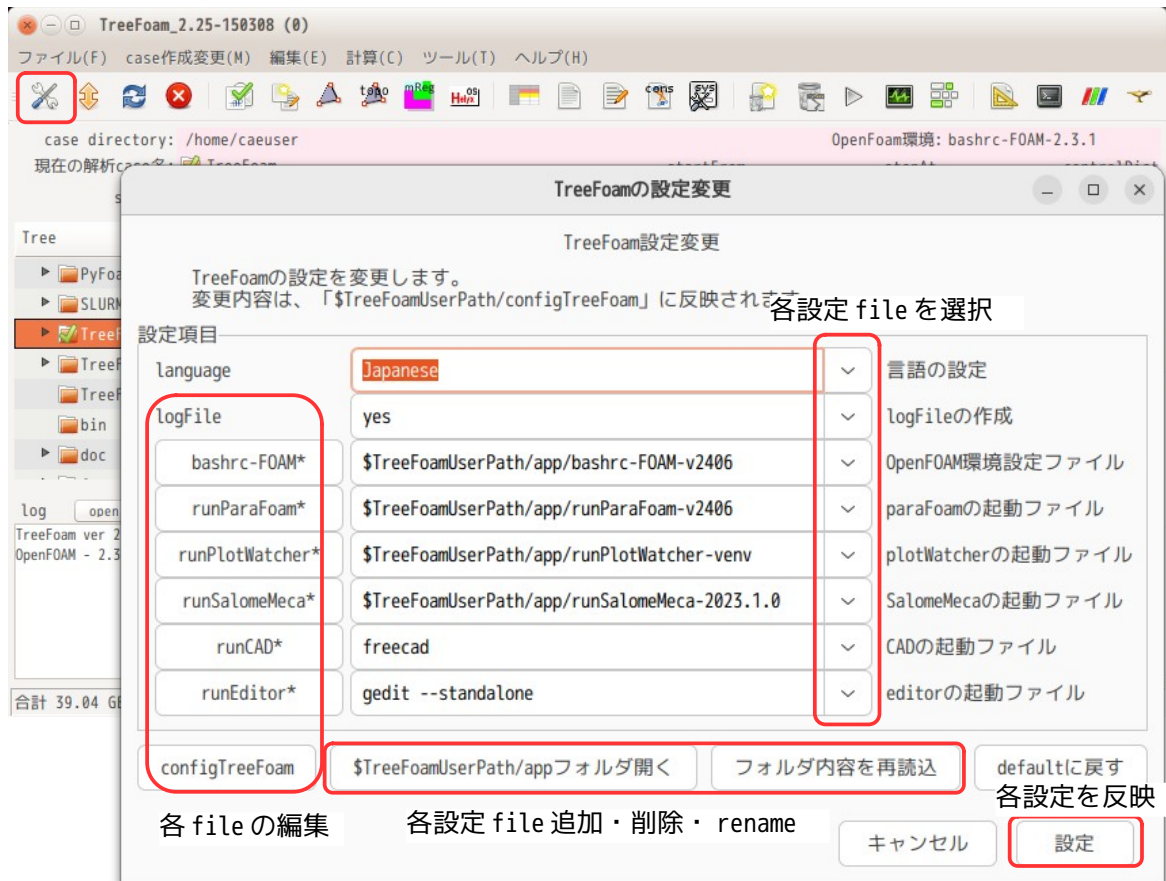
treefoam	環境設定後、treefoam.py を起動
treefoam.py	TreeFoam 本体 (GUI)

端末を起動して、直接「treefoam」を実行しても TreeFoam が起動する。もし、エラーが発生していた場合は、端末を起動して、その端末から treefoam を実行すると、起動した端末にエラーメッセージが出力されるので、エラー内容が確認できる。

### 3-3. configTreeFoam の修正

TreeFoam が起動した後、TreeFoam の作動環境を決定している configTreeFoam を各自の環境に合わせる必要がある。(configTreeFoam の内容は、2-3 項参照。)

configTreeFoam の修正は、下図の様に TreeFoam 上の  をクリックして現れた「TreeFoam の設定変更」画面上で行う。

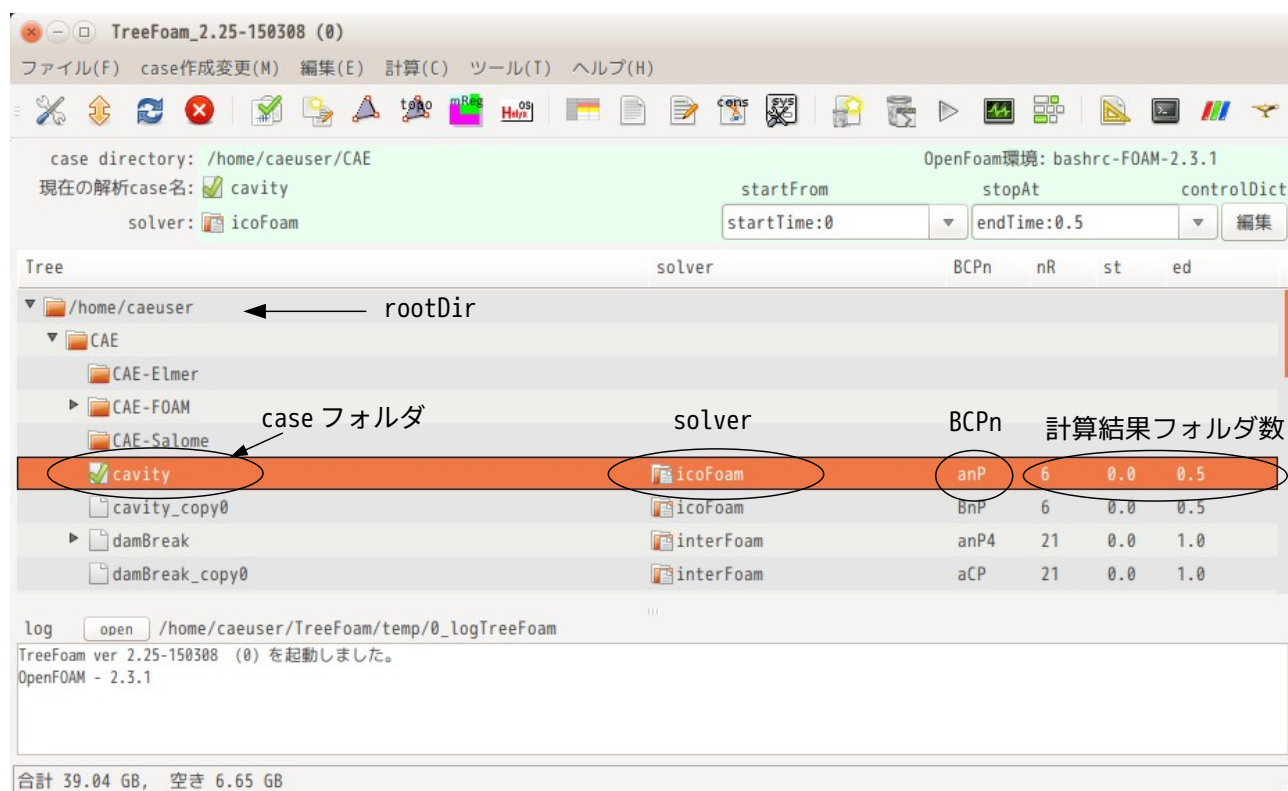



この画面上で、各種設定ファイル名を設定したり、設定ファイルの内容を編集したりする事ができる。(各設定ファイルは、「\$TreeFoamUserPath/app」フォルダ内に保存されている。)

内容を修正した後は、「設定」ボタンをクリックして、configTreeFoam にその修正内容を反映させる。

## 4. TreeFoam の起動画面

TreeFoam の画面ではフォルダがツリー表示され、OpenFOAM の case フォルダには、設定している solver や計算結果フォルダ数等の情報が下図の様に表示される。



上記の画面では、マークの付いているフォルダ「cavity」が解析 case として設定されている。TreeFoam 上から、この case は、以下の内容である事が判る。

```

solver :icoFoam (icoFoamが設定)
BCPn   :anP (ascii, 非圧縮, シングルコアでデータが保存されている。)
        B   :acii(a) or Binary(B)
        C   :非圧縮(n) or 圧縮(C)ファイル
        Pn  :並列数 (例: P4 は、4 並列の処理)
nR      :6   (計算結果 folder 数が 6 ケ)
st      :0.0 (計算開始時間が「0.0」)
ed      :0.5 (計算終了時間が「0.5」)

```

フォルダをツリー表示する時に、そのフォルダが OpenFOAM の case かどうか (system/controlDict が存在するかどうか) を確認し、case フォルダの場合は、controlDict を読み込み、solver 名や書式、計算結果フォルダ等を表示する。

TreeFoam では、ファイルの書式が binary や圧縮ファイルでも、その書式を判断して読み込む事ができるので、書式にかかわらず、違和感なく使う事ができる。通常、binary ファイルは editor で編集できないが、TreeFoam 上からファイルを開く時、binary を ascii に変換して editor で読み込み、保存する場合は、ascii を binary に変換して保存するので、binary ファイルでも内容の確認・編集ができる様にしている。(8-1-5、9-1-2 項を参照。)

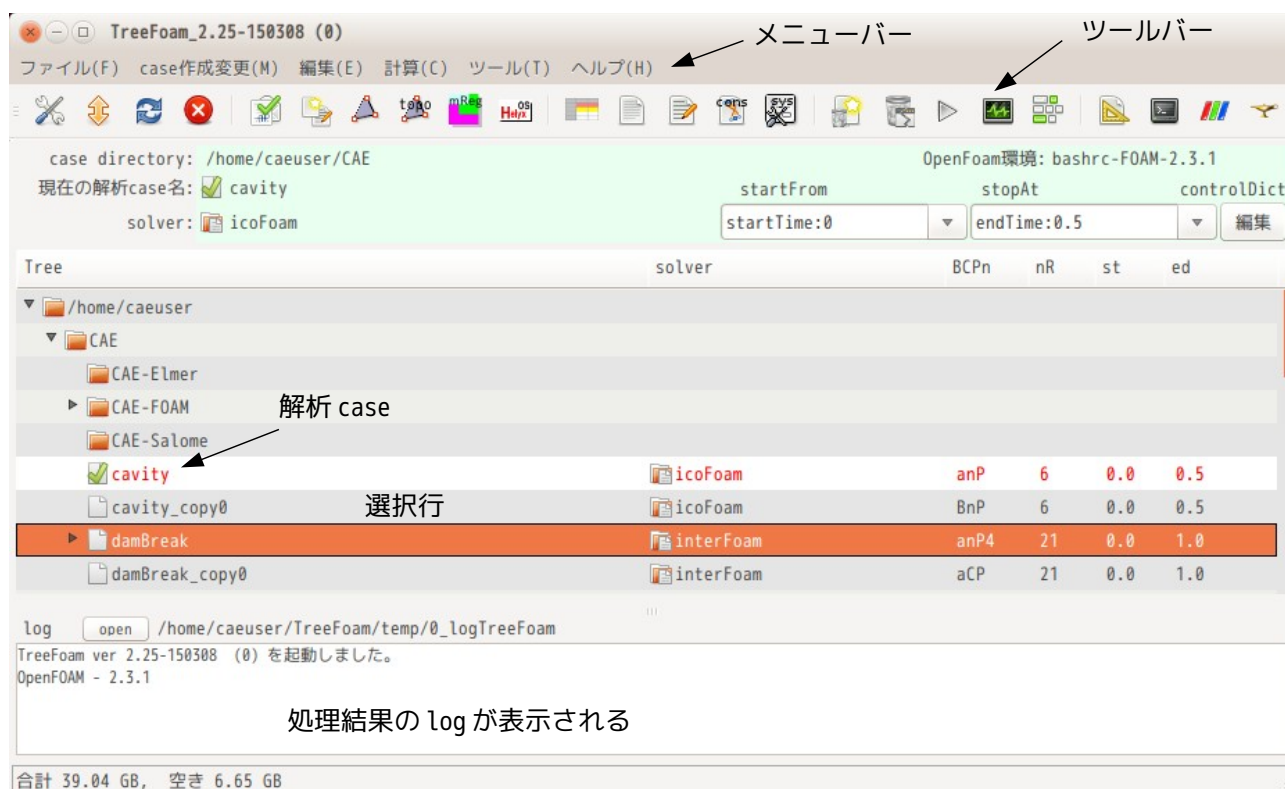
## 5. 基本的な操作方法

TreeFoamには、メニューバー、ツールバー、ポップアップメニューで処理を選択して実行する形式をとっている。また、選択行をダブルクリックして直接処理する方法もある。(ダブルクリックする位置によって処理が異なる。)

メニューバー、ツールバーは、解析 case (☑付きフォルダ) に対する処理で、ポップアップメニュー、ダブルクリックは、選択行に対する処理になるので、使い分けて処理する。  
下図の例では、メニューバー、ツールバーは、解析 case 「cavity」 に対する処理になり、ポップアップメニュー、ダブルクリックは、選択行 「damBreak」 に対する処理となる。

各処理状況は、TreeFoam 下部のテキストボックスにその log が表示されるので、その処理結果を確認できる。

また、計算開始時間「startFrom」と計算終了時間「stopAt」は、修正する頻度が高いので、直接 TreeFoam の上部テキストボックスを操作し修正できる (controlDict を書き換える) 様になっている。



FOAM 端末に関しては、使用頻度が高いので、メニューバー、ツールバー、ポップアップメニューのいずれでも起動できる。希望する directory で FOAM 端末を起動する場合は、その directory を TreeFoam 上で選択し、ポップアップメニューから起動するのが、最もスムーズに起動できる。

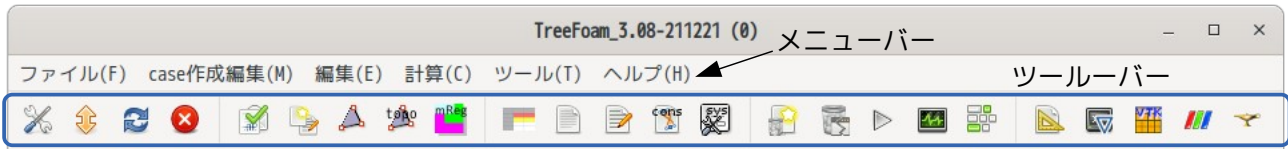
### 5-1. メニュー構造とその内容

メニューは、メニューバー、ツールバー、ポップアップメニューがある。また、ダブルクリックによって処理する方法もあるので、操作しやすい方法を選択する。  
また、多用するコピーや貼り付け、case 貼り付けについては、ショートカットキーを準備している。  
操作方法の例は、6 項を参照。

#### 5-1-1. メニューバー、ツールバー







解析 case (☑付きのフォルダ) に対する処理は、基本的にメニューバーとツールバー上で行う。










これらのメニューバーとツールバーの処理は、以下になる。

## 1) ファイル (E)

-  **configTreeFoamの変更 (I)**  
TreeFoamの環境を決定している「TreeFoam/configTreeFoam」ファイルを GUI 上で編集する。  
\$TreeFoamUser/app フォルダ内に各設定ファイルを準備する事で GUI 上でこれらファイルを編集する事ができる。
-  **root の変更 (C)**  
TreeFoam が表示している Tree 構造の最上位を設定する。  
この root の directory は、\$HOME 以下で設定する。
-  **再読み込み (R)**  
TreeFoam が表示している Tree 構造を再読み込みして、再表示する。  
端末、あるいはデスクトップ上でフォルダを追加したり、削除した場合は、TreeFoam 上の Tree 構造が狂ってくる。この様な場合、これを実行して Tree を再表示させる。
-  **閉じる (Q)**  
TreeFoamを終了する。  
TreeFoam終了時は、このボタンをクリックして終了させる。  
window 上部のをクリックしても TreeFoam を終了させる事はできるが、これで終了させると、終了処理を行わず閉じてしまうので、 ボタンで TreeFoam を終了させる。

## 2) case 作成変更 (M)

-  **解析 case として設定 (S)**  
選択しているフォルダを解析 case として設定する。  
この解析 case がメニューバー、ツールバーの操作対象になる。
-  **新しい case 作成 (N)**  
ここで以下の3種類の操作ができる。
  - ・ 解析 case として設定しているフォルダ内に tutorials のケースをコピーする。
  - ・ 解析 case 内の solver を別の case の solver に入れ替える。
  - ・ 解析 case 内のメッシュを別の case のメッシュに入れ替える。
-  **mesh 編集 (M)**  
ここでメッシュに関する操作を行う。
  - ・ メッシュ作成  
blockMesh や snappyHexMesh、cfMesh でメッシュを作成する。
  - ・ メッシュ変換  
unv 形式のメッシュ (ファイル名: mesh.unv) を FOAM 形式に変換する。  
メッシュの scale を変更する。
  - ・ 内部パッチの作成  
faceZone から内部 patch (baffle) を作成する。
  - ・ 領域分割  
mesh を分割して multiRegion タイプ case を作成する。
-  **topoSetEditor 起動 (I)**  
topoSetEditor を起動して、特定の mesh を抽出したり、加工できる。
-  **multiRegion の設定 (R)**  
multiRegion タイプの case を操作する。
  - ・ 領域間の境界条件を設定する。
  - ・ region に設定されている全ての境界条件を保存したり、設定したりできる。
  - ・ region を削除したり、region 名を変更できる。
  - ・ region 内の file に容易にアクセスでき、その内容が編集できる。

## 3) 編集 (E)

## 開く (O)

解析 case のフォルダをファイルマネージャ (nautilus) で開く。



## gridEditor の起動 (G)

gridEditor を起動して、field の初期値や境界条件を編集する。  
boundary、各 field の internalField と boundaryField の内容が表形式で編集できる。



## field 編集 (F)

解析 case の field を editor で開き、編集する。



## fieldDataSet 又は clear (S)

setFields を実行して、field にデータをセットしたり、指定した field の internalField や boundaryField をクリアする。



## properties 編集 (P)

解析 case の constant フォルダ内の file 名を指定して、editor で開く。  
その file が圧縮 file や binary でも editor で開き、編集できる。



## dictionary 編集 (D)

解析 case の system フォルダ内の file 名を指定して、editor で開く。  
その file が圧縮 file や binary でも editor で開き、編集できる。

## コピー (C) ctrl-C

選択しているフォルダを clipBoard にコピーする。  
clipBoard は、system の clipBoard を使っているため、ここでコピーした folder をファイルマネージャ (nautilus) でも貼り付ける事ができる。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## 貼り付け (P) ctrl-V

clipBoard にコピーされている folder や file を選択しているフォルダに貼り付ける。  
clipBoard は、system の clipBoard を使っているため、ファイルマネージャ (nautilus) 側で、コピーした folder や file も貼り付ける事ができる。  
また、貼付け後、親フォルダ以下のリンクのチェックを行っているので、貼り付けたファイルの絶対参照リンクが壊れた場合も修復できる。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## case 貼り付け (B) ctrl-B

system の clipBoard にコピーされている case を、選択しているフォルダ内に貼り付ける。  
この貼り付けを使うと、計算結果等の余分な folder や file は、貼り付けない。  
また、貼付け後、親フォルダ以下のリンクのチェックを行っているので、貼り付けたファイルの絶対参照リンクが壊れた場合も修復できる。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## mesh の貼り付け...

コピーした case 内のメッシュを、選択している case 内に貼り付ける。(mesh 貼り付け用の dialog が起動する。)  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## field の mapping 貼り付け...

コピーした case 内の field を、選択している case の field に mapping する。(mapFields 実行用の dialog が起動する。) する。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## フォルダ名変更 (R)

選択しているフォルダ名を変更する。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## 新しいフォルダ追加 (N)

選択しているフォルダ内に新しいフォルダを追加する。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

## フォルダ削除 (E)

選択しているフォルダをゴミ箱に移動する。  
この操作は、対象が解析 case ではなく、選択しているフォルダになる。

解析 case に設定しているフォルダは、削除できない。

#### 4) 計算 (C)



case の初期化  
計算結果フォルダや不要な folder、file を削除して、case を初期化する。



計算結果の削除  
計算結果フォルダのみ削除する。



計算開始 (G)  
解析 case として設定している case の solver をシングルコアで実行する。



plotWatcher 起動 (W)  
解析 case の solver を実行中（実行後）、このボタンをクリックすると plotWatcher が起動するので、solver 実行時の残渣が確認できる。



並列計算 (P)  
ここで、並列計算を行う。  
decomposeParDict の作成、各 processor 毎の領域分割、並列計算開始、計算結果の再構築、各 processor 内の file 操作（file コピーと削除）が行える。



流体-構造連成解析  
OpenFOAM と FrontISTR を組み合わせて、流体-構造の連成解析する。

#### 5) ツール (I)



CAD の起動 (C)  
configTreeFoam で設定されている CAD を起動する。



FOAM 端末の起動 (I)  
configTreeFoam で設定されている FOAM 端末を起動する。



meshViewer の起動  
meshViewer を起動する。



paraFoam の起動 (P)  
configTreeFoam で設定されている paraFoam を起動する。



salome の起動 (S)  
configTreeFoam で設定されている salome を起動する。

#### 6) ヘルプ (H)

使い方 (U)  
ヘルプを表示する。

バージョン表示 (V)  
TreeFoam のバージョンを表示する。

### 5-1-2. ポップアップメニュー

ポップアップメニューは、基本的に選択行に対する処理を行う。ポップアップメニューは、右クリックする場所で、メニュー内容が異なっている。

#### 2) solver 部



マウントしているサーバに応じた Job 管理ツールが起動する。この画面上で、Job ファイルの作成、編集や Job 投入などが行える。

現在のところ、FOCUS と名古屋大学 XC400 用の Job 管理ツールを備えている。

#### sshfs サーバ マウント

選択している folder に、sshfs コマンドでサーバをマウントする。マウント方法は、「`~/TreeFoamUser/data/sshfs_data`」ファイル内に記述。サーバマウント後は、gnome のファイルマネージャからでも、サーバ内容が確認できる。

#### sshfs サーバ アンマウント

マウントしたサーバをアンマウントする。

#### 貼り付け (scp 圧縮転送, cp)

サーバとローカル間で file や folder を高速転送 (圧縮転送: scp -Cr) して貼り付ける。貼り付ける file や folder は、system の clipboard の内容になるので、予めそれらをコピーしておく。圧縮転送したくない場合は、TreeFoam や gnome のファイルマネージャ上で通常の copy&Paste すれば済む。サーバ内の file、folder ををサーバ内に貼り付ける場合は、ssh で cp コマンドを送出して、貼り付ける為、高速で copy&Paste できる。

#### case 貼り付け (scp 圧縮転送, cp)

上記の貼り付けと同じ考え方で、case 内の計算結果など、余分な folder や file を貼り付けない。(case を初期化した状態で貼り付ける。)

#### server 内 folder 削除 (rm)

server 内 folder を削除する。local 側から「rm -rf」コマンドを送出して、folder を削除するので、高速に削除できる。

#### CAD の起動

選択しているフォルダをカレントディレクトリとして CAD を起動する。  
configTreeFoam で設定されている CAD が起動する。

#### meshViewer の起動

meshViewer を起動する。

#### salomeMeca の起動

選択しているフォルダをカレントディレクトリとして salome を起動する。  
configTreeFoam で設定されている salome が起動する。

### 2) solver 部のポップアップメニュー

#### 開く

選択しているフォルダをファイルマネージャ (nautilus) で開く。  
これにより、このフォルダ内のファイル操作ができる。

#### 端末の起動

選択しているフォルダをカレントディレクトリとして、端末を開く。  
この端末は、FOAM 端末ではない。

#### 選択 case として設定

選択しているフォルダを解析 case (📁付きフォルダ) として設定する。

#### FOAM 端末の起動

選択しているフォルダをカレントディレクトリとして、FOAM 端末を開く。  
OpenFOAM 用に環境設定された端末を起動する。

#### gridEditor 起動

選択しているフォルダをカレントディレクトリとして、gridEditor を起動する。

### 3) 結果部のポップアップメニュー

#### 開く

選択しているフォルダをファイルマネージャ (nautilus) で開く。  
これにより、このフォルダ内のファイル操作ができる。

#### 端末の起動

選択しているフォルダをカレントディレクトリとして、端末を開く。  
この端末は、FOAM 端末ではない。

#### gridEditor 起動

選択しているフォルダをカレントディレクトリとして、gridEditor を起動する。

**plotWatcher 起動**

解析 case の solver を実行中（実行後）、このボタンをクリックすると plotWatcher が起動するので、solver 実行時の残渣が確認できる。

**meshViewer の起動**

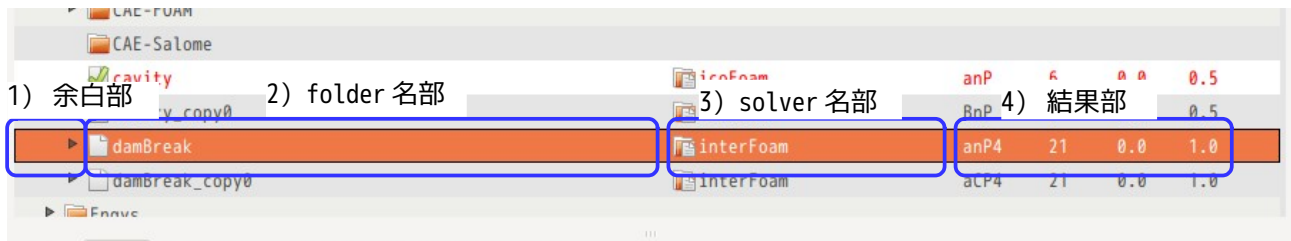
meshViewer を起動する。

**5-1-3. ダブルクリックによる操作**

ダブルクリックによる操作は、メニューを選択する必要がない為、素早く処理を行う事ができる。

この処理は、ダブルクリックする場所によって、処理が異なってくる。

操作対象は、ダブルクリックした行が操作対象になるが、ダブルクリックすると必然的にその行が選択される。その処理内容は以下。

**1) 余白部をダブルクリック**

ダブルクリックした行を解析 case (☑付きのフォルダ) として設定する。

**2) folder 名をダブルクリック**

ダブルクリックした folder をファイルマネージャ (nautilus) で開く。

**3) solver 名をダブルクリック**

ダブルクリックした case の controlDict を editor で開く。ただし、ダブルクリックした行が解析 case ではない場合は、「解析 case として設定するか」の問い合わせを行い、解析 case として設定した後、controlDict が開く。

**4) 結果部をダブルクリック**

ダブルクリックした case をカレントディレクトリとして、paraFoam を起動する。ただし、ダブルクリックした行が解析 case ではない場合は、「解析 case として設定するか」の問い合わせを行い、解析 case として設定した後、paraFoam が起動する。

## 6. 基本的な操作方法の例

TreeFoamの基本的操作の例として、tutorialsのcavityとdamBreakを実行してみる。

### 6-1. 天井駆動のキャビティ流れ (cavity) の操作例

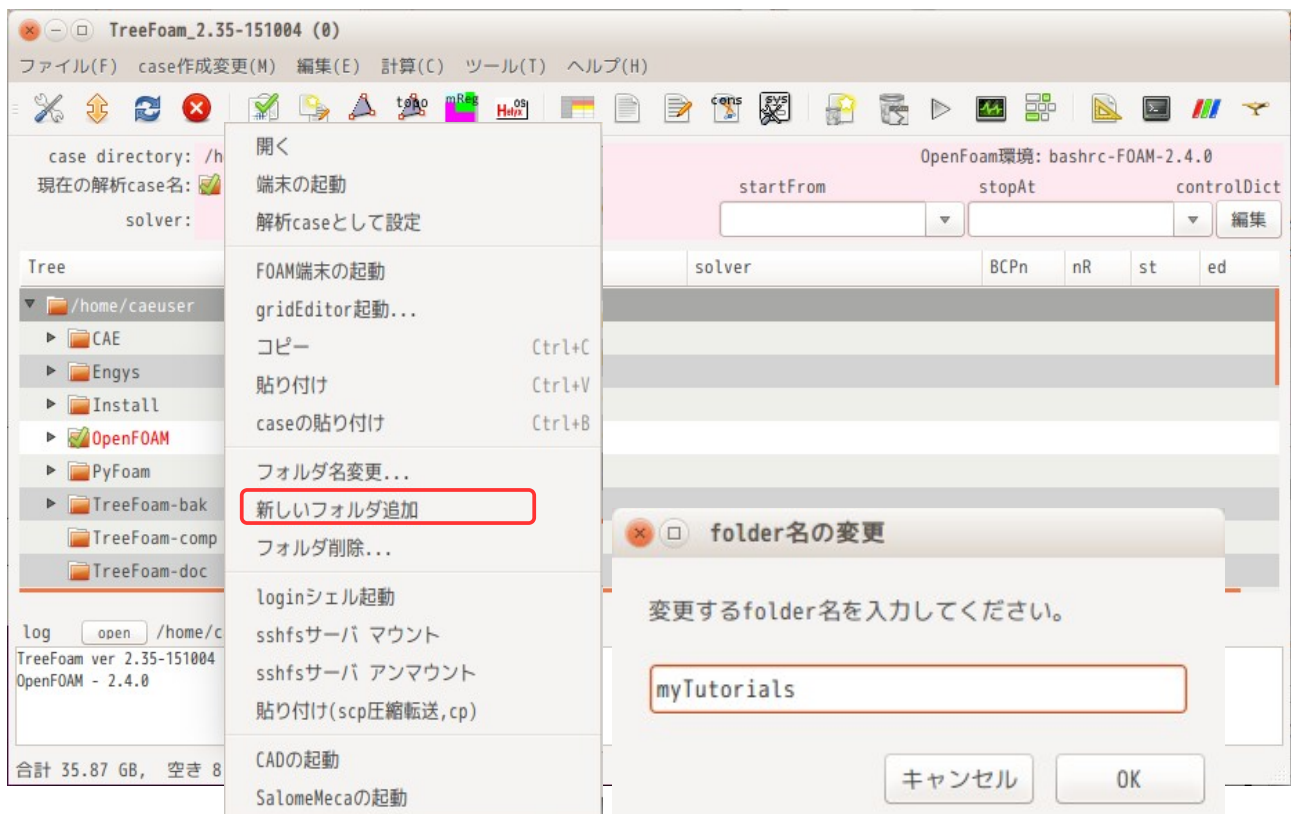
このキャビティ流れの計算を以下の様に実行してみる。まず、tutorialsのcavityを、新しく作成したフォルダ内にコピーして、この中でcavityを計算する。


- 1) \$HOME 直下に計算用のフォルダ「myTutorials」を作成する
- 2) tutorialsの「cavity」を「myTutorials」フォルダにコピーする。
- 3) blockMeshを実行してメッシュを作成する。
- 4) solver「icoFoam」を実行する。
- 5) paraFoamで結果を確認する。
- 6) 境界条件を変更して再計算
- 7) constant、systemフォルダの内容確認
- 8) controlDictの内容確認

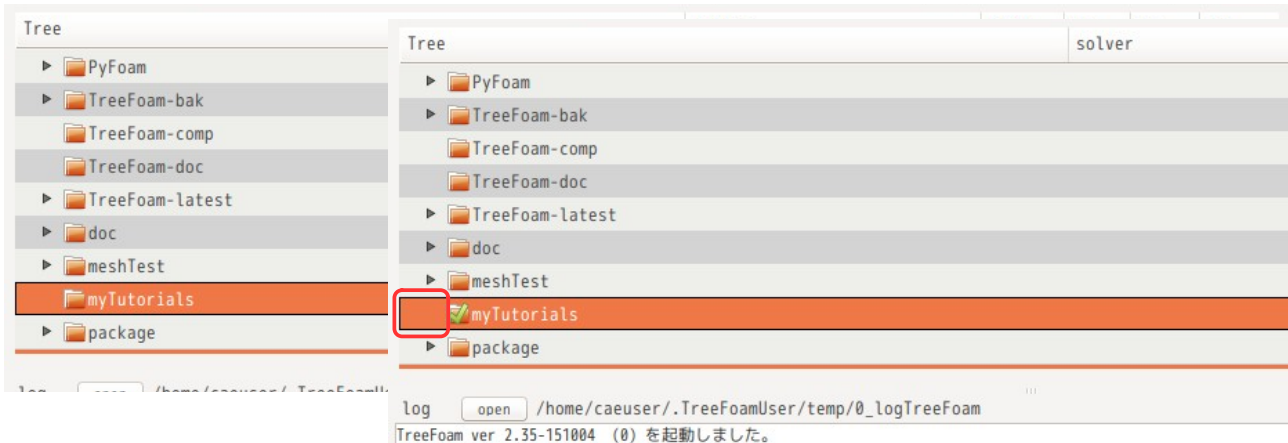
#### 6-1-1. myTutorials フォルダ作成

\$HOME 直下に「myTutorials」を作成する。


rootDir (新しいフォルダを作成する場所) を選択して、右クリックでポップアップメニューを表示させ、「新しいフォルダ追加」をクリックして、新しいフォルダ名「myTutorials」を、現れたダイアログに入力してフォルダを作成する。



この後、下図の様に「myTutorials」フォルダが追加されている。  
myTutorials フォルダ名の左側 (   部) をダブルクリックして  マークを付けておく。

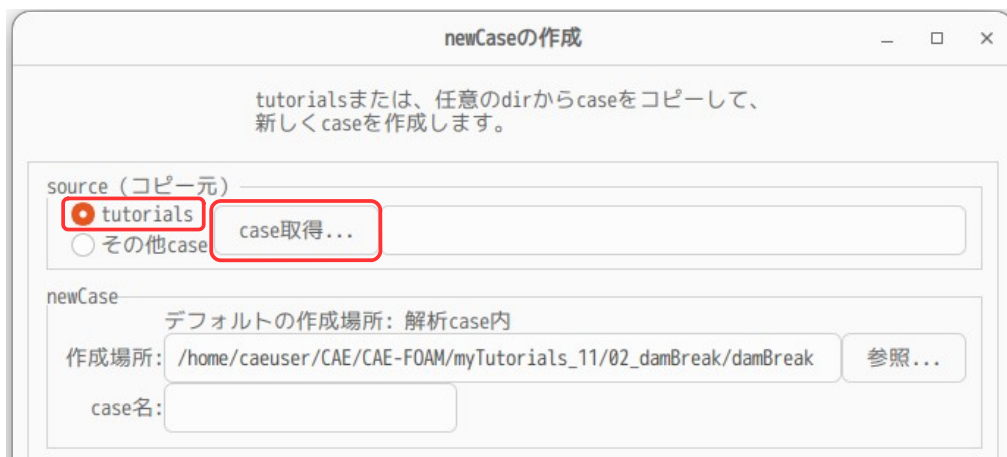


### 6-1-2. tutorials の「cavity」を「myTutorials」フォルダにコピー

myTutorials フォルダができ上がったので、このフォルダに tutorials の「cavity」をコピーする。  
まず、 ボタンをクリックする。



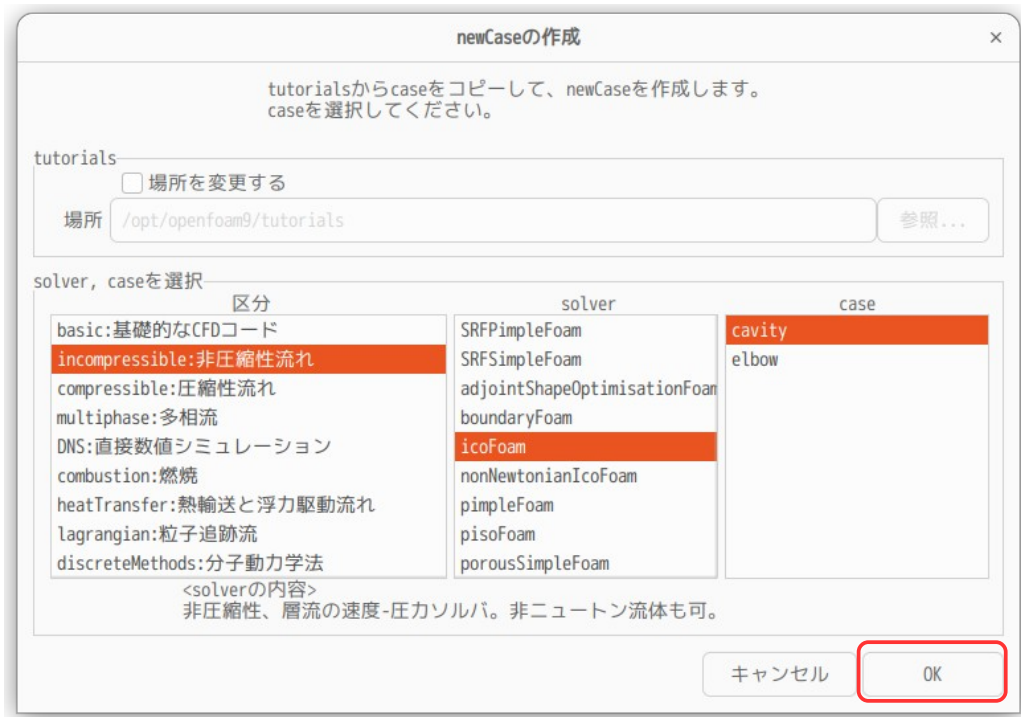
この後、以下の画面が現れる。この画面上で、「tutorials」ラジオボタンが選択されていることを確認の上、「case 取得...」ボタンをクリックする。



次の画面上で、区分「incompressible:非圧縮性流れ」、solver「icoFoam」、case「cavity」を選択し、「OK」ボタンをクリックする。  
(OF-11, 12, 13 の場合は、「legacy/incompressible」、「icoFoam」、「cavity/cavity」を選択。)

ここで、もし、solver 名等が表示されない場合は、tutorials の場所が間違っているので、tutorials の「場所を変更する」をチェックし、「参照...」ボタンで tutorials の場所を指定すれば、その内容が表示される。また、solver 名を選択した時点で、その solver の処理内容が表示されるので、参考になる。



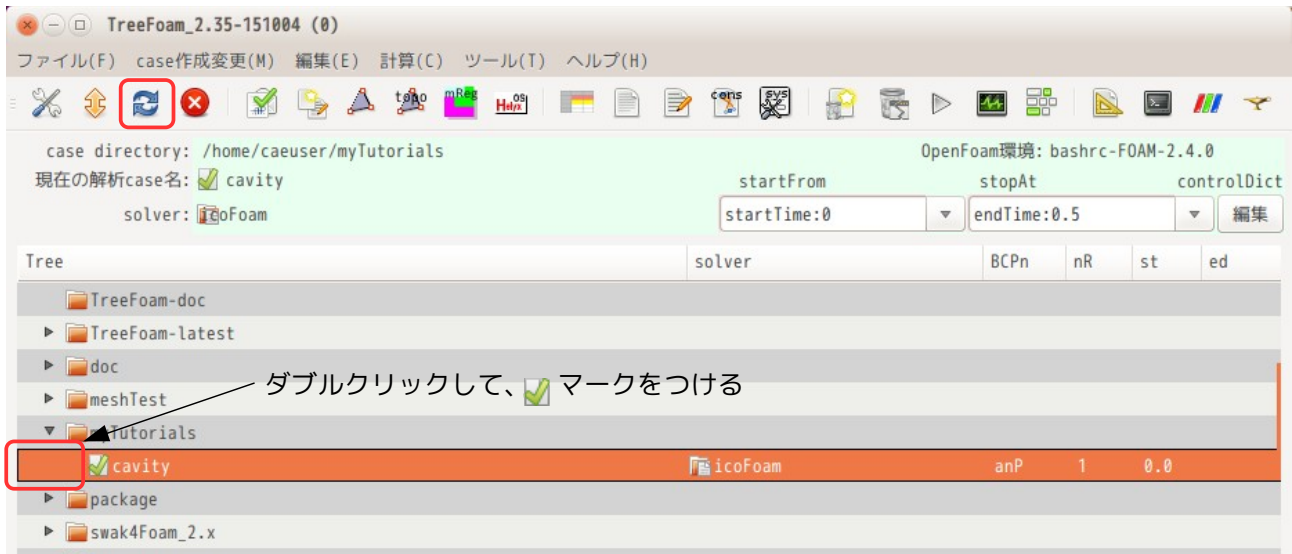


以上で、tutorialsの「cavity」が選択できたことになる。下図の□部に tutorials「cavity」の directory が取得できている。

この後、作成場所と case 名を確認して、「コピー開始」ボタンをクリックする事によって、「cavity」が「myTutorials」フォルダ内にコピーされる。  
コピー後は、「閉じる」ボタンでダイアログを閉じておく。



この後、下図の様に🔄ボタンをクリックして、ツリー構造を再読み込みし、「myTutorials/cavity」に✅マークを付けておく。



### 6-1-3. blockMesh 作成

case 内に blockMeshDict が準備されているので、blockMesh を実行してメッシュを作成する。

⚠ ボタンをクリックして、現れた画面内の「blockMesh 実行」ボタンをクリックする事で、blockMesh を作成する事ができる。

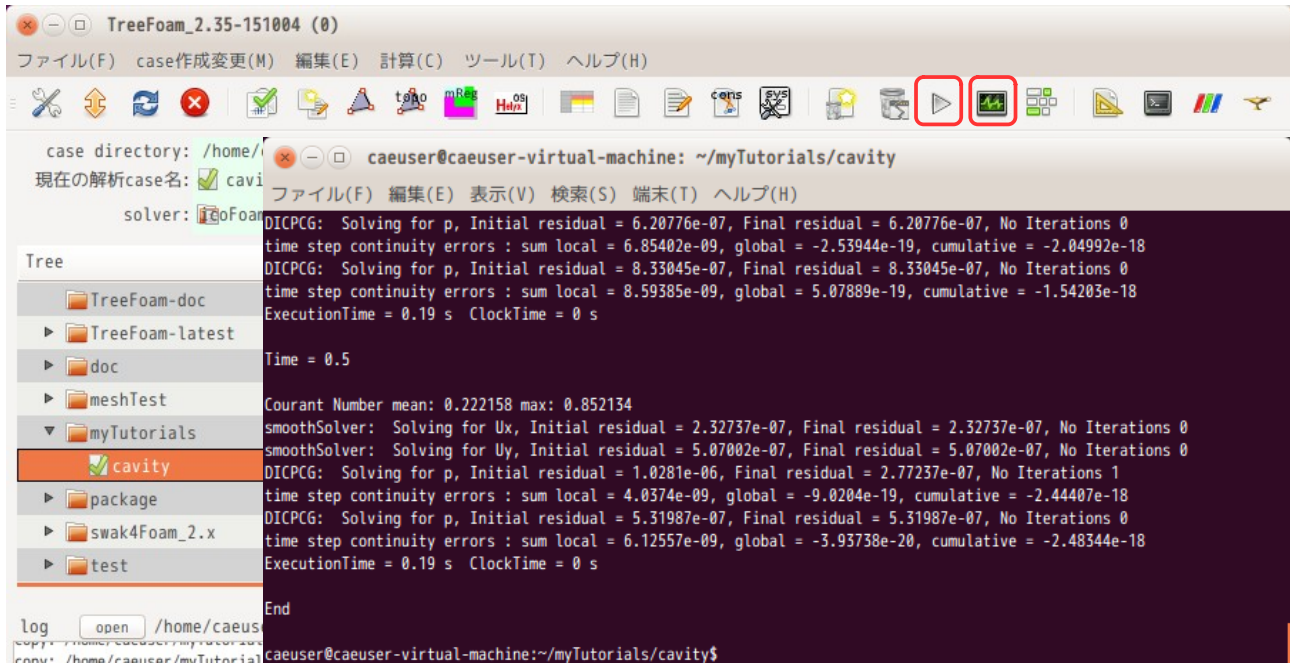



### 6-1-4. icoFoam の実行

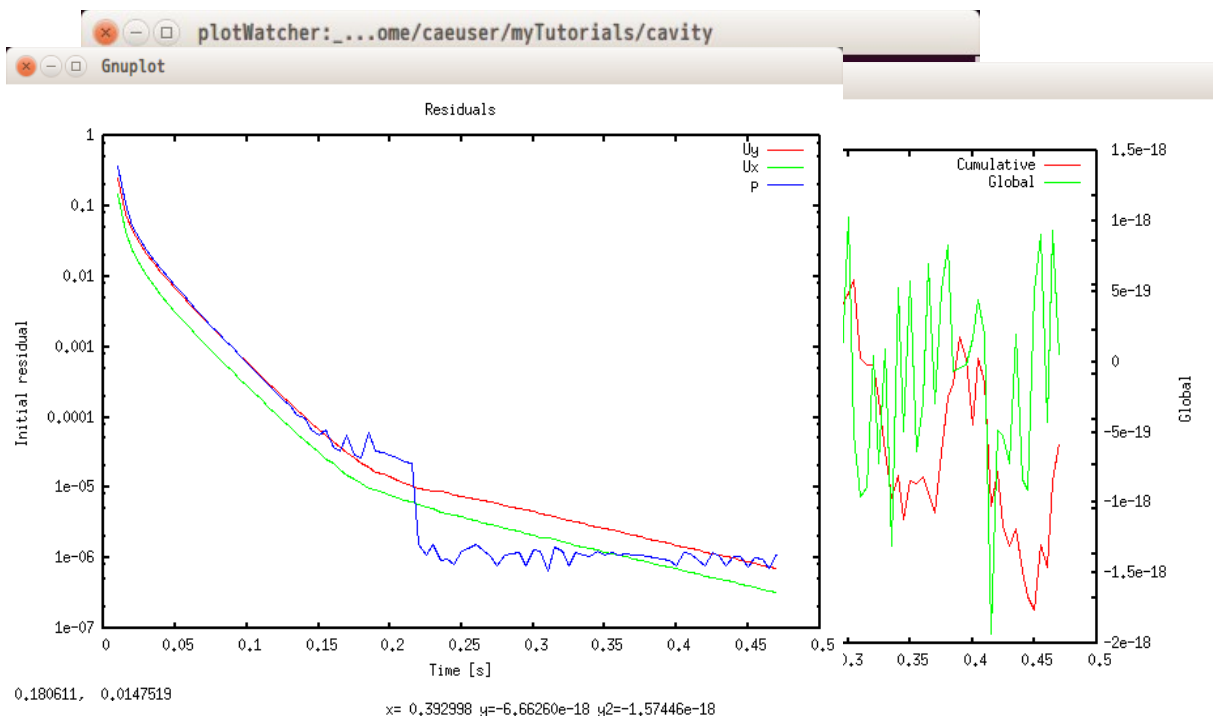
controlDict に設定されている solver (今回の場合、icoFoam) を実行する為には、▶ ボタンをクリックす

る事で、実行できる。


▶ ボタンをクリックすると、下図の様に FOAM 端末が起動し、この中で icoFoam が実行される。solver 実行時の log は、cavity フォルダ内の「solve.log」ファイルに残されているので、実行後でも log が確認できる。



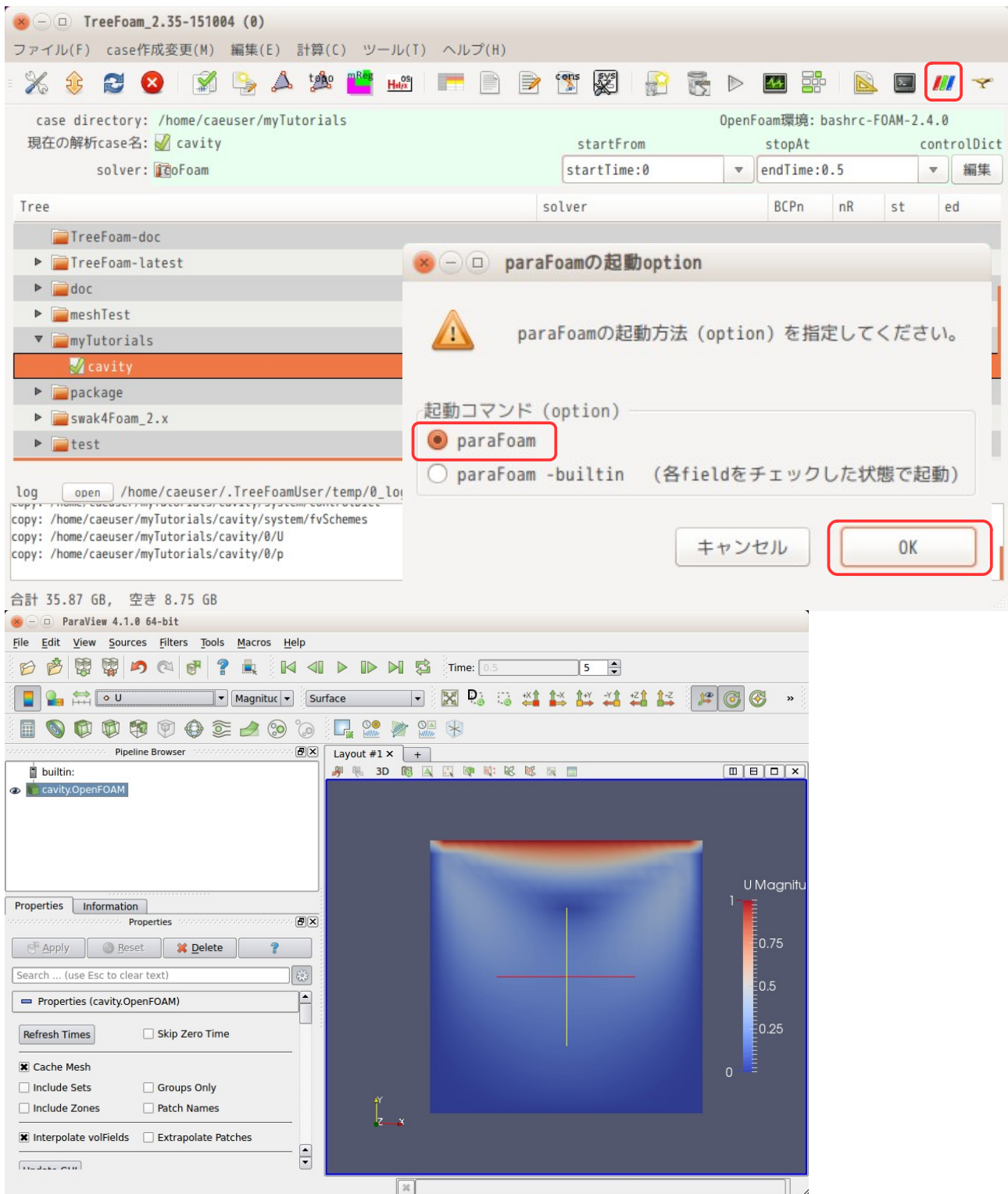
また、実行中（実行後）の残渣を確認するには、 ボタンをクリックすると、残渣が以下の様に表示される。端末を起動した上で plotWatcher を起動している為、閉じる時は端末も閉じておく。




### 6-1-5. paraFoam による結果の確認

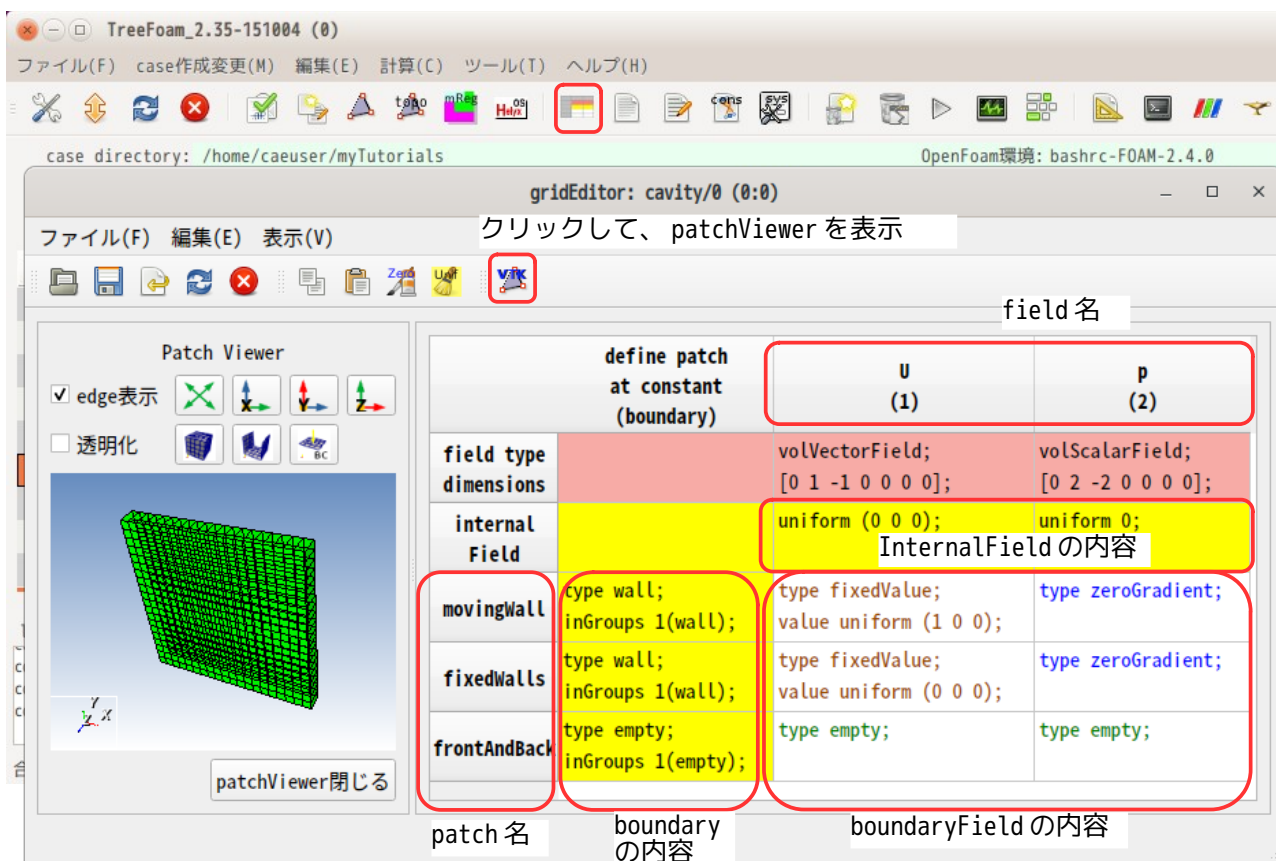
結果の確認は、paraFoam を起動し確認する。TreeFoam 上から  ボタンをクリックして、「paraFoam」を選択後、「OK」ボタンをクリックする事で、paraFoam が起動する。





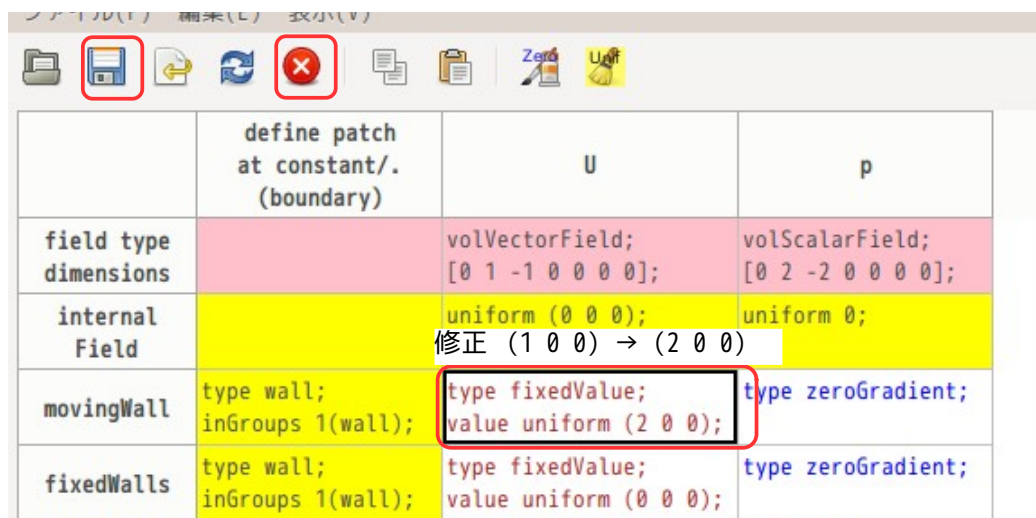
#### 6-1-6. 境界条件を変更する場合



今の境界条件 (boundary、各 field の internalField と boundaryField の内容) は、TreeFoam 上の  ボタンをクリックする事で gridEditor が起動し、これらが確認できる。






gridEditor 上からは、上図の   内の項目全て編集ができる。

境界条件を変更する為に、U fieldの movingWall の内容を以下の様に変更してみる。修正は、該当するセルを選択し、「F2」キーを押すかダブルクリックして、セル内容を修正する。





上図の様に修正後、 をクリックして修正内容を保存し、 をクリックして、gridEditor を終了する。


gridEditor の終了は、必ず  ボタンをクリックして終了させる。これは、 ボタンで終了させると、終了処理を行って、正常に gridEditor を閉じる為。(window 上部の  ボタンで直接 window を閉じると、終了処理を行わず、強制的に window を閉じてしまう。)

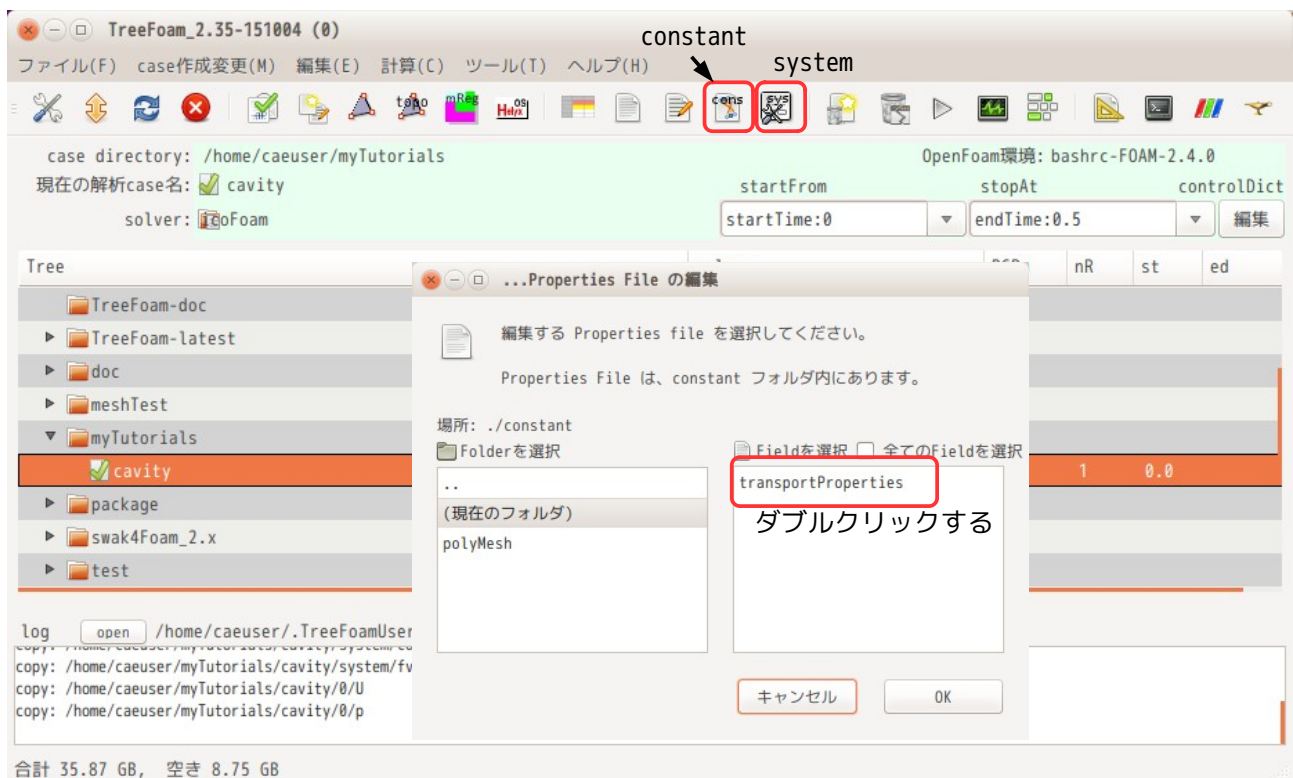
以上の操作で、U fieldの movingWall patch 内容が修正された事になる。  
また、この gridEditor は、ファイルの書式が ascii や binary、圧縮ファイルでも読み込み、保存ができるので、ファイルの書式を意識せず扱う事ができる。

### 6-1-7. constant、system フォルダの内容確認

TreeFoam 上から、constant フォルダと system フォルダの内容は、それぞれ、 ボタンと  ボタンをクリックする事で、そのフォルダ内のファイル名のリストが表示されるので、ここで確認できる。この画面上で、ファイル名をダブルクリックすると、editor が起動し、その内容を確認する事ができる。

この画面上でファイルを editor で開く場合、そのファイルが binary ファイルであっても、binary を ascii に変換して、editor で開き、編集が可能になる。保存する場合は、asii を binary に変換して保存する。この為、ファイルの書式に関わらず、ファイルの内容が確認・編集できる。

今回の case 内では、constant/transportProperties を使っているので、 ボタンをクリックしてこの内容を確認してみる。



transportProperties のファイル名をダブルクリックして、この内容を editor で確認した結果が下図になる。

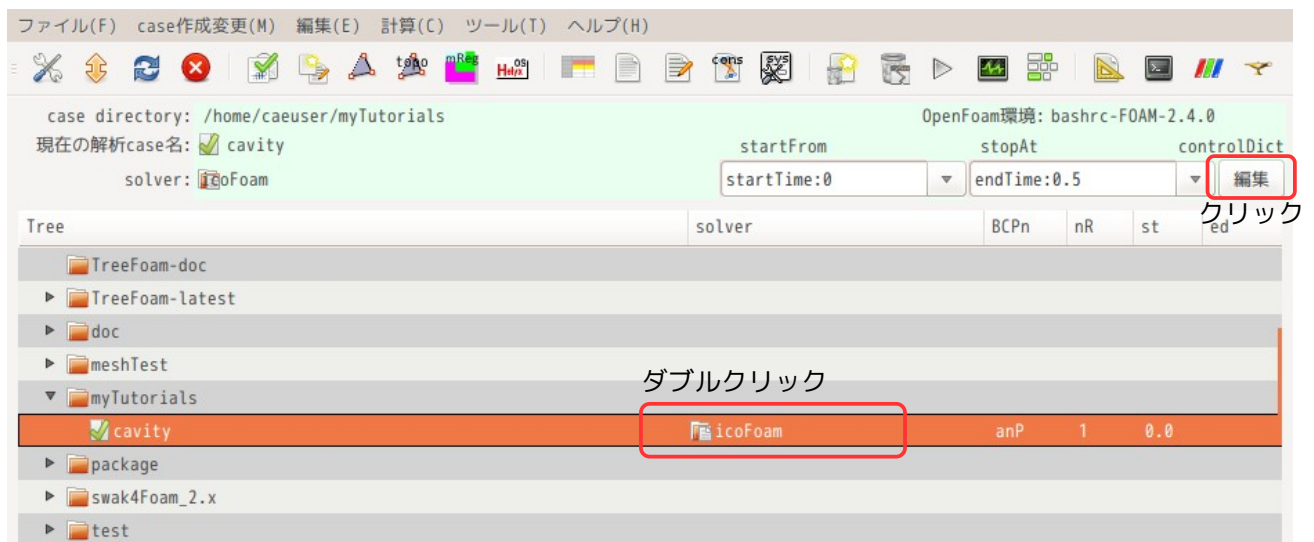
```

transportProperties
1 /*-----* C++ *-----*/
2 | =====
3 | \ \ / F i e l d | OpenFOAM: The Open Source CFD Toolbox
4 | \ \ / O p e r a t i o n | Version: 2.4.0
5 | \ \ / A n d | Web: www.OpenFOAM.org
6 | \ \ M a n i p u l a t i o n |
7 /*-----*/
8 FoamFile
9 {
10     version      2.0;
11     format        ascii;
12     class         dictionary;
13     location      "constant";
14     object        transportProperties;
15 }
16 // *****
17
18 nu              nu [ 0 2 -1 0 0 0 ] 0.01;
19
20
21 // *****

```

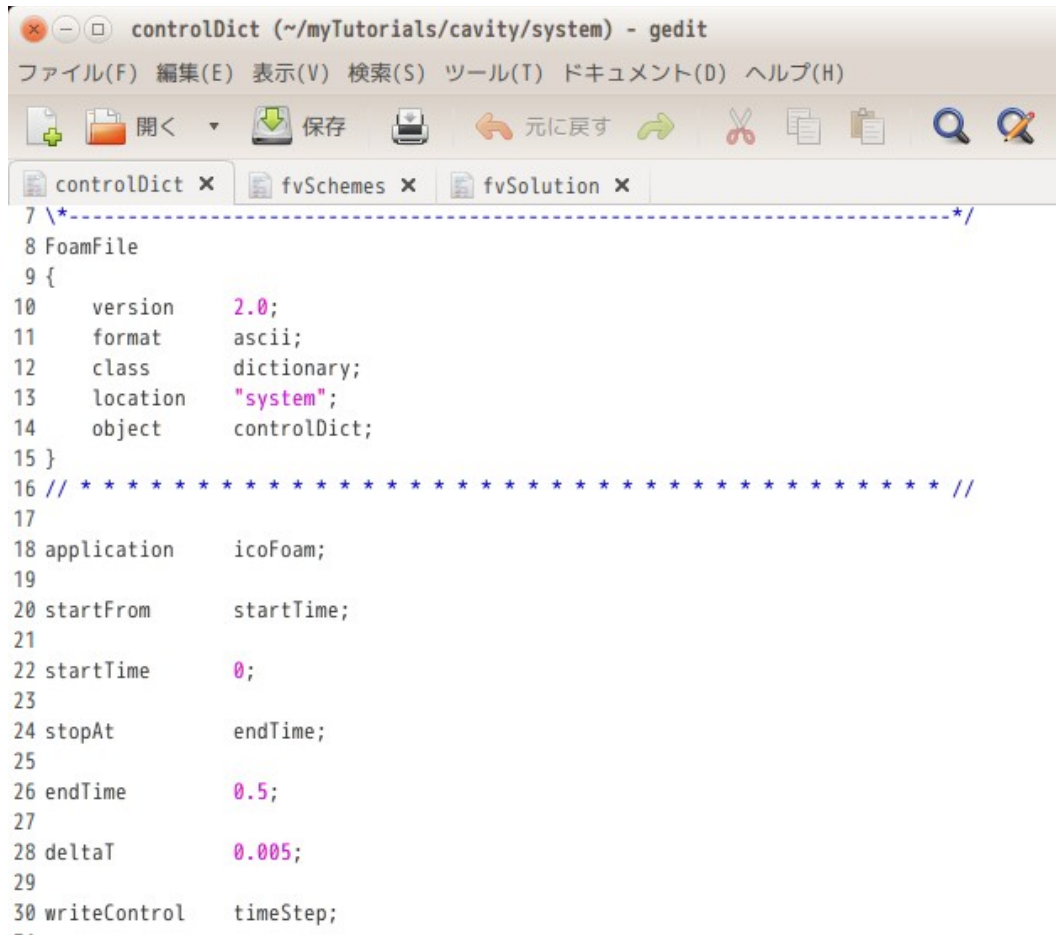
#### 6-1-8. controlDict の内容確認

計算方法の制御を行っている controlDict の内容の確認は、使用頻度が高いので、TreeFoam 上から直ぐに確認できる様にしている。その方法は、下図の「編集」ボタンをクリックするか、solver 名「icoFoam」をダブルクリックする事で、controlDict の内容が editor で表示され、これを確認する事ができる。



以下が、controlDict の内容になる。同時に fvSchemes と fvSolution も同時に確認できる。






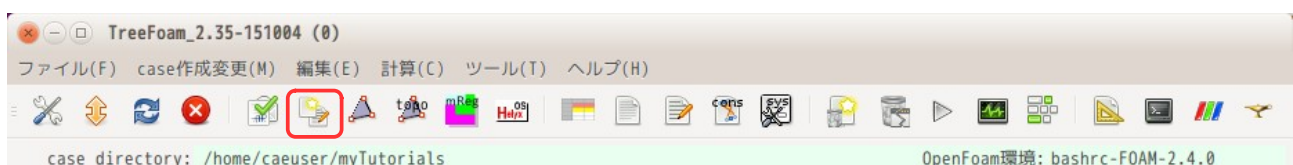
## 6-2. ダムの決壊 (damBreak) の操作例

ダムの決壊を以下の様に行ってみる。tutorials の damBreak をフォルダ「myTutorials」内にコピーして、この中で、実行する。

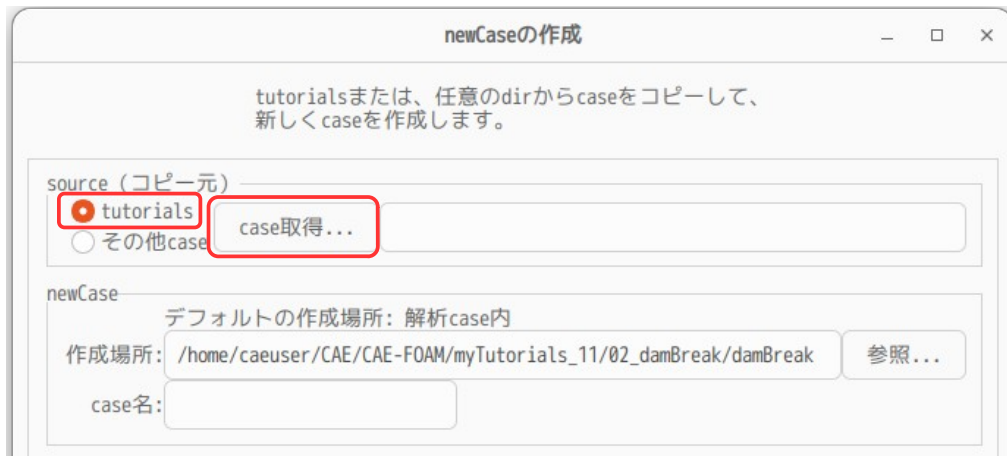
- 1) tutorials の「damBreak」を「myTutorials」フォルダにコピーする。
- 2) blockMesh を実行してメッシュを作成する。
- 3) setFields で alpha.water のフィールドに値をセット
- 4) 境界条件の確認
- 5) solver 「interFoam」を実行する。
- 6) paraFoam で結果を確認する。
- 7) 並列計算の確認

### 6-2-1. tutorials の「damBreak」を「myTutorials」フォルダにコピー

tutorials の「damBreak」をコピーする為に、TreeFoam 上の  ボタンをクリックする。

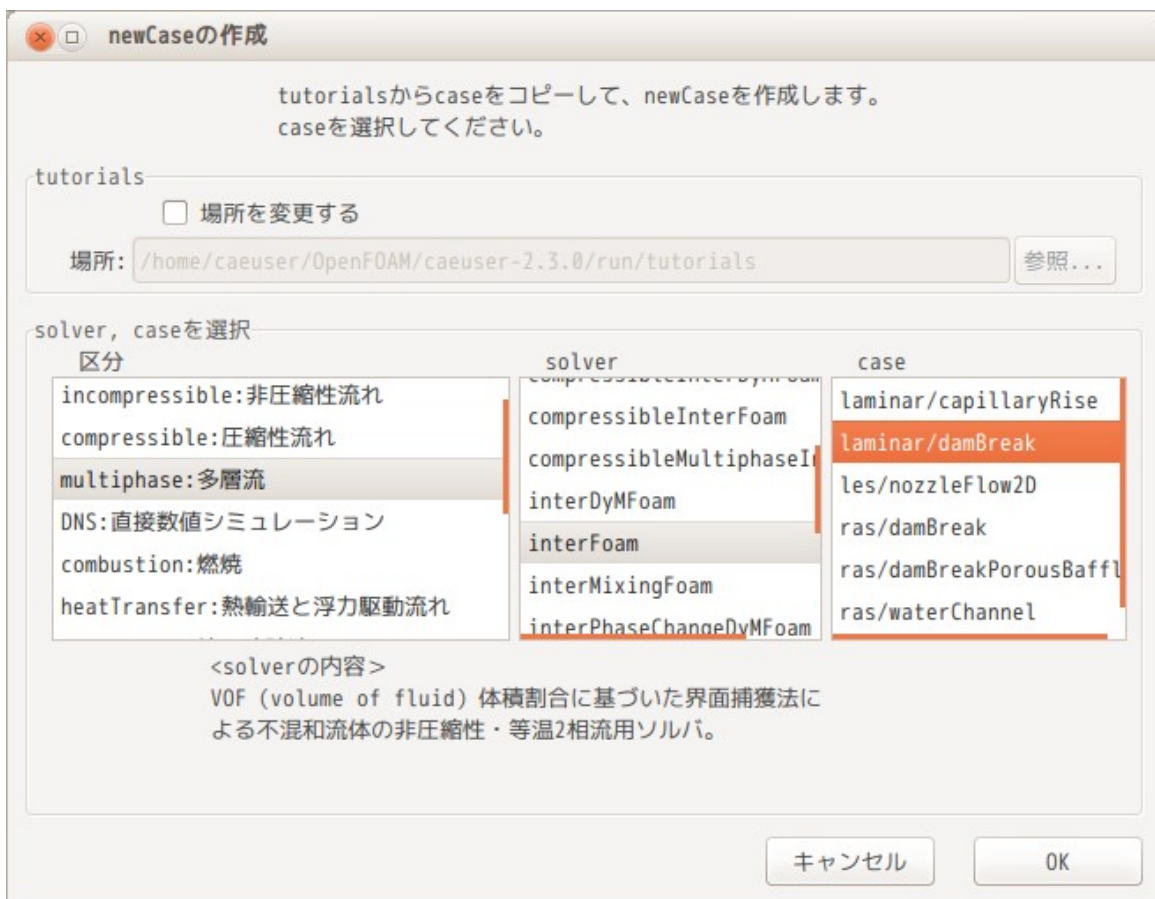


この後、以下の画面が現れるので、この画面上で、「tutorials」ラジオボタンが選択されていることを確認の上、「case 取得...」ボタンをクリックする。



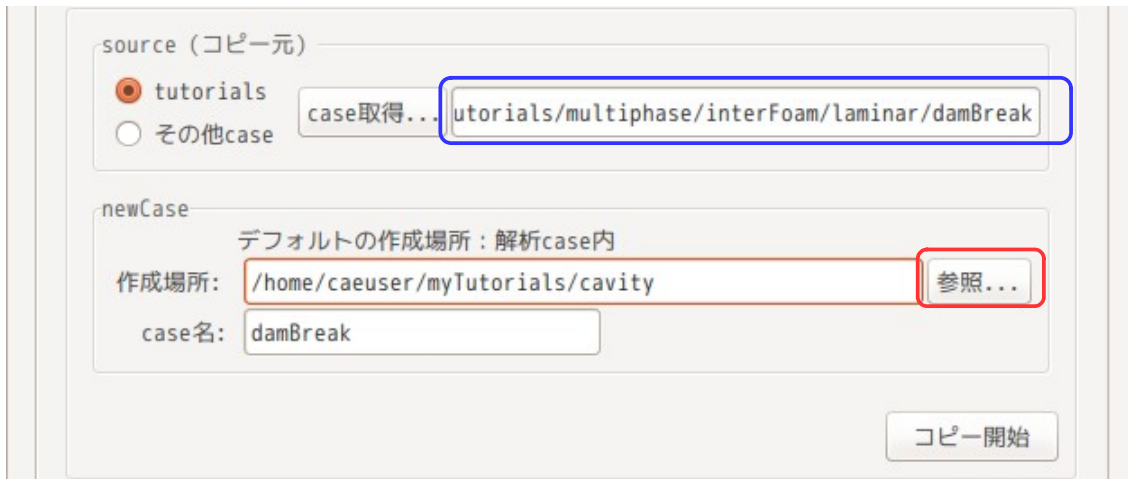
この後、以下の画面が現れるので、区分「multiphase:多層流」、solver「interFoam」、case「laminar/damBreak」を選択し、「OK」ボタンをクリックする。  
(OF-11, 12, 13の場合は、「foamRun:一般的なCFD」、「incompressibleVoF」、「damBreakLaminar」を選択する。)

尚、solver を選択した時点で、画面下部にその solver の内容が表示されるので参考になる。

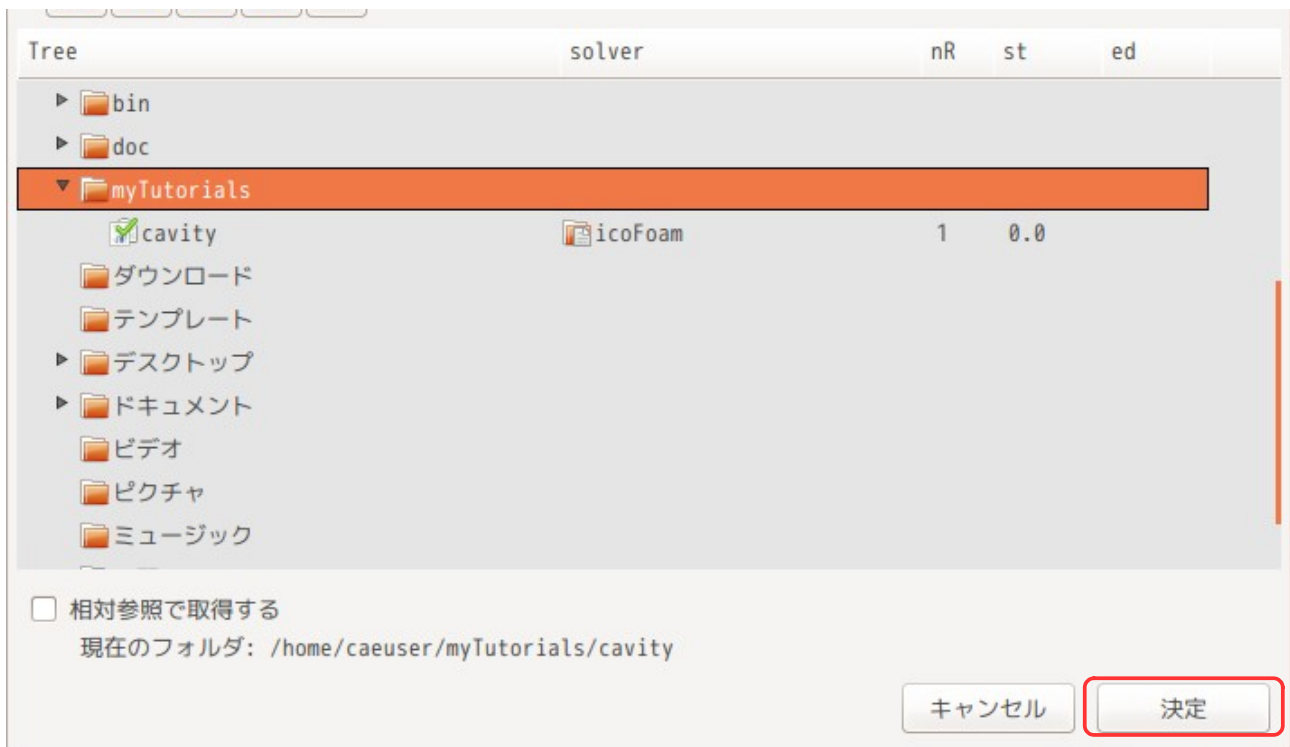


この操作により、tutorials内の「damBreak」のdirectryが   内の様に取得できる。

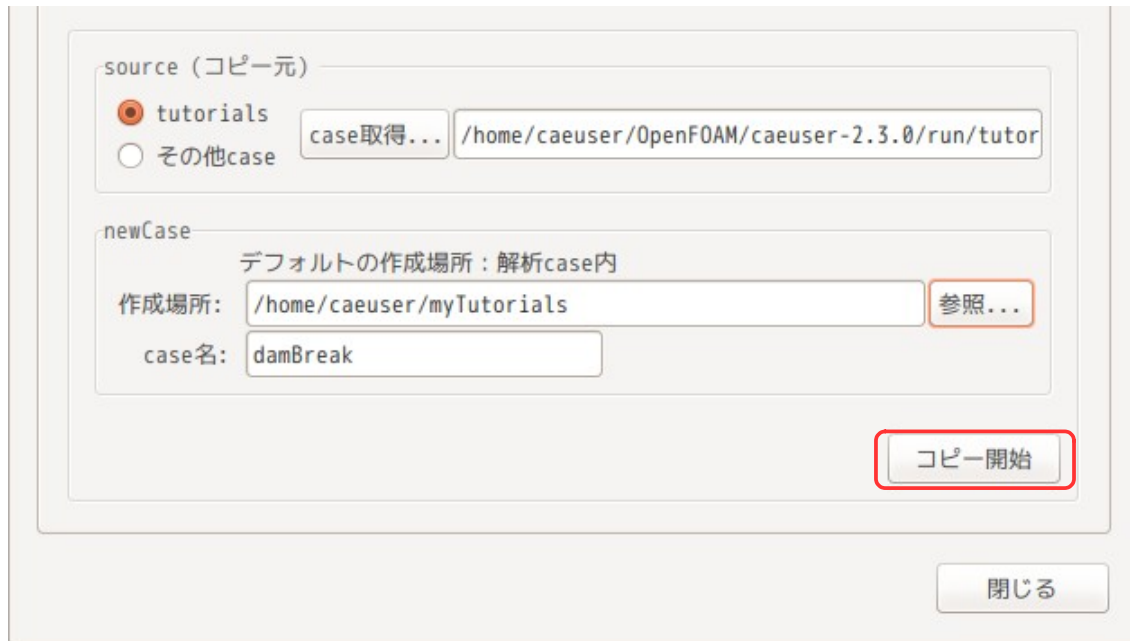
この後、newCaseの作成場所が「/home/caeuser/myTutorials/cavity」になっているので、これを、「参照...」ボタンをクリックして、「home/caeuser/myTutorials」に変更する。  
デフォルトのコピー先は、☒マーク付きのフォルダに設定されており、これがcavityになっている。  
(☒マーク付きフォルダを予め「myTutorials」フォルダに設定しておけば、この操作は不要になる。)






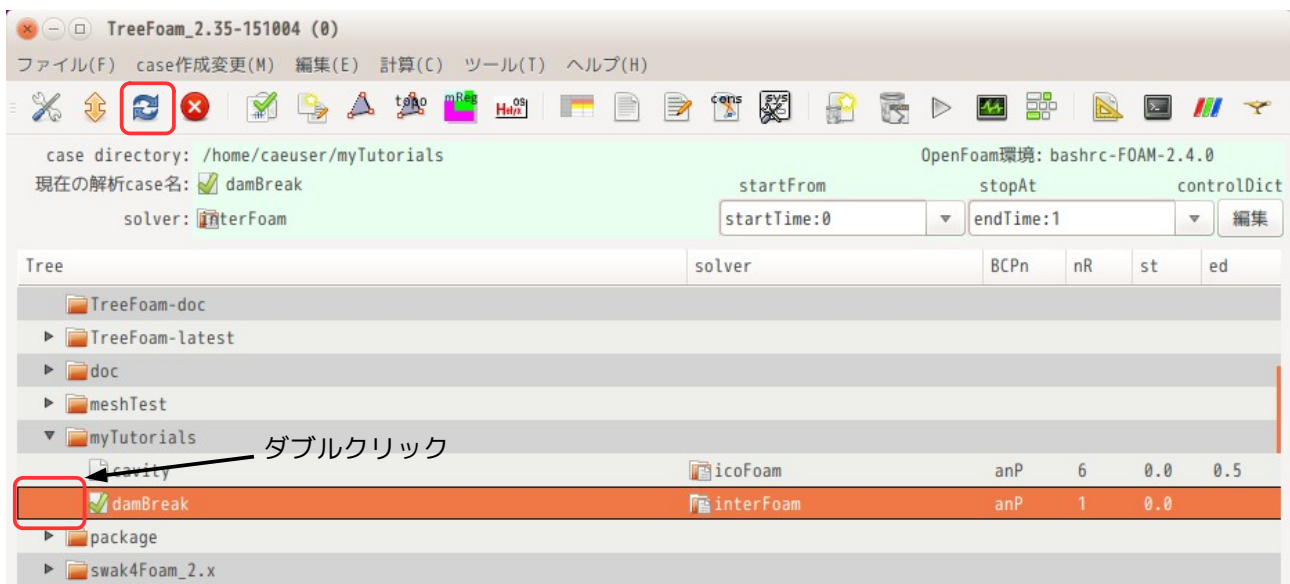
「参照...」ボタンをクリックすると、以下の画面が現れるので、「myTutorials」を選択し、「決定」ボタンをクリックして、画面を閉じる。



newCase の作成場所「/home/caeuser/myTutorials」が取得できたので、「コピー開始」ボタンをクリックして、damBreak をコピーする。





コピー後は、「閉じる」ボタンをクリックして、「新しい case の作成」画面を閉じておく。  
画面を閉じた後、下図の様に  ボタンをクリックして、ツリー構造を再読み込みし、 部をダブルクリックして、「myTutorials/damBreak」に  マークを付けて解析 case に設定しておく。

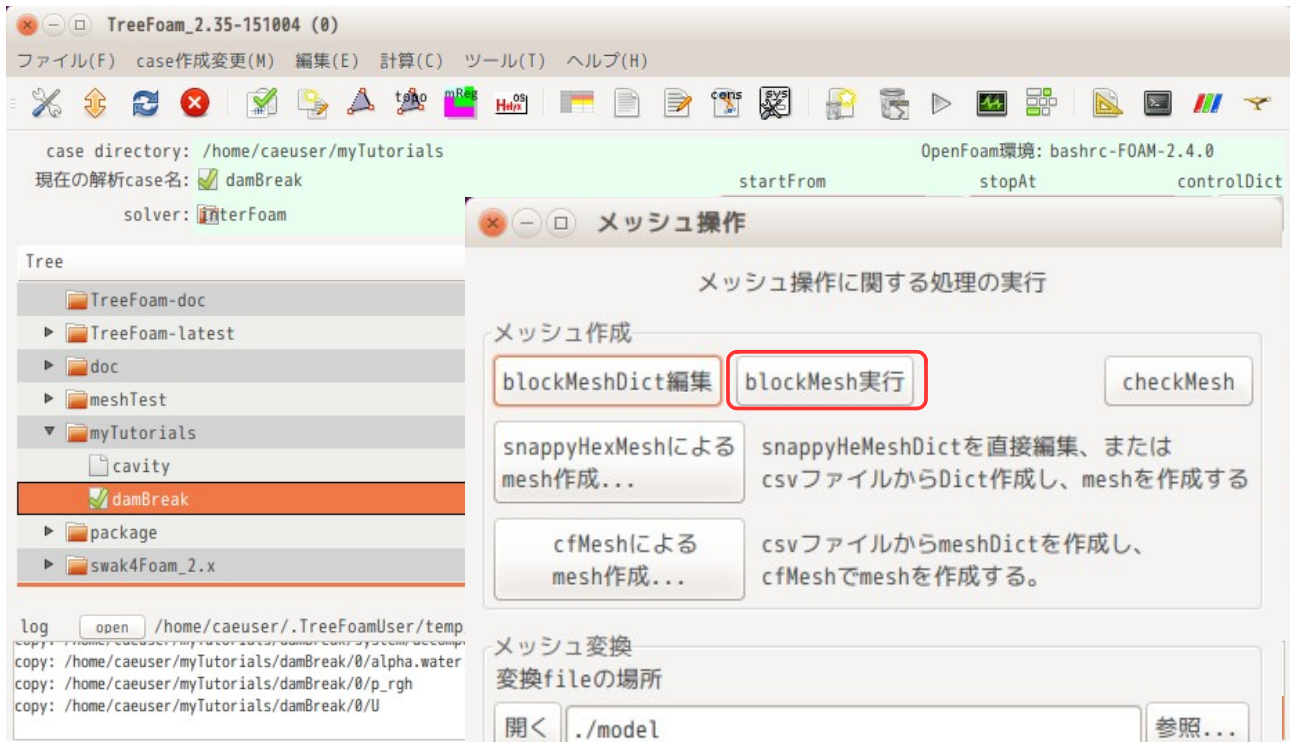


damBreak フォルダ内に「Allrun」があるので、これを実行すれば、case が完成し実行できるが、ここでは、手動で順番に実行してみる。  
まずは、case 内に「0」フォルダがあるかどうか確認し、無い場合は、「0.orig」フォルダをコピーして「0」フォルダを作成しておく。

### 6-2-2. blockMesh の作成

blockMeshDict は「constant/polyMesh」フォルダに準備されているので、blockMesh コマンドを実行すれば済む。この為、 ボタンをクリックして、現れた画面内の「blockMesh 実行」ボタンをクリックする事で blockMesh コマンドが実行できる。  
でき上がったメッシュは、 ボタンをクリックすると、paraFoam が起動するので、これで確認できる。




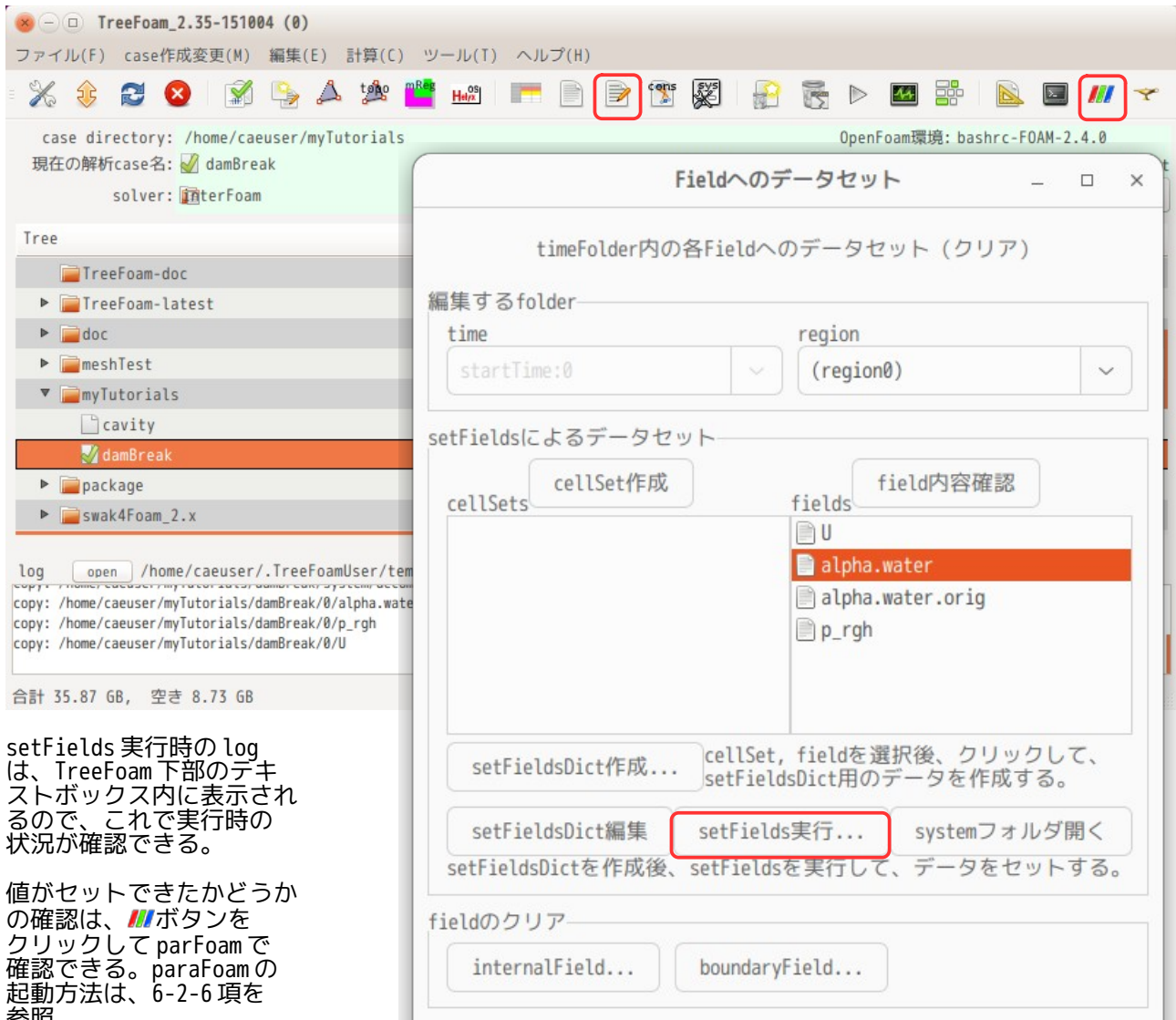


### 6-2-3. setFields で値をセット


damBreak は、「alpha.water」field に値をセットする必要がある。この field が、case 内には存在しない場合は、「damBreak/0/alpha.water.org」をコピーして「damBreak/0/alpha.water」に名称を変更しておく。

myTutorials	→	myTutorials	
damBreak		damBreak	
0		0	
U		U	
alpha.water.org		alpha.water.org	
p_rgh		alpha.water → コピーしてこの field を作成する。	
		p_rgh	

この後、setFields を実行する事になる。setFields 実行は、TreeFoam 上の  ボタンをクリックして、現れた画面上で、「setFields 実行...」ボタンをクリックする



#### 6-2-4. 境界条件の確認

damBreak の境界条件 (boundary, 各 field の internalField と boundaryField の内容) は、TreeFoam 上の  ボタンをクリックする事で gridEditor が起動し、これらが確認できる。

The screenshot shows the TreeFoam 2.25-150308 application window. The main window displays the case directory as /home/caeuser/myTutorials and the current case name as damBreak. The solver is set to interFoam. The OpenFoam environment is bashrc-FOAM-2.3.1. The startFrom and stopAt fields are set to startTime:0 and endTime:1 respectively. The controlDict field is set to edit. The Tree view shows the case structure with damBreak selected. The gridEditor window is open, showing the field names and their definitions.

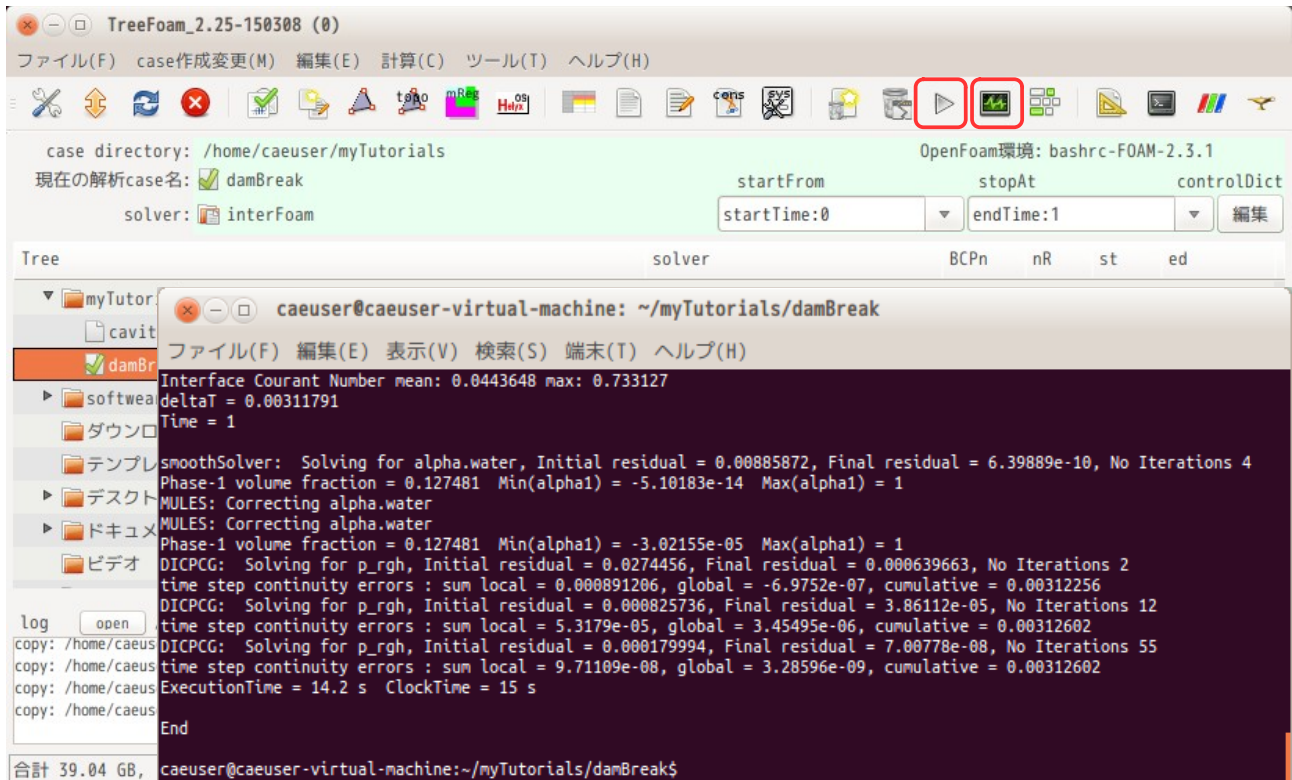
field 名	U	alpha.water	alpha.water.org	p_rgh
field type	volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions	[0 1 -1 0 0 0];	[0 0 0 0 0 0];	[0 0 0 0 0 0];	[1 -1 -2 0 0 0];
internal Field	uniform (0 0 0);	nonuniform List<scalar> 2268 ( 1 1...	uniform 0;	uniform 0;
LeftWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type fixedFluxPressure; value uniform 0;
rightWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type fixedFluxPressure; value uniform 0;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type fixedFluxPressure; value uniform 0;
atmosphere	type patch;	type pressureInletOutletVelocity; value uniform (0 0 0);	type inletOutlet; inletValue uniform 0; value uniform 0;	type totalPressure; p0 uniform 0; U U; phi phi; rho rho; psi none; gamma 1; value uniform 0;
defaultFaces	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;


patch 名    boundary の内容    baoundaryField の内容

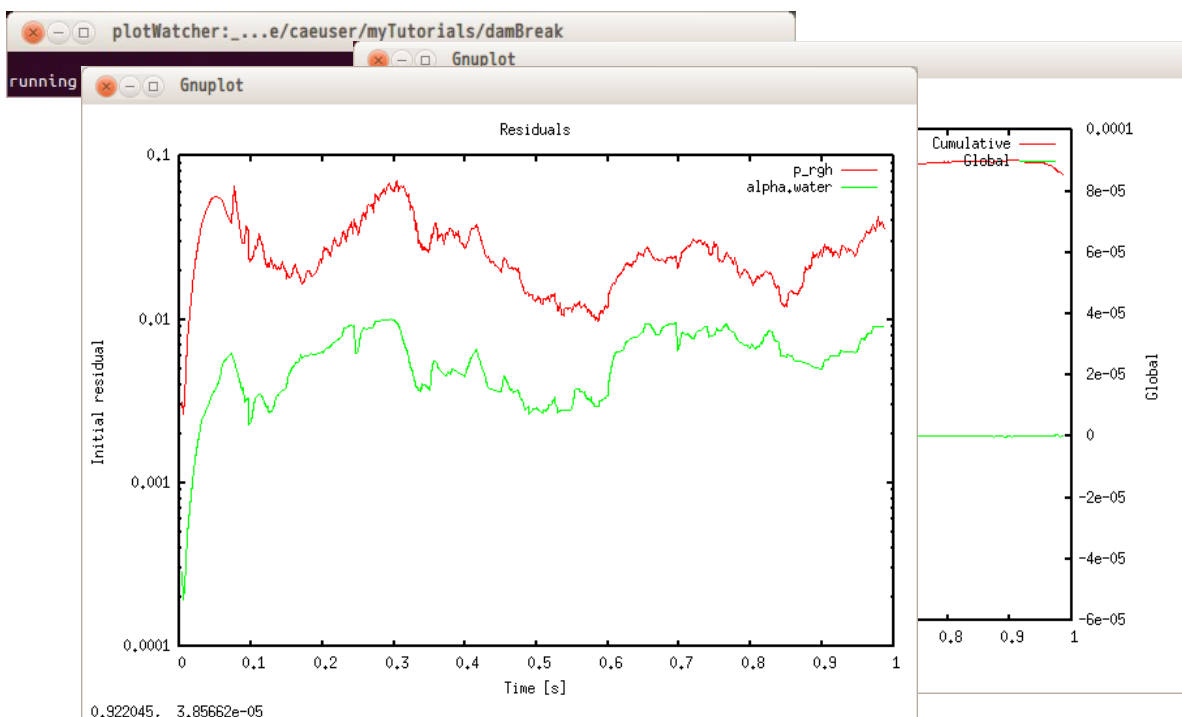
### 6-2-5. interFoam の実行

今の境界条件と初期値で計算させるには、TreeFoam 上の ▶ ボタンをクリックする事で、controlDict 内に記述されている solver : interFoam を実行する事ができる。


また、実行中（実行後）の残渣を確認する為には、TreeFoam 上の 残渣 ボタンをクリックする事で、残渣を確認する事ができる。以下が solver : interFoam を実行した結果になる。（FOAM 端末を起動し、この中で interFoam を実行している。）



下図が実行中（実行後）に  ボタンをクリックして、plotWatcher 起動して残差を確認した結果になる。尚、残差の画面を閉じる時は、端末内で plotWatcher が動いているので、端末を閉じる事により、plotWatcher を停止させる事ができる。

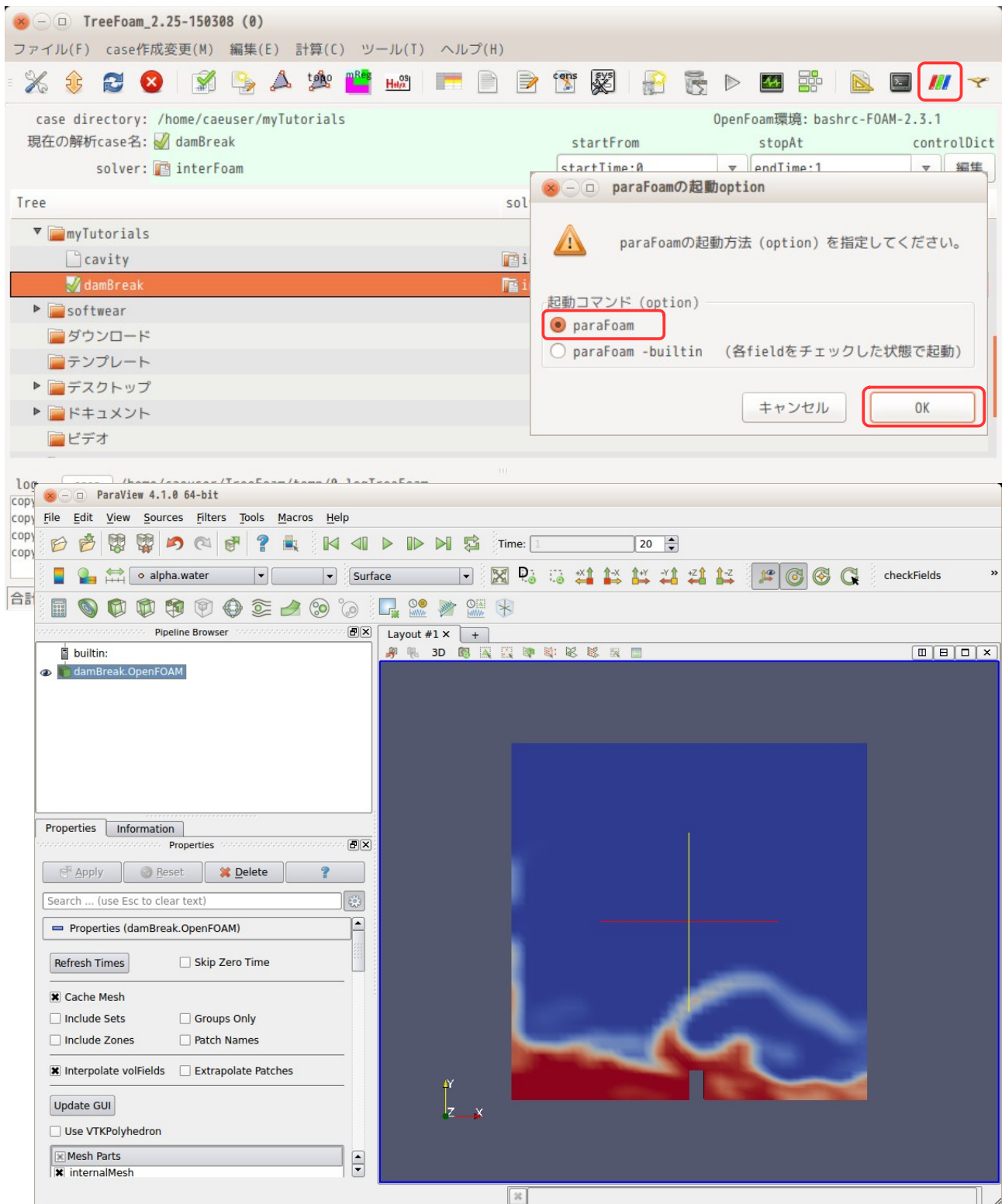


## 6-2-6. 結果の確認


計算結果を確認するためには、TreeFoam 上で  ボタンをクリックして、option 無しの「paraFoam」を選択して「OK」ボタンをクリックする事で、paraFoam が起動するので結果を確認できる。下図が確認した結果




になる。



### 6-2-7. 並列計算

並列計算を行う前に現在の計算結果を削除しておく。削除は、 ボタンをクリックする事で、log や計算結果等の不要なファイルを削除し、case を初期化する事ができる。

計算結果を削除した後、TreeFoam 上から  ボタンをクリックする。クリックした時点で、TreeFoam は、

case 内に decomposeParDict があるかどうかを確認し、存在しない場合は、デフォルトの decomposeParDict をコピーして作成する。この為、decomposeParDict の存在を意識せず、並列計算ができる。

並列計算用の画面上で並列数 (nCPU) を確認する。並列数を修正するようであれば、テキストボックス中の数字を直接変更し、「nCPU,method 設定」ボタンをクリックする。(「nCPU,method 設定」ボタンが非アクティブになっているが、nCPU 数を変更すると、アクティブに変わり、設定できる。)



また、同時に mesh 分割方法 (method) も確認できるので、確認する。今回の場合、simple の (2 2 1) で設定されていることが確認できる。分割方法や分割数を変更するのであれば、ここで直接修正する。修正後は、「nCPU,method 設定」ボタンをクリックして、decomposeParDict に反映させる。また、「Dict 確認・編集」ボタンで、decomposeParDict を editor で開く事ができるので、その設定内容を確認する事ができる。以下はその設定内容になる。

```

16 // *****
17
18 numberOfSubdomains 4;
19
20 method          simple;
21
22 simpleCoeffs
23 {
24     n              ( 2 2 1 );
25     delta          0.001;
26 }
--

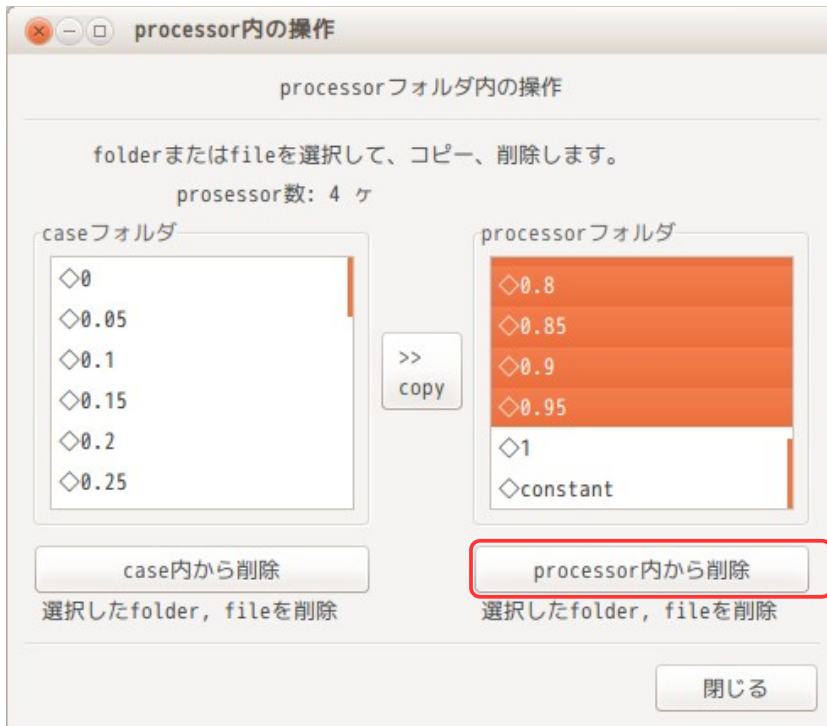
```

この後は、メッシュを各 processor 毎に分割する。この為に、下図の「mesh 分割」ボタンをクリックする。この時の処理状況が TreeFoam 下部のテキストボックス中に log が表示されるので、処理状況が確認できる。


分割後は、「並列計算開始」ボタンをクリックして並列計算を開始させる。計算終了後は、「結果の再構築」ボタンにより、各 processor 毎に分割されている計算結果を集めて case フォルダ直下に保存する。



計算結果を再構築した後は、各 processor 毎の計算結果は不要になる。これを削除する為に「各領域の file 操作」ボタンをクリックすると、下図の「processor 内の file 操作」画面が現れるので、この画面上で削除するフォルダを選択し、一括して削除できる。各 processor 内の time フォルダは、基本的に最初と最後のみ残しておけば、問題ない。これを残しておくと、結果ファイルの容量が 2 倍になってしまう。



最初と最後の time フォルダ  
以外を選択し、  
「processor 内から削除」  
ボタンをクリックして削除  
する。

結果を再構築した後は、TreeFoam 画面上の  ボタンをクリックして paraFoam を起動し、結果を確認する事ができる。



## 7. メッシュ作成の例

snappyHexMesh と cfMesh を使ってメッシュを作成してみる。  
 snappyHexMesh の場合は、cellZone や faceZone を作成することができる為、内部 patch や multiRegion モデルの作成が容易にできる。  
 cfMesh の場合は、現段階では cellZone や faceZone の作成ができないので、上記した事が難しくなるが、これらを作成しない通常のメッシュを作成するだけであれば、snappyHexMesh よりも容易にメッシュを作成することができる。(設定項目が少ない。)

上記した事を踏まえて、ここで、snappyHexMesh を使って通常のメッシュと cellZone や faceZone 付きのメッシュを作成し、さらに cfMesh を使って通常のメッシュを作成してみる。  
 最後に、salome で作成したメッシュを FOAM 形式にグループ名 (volume 名、face 名) 付きで変換できる様にしているので、このメッシュも作成してみる。

### 7-1. snappyHexMesh による通常メッシュの作成

snappyHexMesh は、stl ファイルさえ準備できれば、ほぼ自由な形状のメッシュを作成することができるので、重宝するが、その設定項目が多数あり、使いづらい面がある。この為、TreeFoam 上で snappyHexMesh を使ってメッシュを切る時、使いやすさに重点をおいてメッシュが作れる様にしている。具体的には、以下に示す方針でメッシュが作成できる様にした。

- 1) 座標の入力はしない。(stl ファイルから座標を拾う)
- 2) blockMeshDict、snappyHexMeshDict を意識しなくても、メッシュが作れる。
- 3) メッシュの修正が楽に行える。
- 4) メッシュの微調整は、直接 dict ファイルを修正。

blockMesh については、そのメッシュ作成方法が座標入力と分割数でメッシュサイズを決めているので、直感的に判りにくい。この為、座標は stl ファイルから拾い、分割数はメッシュサイズを入力すれば済む方法にしている。snappyHexMesh については、準備した stl ファイルの区分 (patch, wall, faceZone, cellZone 等) とメッシュサイズを明確にする事で、メッシュが切れる様にしている。

メッシュ作成用のデータは、データ入力用の dialog を用意しているので、ここで入力する。入力したデータから blockMeshDict、snappyHexMeshDict を作り出してメッシュを作成する。

前記した方法で、メッシュを作成してみる。作成するメッシュは、通常のメッシュを作成してみる。

#### 7-1-1. case の作成

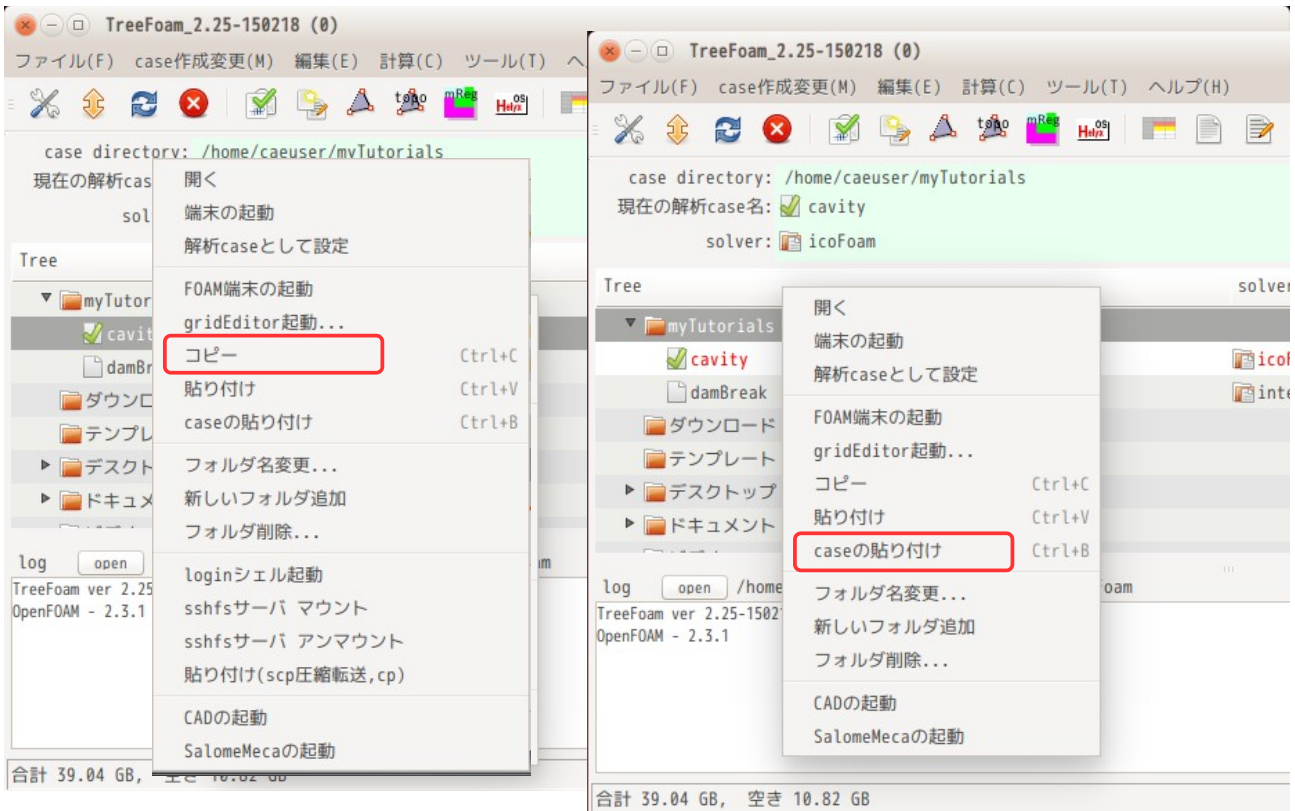
まず基本となる case を作成する。mesh を作成するだけの為、case は「0」、「constant」、「system」フォルダがあれば何でも構わないが、6-1 項で作成した「cavity」をコピーして使う。(今後、mesh 作成用の case は、cavity の case を使って作成する事を前提として進める。)

case のコピー方法は、「cavity」を選択して、右クリックでポップアップメニューを表示させ、「コピー」を選択する。この後、「myTutorials」フォルダを選択して、再びポップアップメニューを表示させ、「case の貼り付け」を選択して、case を貼り付ける。

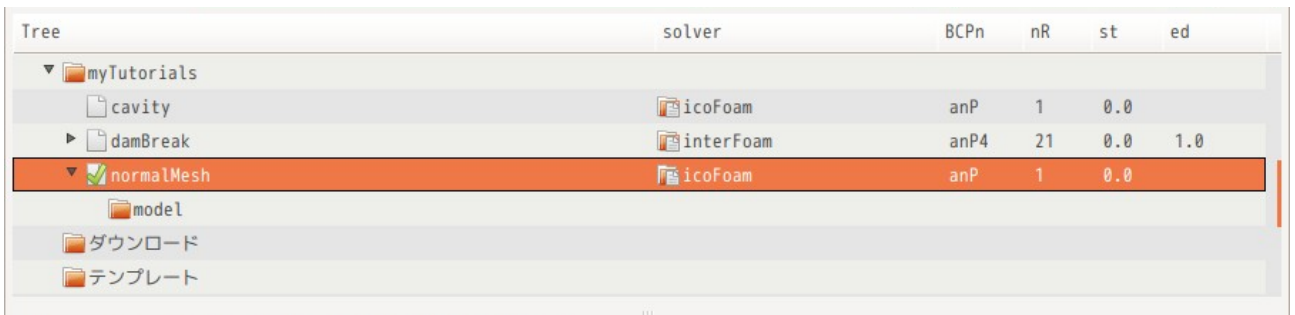
この後、ポップアップメニューから「folder 名変更...」を選択して、case 名を「normalMesh」に変更する。さらに、ポップアップメニューの「新しいフォルダ追加」を選択して、normalMesh フォルダ内に stl ファイル保存用のフォルダ「model」を作成しておく。

最終的にフォルダの構成は、以下になる。

myTutorials	
normalMesh	メッシュ作成用 case
0	
constant	
model	stl ファイル保存用フォルダ
system	

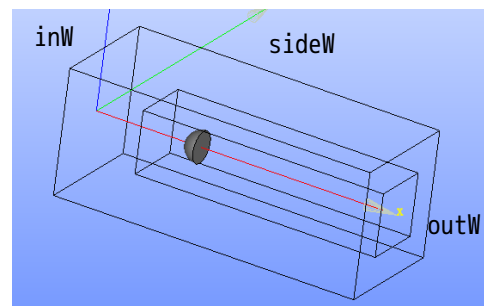
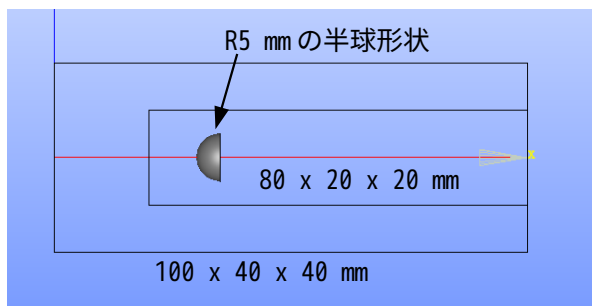


最終的な状態 (case 名が「normalMesh」に変更され、model フォルダが追加されている。)



## 7-1-2. モデル形状



以下のモデルのメッシュを作ってみる。

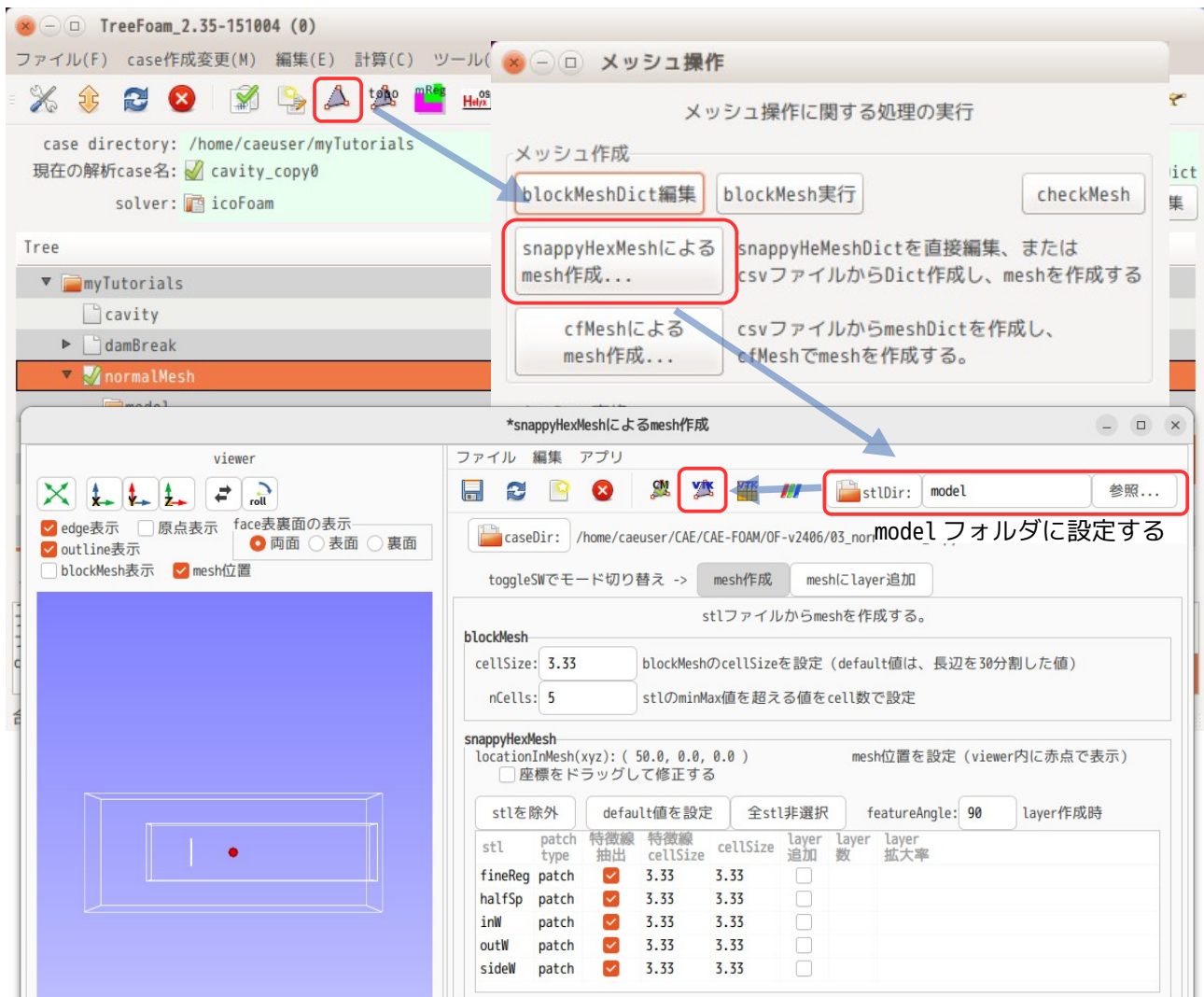


stl ファイルは、以下を作成する。(今回は salome を使って stl ファイルを作成している。)

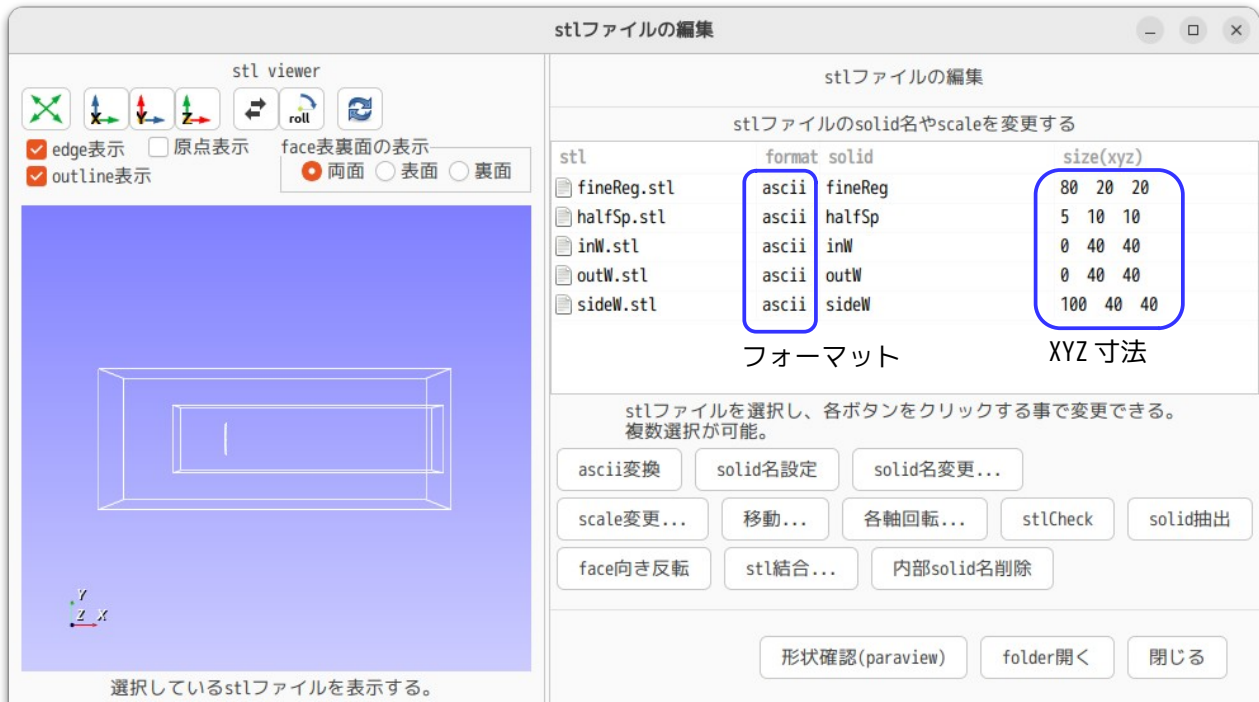
部位	stl ファイル	備考
100x40x40	inW.stl, sideW.stl, outW.stl	patchを作成
80x20x20	fineReg.stl	cell サイズを細かくする為の領域を定義
半球	halfSp.stl	この部分をくり抜く (patchを作成)

できあがった stl ファイルは、normalMesh/model フォルダ内に保存しておく。  
 これら保存した stl ファイルは、まず TreeFoam 上で stl ファイルのフォーマットや寸法を確認しておく。  
 (これらの stl ファイルは、\$TreeFoamPath/data/stlFiles/normalMesh 内に保存している。)

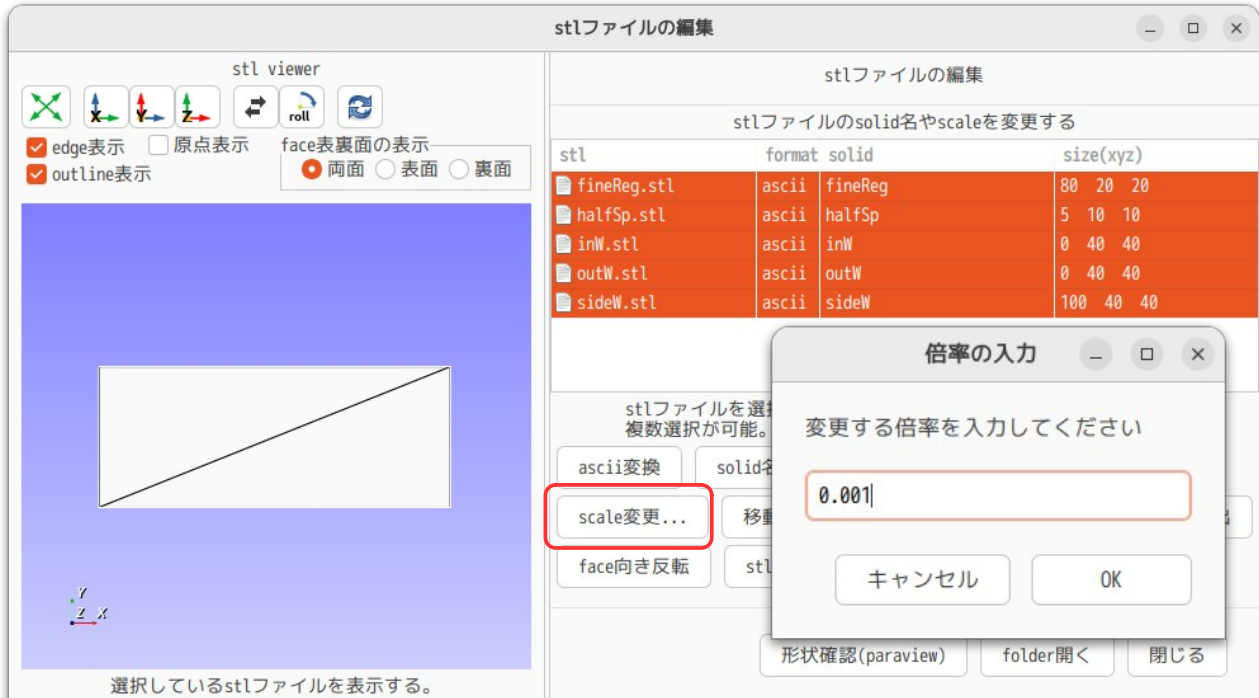
TreeFoam 上の  ボタンをクリックして「snappyHexMesh による mesh 作成...」ボタンをクリック、この後、stlDirを確認し、 ボタンをクリックして、「stl ファイルの編集」画面上でフォーマットと寸法を確認する。



フォーマットは、全て ascii だが、xyz の寸法が全て mm 単位の値になっている事が判る。(下図参照)  
 (もし、フォーマットが binary の場合は「ascii」変換ボタンをクリックして ascii に変換できる)





stl が mm 単位で作成されているので、全ての stl ファイルを 1/1000 に縮小する。  
 倍率変更は、画面上で変更するファイルを選択し、「scale 変更...」ボタンをクリック、倍率「0.001」を入力して、m 単位に変更する。

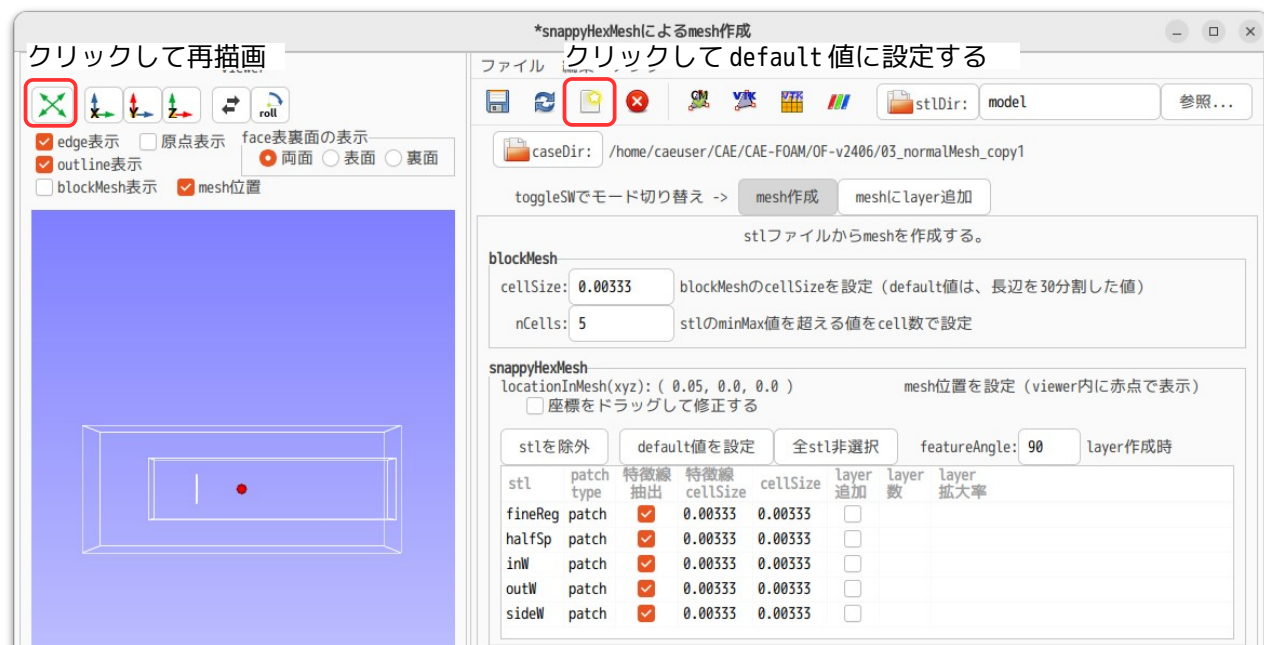


尚、この「stl ファイルの編集」画面上では、scale の変更の他に、solid 名の変更や、複数の stl ファイルを結合したり、face の向きを反転させる事ができる。操作方法は、いずれも対象ファイルを選択してボタンをクリックする事で実現できる。また、ポップアップメニューでファイルコピーや削除する事もできる。(詳細は、9-1-1 項を参照)



## 7-1-3. メッシュ作成用データ入力

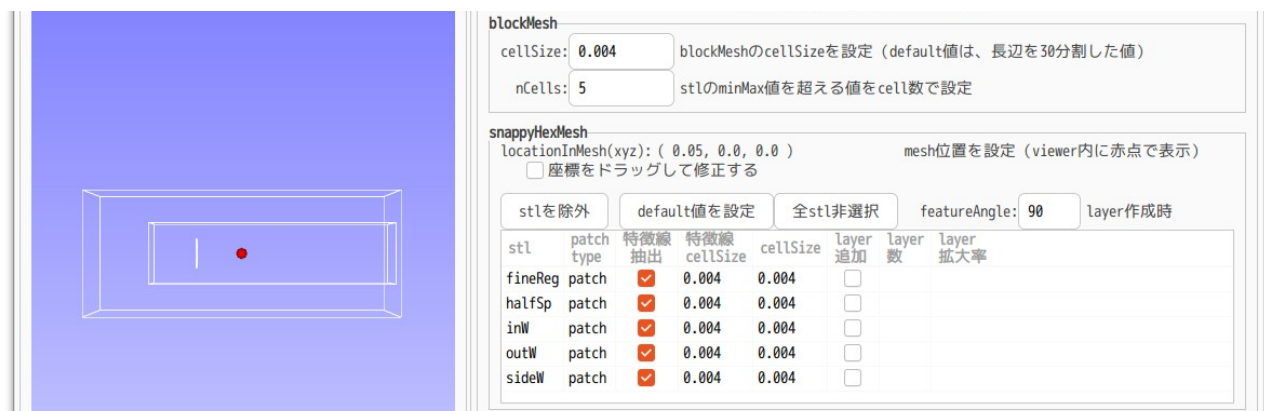
「snappyHexMesh による mesh 作成」画面上で、メッシュ作成用のデータを入力する。  
 まず、画面上の  と  ボタンをクリックして、表示されている値を default 値に戻し、view を再描画する。  
 (前項で stl の倍率を変更した場合は、倍率変更前の値が表示されている。)



## &lt;blockMesh の設定&gt;

blockMesh 関係の値を設定する。 cellSize と nCells を設定する。  
 cellSize : blockMesh の cellSize であり、この値で block を分割する。  
 (default 値は、モデルの長辺を 30 分割した値)  
 nCells : blockMesh の block をモデルサイズに nCells 分を加えた値で作成する。  
 (default 値は、nCells=5。5cell 分 block を大きく作成する。)

cellSize は、0.00333 なので、きりの良い 0.004 に設定する。nCells は、そのまま。これらの値を確定 (enter を入力) すると、cellSize=0.004 に対応した値が再設定される。(下図参照)



## &lt;snappyHexMesh の設定&gt;

snappyHexMesh の場合は、まず locationInMesh の座標を確認する。この座標は、viewer 内の赤点で表示されている。この座標は、メッシュ位置を示す座標で、この点を起点としてメッシュを作成すると考えると理解しやすい。

default では、モデルサイズ (XYZ) の中心座標が設定されている。モデル中心に空洞がある様なモデルでは、モデル中心にメッシュが無いため、座標を変更する必要がある。今回の場合は、モデル中心に空洞が無い為、default 値のままの値を使う。

この座標を変更する場合は、「座標をドラッグして修正する」をチェックして、赤点をドラッグして位置を変更する。

次に、各 stl ファイルを使ってメッシュを作成するが、この各 stl ファイルの扱い方法を定義する。  
定義内容は、以下を定義する

項目	内容	default 値
patchType	patch, wall, cellZone など	patch
特徴線の抽出	特徴線を抽出する、しないを設定	抽出する
特徴線 cellSize	特徴線抽出する場合、特徴線の cellSize を設定	blockMesh の cellSize
cellSize	stl ファイルの領域の cellSize を設定	↑
layer 追加	layer を追加する、しないを設定	layer 追加しない
layer 数	layer 追加する場合、追加する layer 数を設定	3
layer 拡大率	layer 追加する場合、layer の拡大率を設定	1.2

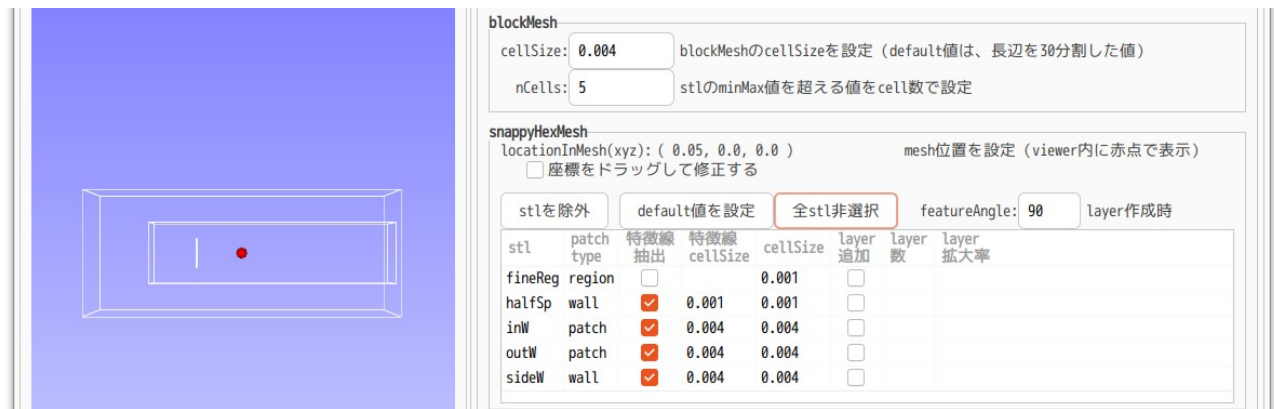
データの編集方法は、該当項目をクリックする事によって、dropDown メニュー表示、cell 編集モード移行するので、その項目が編集できる。

patchType の内容は、以下の内容が設定できる。

patchType	内容
patch	patch を定義
wall	wall を定義
empty	empty を定義
symmetry	symmetry を定義
symmetryPlane	symmetryPlane を定義
cellZone	cellZone を定義
cellZoneOther	cellZone 定義している領域以外を定義
faceZone	faceZone を定義
region	volume 領域を定義 (領域定義のみで cellZone は作らない)
face	face 領域を定義 (領域定義のみで faceZone は作らない)

上記を cellZone, cellZoneOther, faceZone, その他に分類して、メッシュを作成する順番は、  
その他、cellZone, cellZoneOther, faceZone の順番でメッシュを作成する。この順番にしないと cellZone  
や faceZone が作成されない事があるため。

最終的に以下のように設定した。



fineReg	cellSize を細かく設定する為に設けているので、 patchType:region, 特徴線抽出せず, cellSize:0.001 に設定
halfSp	cellSize が細かい wall として設定 patchType:wall, 特徴線抽出, cellSize:0.001
inW, outW	流入面、流出面として設定している。 PatchType:patch, 特徴線抽出, cellSize:0.004
sideW	外周部 patchType:wall, 特徴線抽出, cellSize:0.004

尚、ここでは、layer を追加していない。layer を追加する場合は、7-1-5 項を参照。

#### 7-1-4. メッシュ作成



メッシュ作成用のデータ入力完了後、メッシュを作成する。  
メッシュ作成は、「Dict 作成」ボタンをクリックして、特徴線抽出用の surfaceFeatureExtractDict、blockMeshDict、snappyHexMeshDict を作成する。

メッシュ作成は、「mesh 作成」ボタンをクリックする。このクリックにより、作成した Dict を順番に実行して、メッシュを作成する。

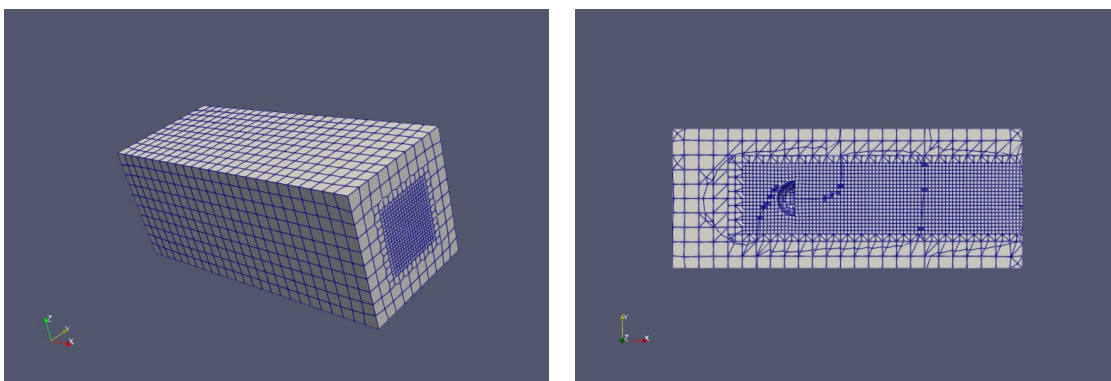
また、メッシュを微調整するようであれば、Dict 作成後、snappyHexMeshDict を修正後、再実行する。



尚、並列処理する場合は、「並列処理」をチェックし、並列数「nProcs」、分割方法「分割数(xyz)」を入力して、「mesh 作成」ボタンをクリックする事で並列処理できる。  
並列処理は、以下を実行しており、3,5,6 項が並列特有の処理になる。

- |                           |                              |
|---------------------------|------------------------------|
| 1. surfaceFeatureExtract  | 特徴線の抽出                       |
| 2. blockMesh              | blockMesh 作成                 |
| 3. decomposePar           | blockMesh を CPU 毎に分割         |
| 4. mpirun (snappyHexMesh) | snappyHexMesh を並列で起動し、メッシュ作成 |
| 5. reconstructPaMesh      | 各 CPU 毎のメッシュを集め再構築           |
| 6. reconstruct cellLevel  | cellLevel を再構築               |
| 7. deleteFolders          | 余分な folder を削除               |

できあがったメッシュは、以下になる。

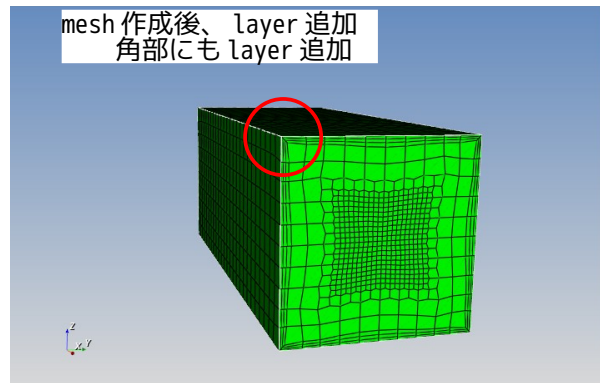
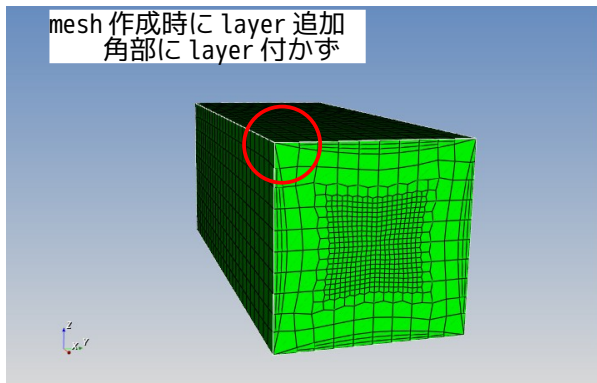


### 7-1-5. レイヤ作成

作成したメッシュは、layer 無しで作成している。  
layer を追加する場合は、以下の 2 通りの方法がある。

- 1) メッシュ作成時に layer を追加
- 2) メッシュ作成後、layer を追加

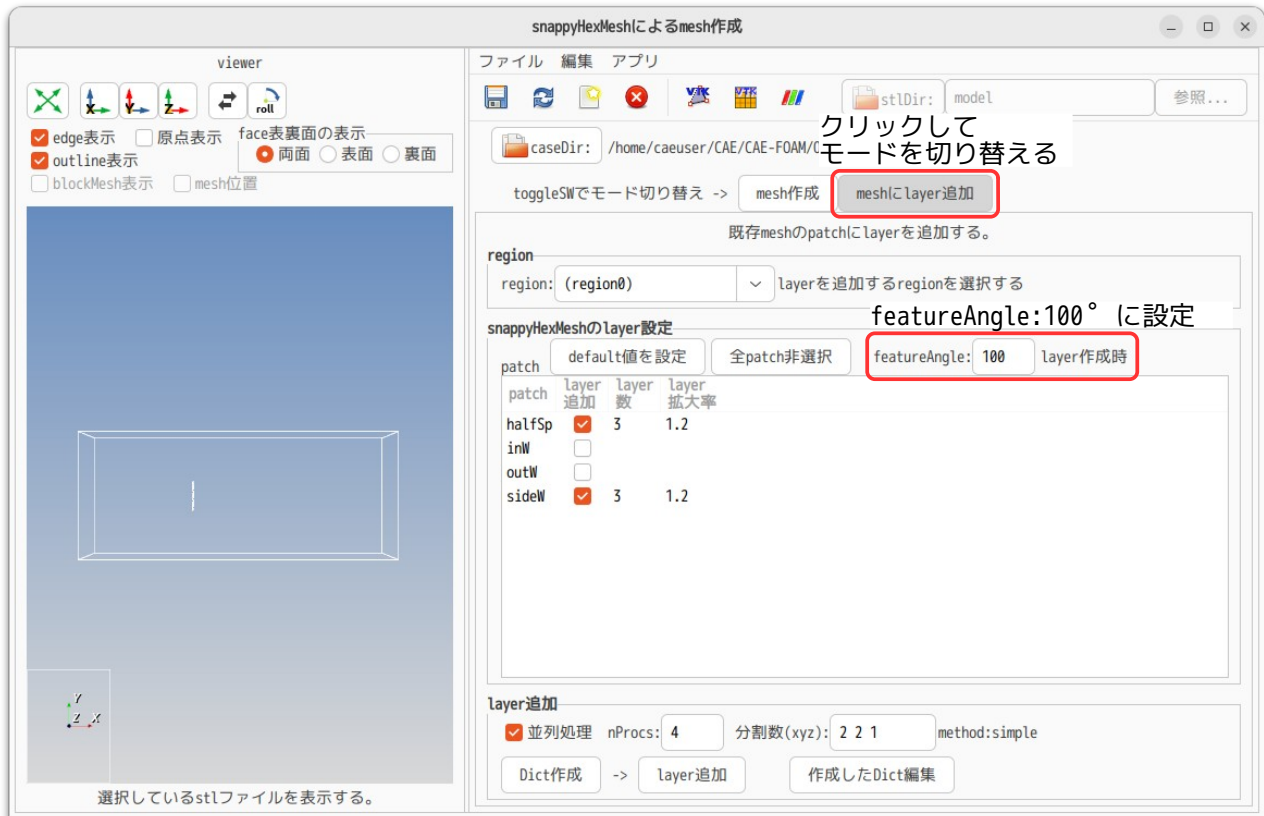
メッシュ作成時に layer を追加すると、以下の様に角部に layer が追加されない。(OpenFOAM の version によっては、角部に layer が追加される事がある。)  
これに対し、メッシュ作成後、指定した patch に layer を追加した場合は、角部にも layer をつけることができる。(下図参照)



この原因は、layer 追加時の「featureAngle」を  
メッシュ作成時：  $80^\circ$   
メッシュ作成後：  $100^\circ$   
に変更しているため。

メッシュ作成時に、featureAngle を  $100^\circ$  に設定してメッシュを作成した場合、モデル外周の edge 部の cell が欠ける事があるため、メッシュ作成時は、featureAngle を  $80^\circ$  に設定している。  
メッシュ作成後は、メッシュの patch に layer を追加するのみの為、featureAngle を  $100^\circ$  に設定している。

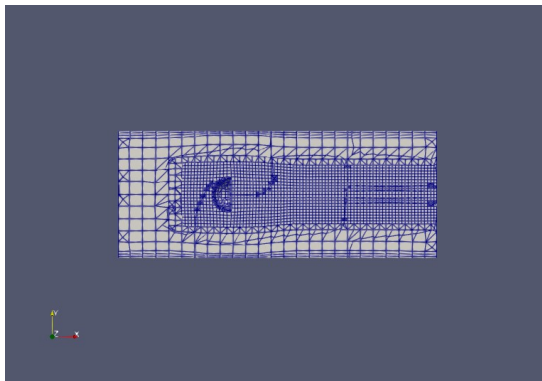
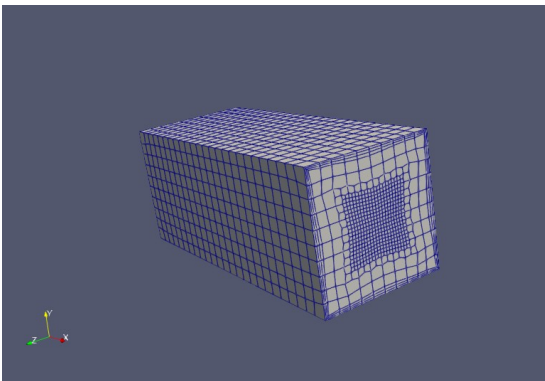
メッシュ作成後に layer を追加するには、前項の方法でメッシュ作成後、toggleSW を「mesh に layer 追加」ボタンをクリックして、モードを切り替える。  
この後、layer を追加したい patch に以下の様にチェックを追加する。以下の例では、halfSp と sideW に layer を追加している。尚、layer 数、layer 拡大率は、必要に応じて修正する。修正方法は、該当部をクリックして cell 編集モードに切り替えて修正する。

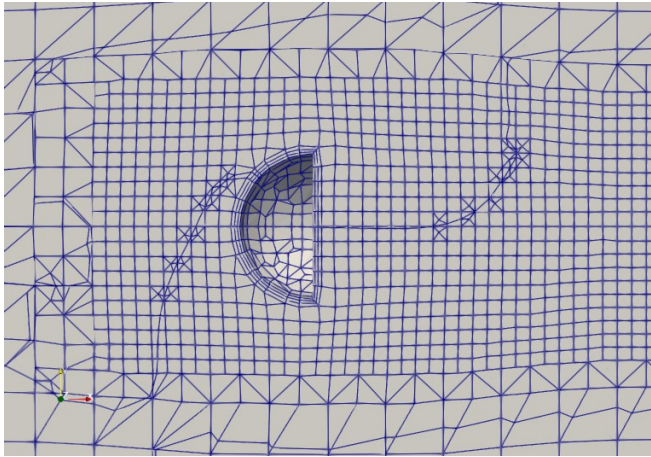


上記設定で layer を追加する為に、「Dict 作成」ボタンをクリック後、「layer 追加」ボタンをクリックする。これにより、layer が追加される。

layer が追加されたメッシュは、overwrite していないので、次の時間ステップフォルダ（今回は、0.005 フォルダ）に保存される。

できあがったメッシュは、以下になる。





layer がうまく追加されている。

メッシュ作成後に layer を追加する方法は、snappyHexMesh で作成したメッシュ以外（ideasUnvToFoam 等で変換したメッシュ）でも layer を追加することができる。

また、multiRegion の case でも、layer を作成したい region が選択できるので、流体-固体境界面にも容易に layer を追加する事ができる。

## 7-2. snappyHexMesh による faceZone や cellZone を含むメッシュ作成の例

モデル内に baffle を追加したり、tutorials の damBreak の様に特定領域に値をセットしたい場合には、予め faceZone や cellZone を作っておくと、baffle の作成や setFields がしやすくなる。この様な faceZone と cellZone を含むメッシュを snappyHexMesh を使って作成し、そのモデルで計算してみる。

### 7-2-1. メッシュ作成用 case の作成

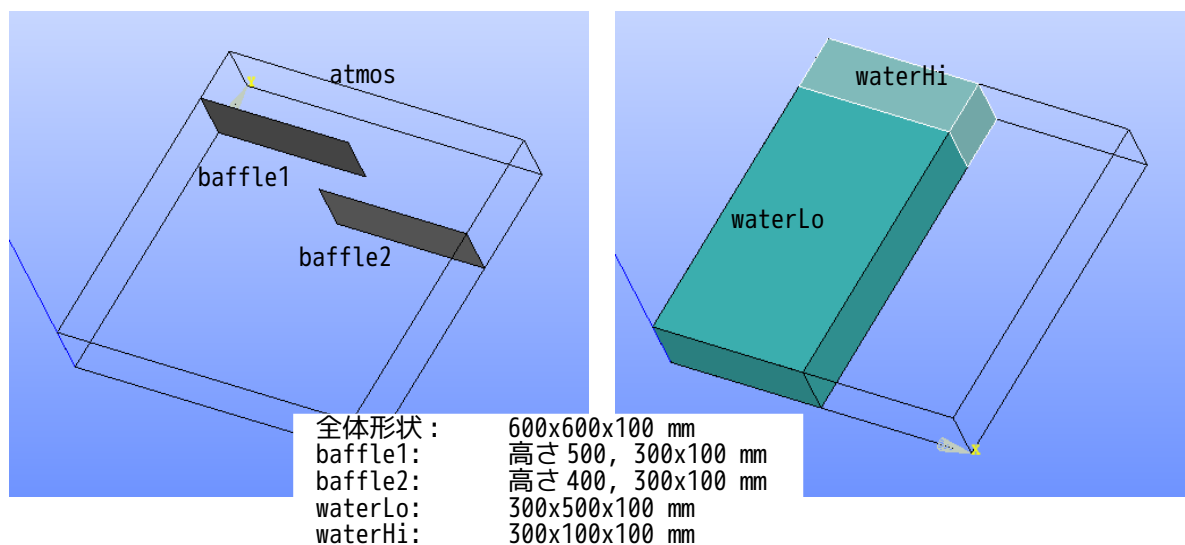
前項（7-1-1 項）と同様な方法で、cavity をコピーして case を作成する。case 名は「faceCellZoneMesh」とした。最終的に以下の様なフォルダ構成とする。

```
myTutorials
├── faceCellZoneMesh  メッシュ作成用 case
│   ├── 0
│   ├── constant
│   ├── model          stl ファイル保存用フォルダ
│   └── system
```

Tree	solver	BCPn	nR	st	ed
▼ myTutorials					
├── cavity	icoFoam	anP	6	0.0	0.5
├── damBreak	interFoam	anP4	21	0.0	1.0
▼ faceCellZoneMesh	icoFoam	anP	1	0.0	
├── model					
├── normalMesh	icoFoam	anP	2	0.0	0.005
└── package					

### 7-2-2. モデル形状

下図の様なモデルを考えてみる。




stl ファイルは、以下を準備する。

(今回の stl ファイルは、salome で作成しており、\$TreeFoamPath/data/stlFiles/faceCellZoneMesh 内に h 保存しているので、ここから入手できる。)

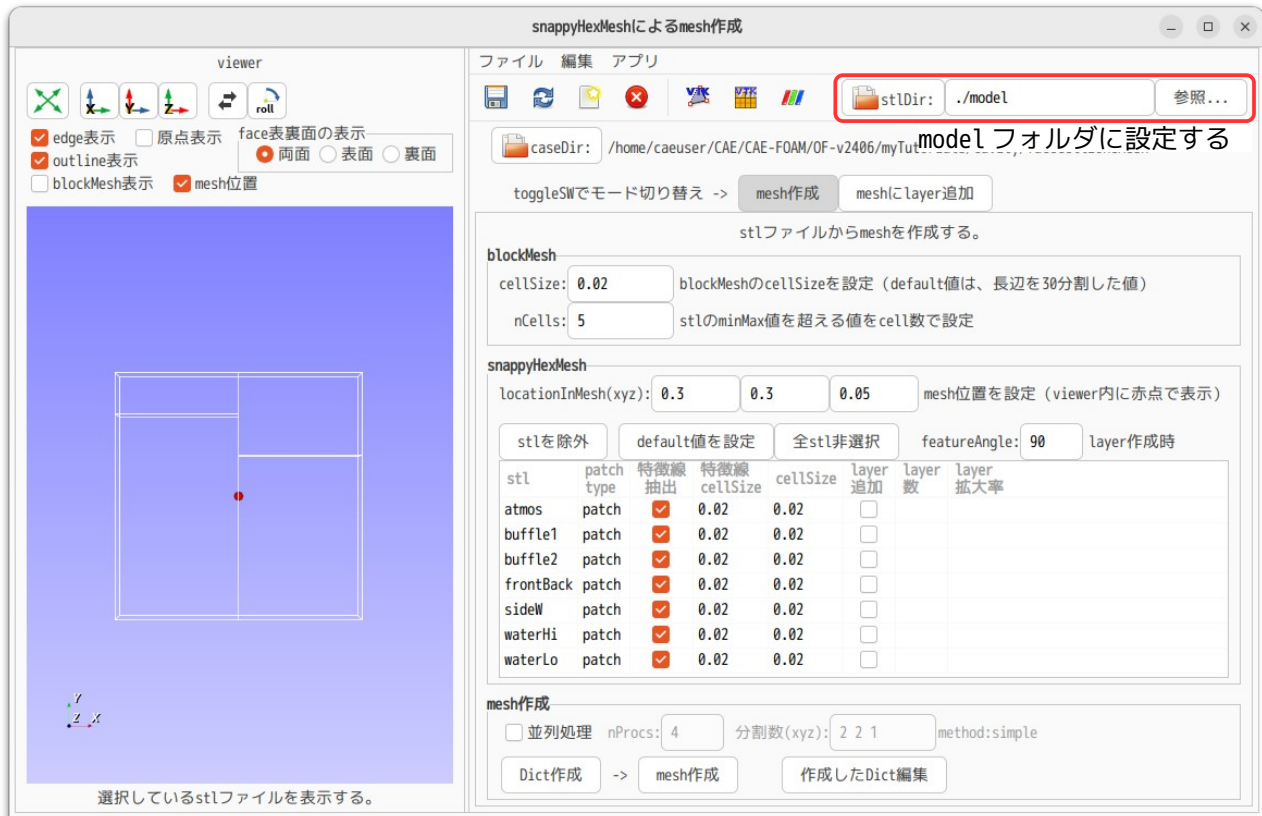
stl ファイル	内容
sideW.stl	wall:側面と底面の3面
atmos.stl	patch:上面
frontBack.stl	wall:表と裏面
baffle1.stl	faceZone
baffle2.stl	faceZone
waterLo.stl	cellZone
waterHi.stl	cellZone

これら全ての stl ファイルを faceCellZoneMesh/model 内に保存しておく。

### 7-2-3. メッシュ作成用データ入力

TreeFoam 上の  ボタンをクリックして「snappyHexMesh による mesh 作成...」ボタンをクリックして、「snappyHexMesh による mesh 作成」画面を表示させる。  
この時、stlDir を確認し、model フォルダ以外ならば、「参照...」ボタンで、faceCellZoneMesh/model に設定する。正しく stlDir が設定されていれば、以下の画面が表示される。





今回の場合、cellSize, patchType のみ修正した。(それ以外は、default のまま。)



cellSizeと各 stl の patchType を上記の様に設定している。

#### 7-2-4. メッシュ作成

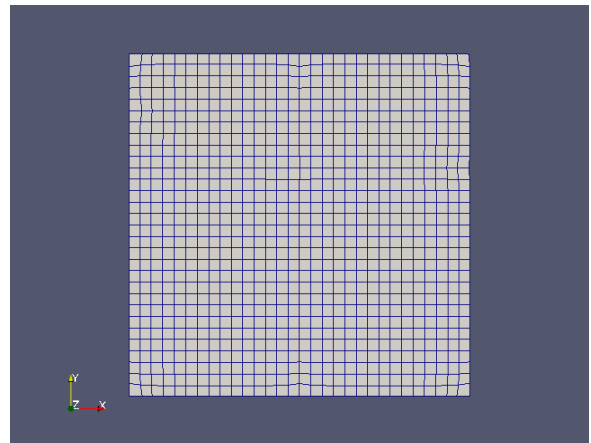
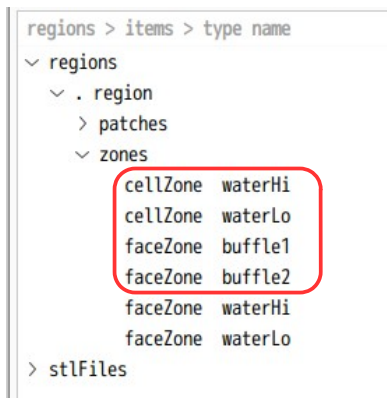
メッシュ作成の為の設定ができたので、「Dict 作成」ボタンをクリックして、各 Dict を作成し、「mesh 作成」ボタンをクリックしてメッシュを作成する。

以下ができあがったメッシュになる。cellZone や faceZone が取得できている。

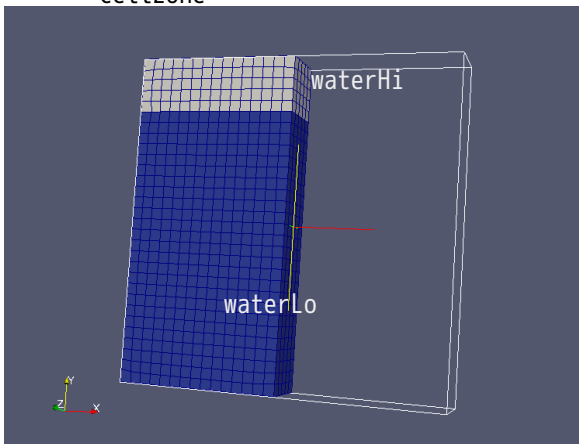
Zones

全体

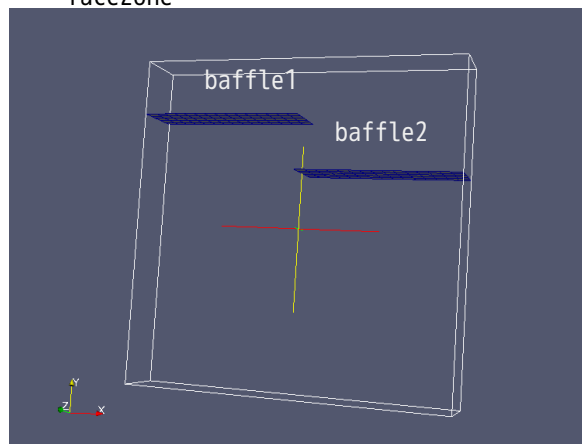




cellZone



faceZone



### 7-2-5. 解析用 case の作成

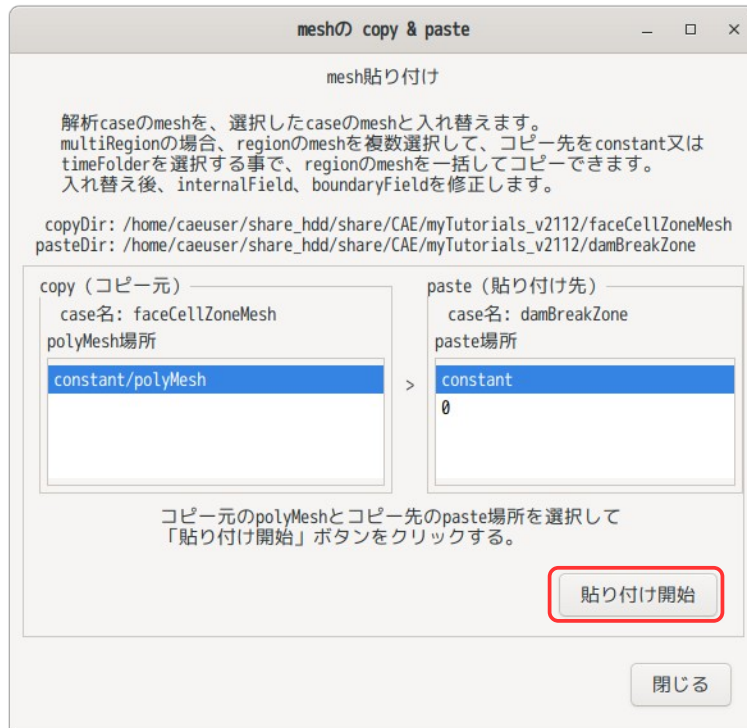
今回のメッシュは、tutorials の damBreak を想定したものである為、case の内容 (field や properties 等) を damBreak の内容に揃える必要がある。この為に、6-2 項で実行した damBreak の case をコピーし、新しく「damBreakZone」の名称に変え、この case 内のメッシュを今回作成したメッシュに入れ替える事にする。

下図は、damBreak の case をコピーして case 名を「damBreakZone」に変更し、解析 case (📁付きフォルダ) として設定した状態。

Tree	solver	BCPn	nR	st	ed
▼ myTutorials					
cavity	icoFoam	anP	6	0.0	0.5
▶ damBreak	interFoam	anP4	21	0.0	1.0
📁 damBreakZone	interFoam	anP	1	0.0	
▶ faceCellZoneMesh	icoFoam	anP	1	0.0	
▶ normalMesh	icoFoam	anP	2	0.0	0.005
▶ package					

今回作成した「faceCellZoneMesh」内のメッシュをコピーして、「damBreakZone」へ mesh 貼り付けするが、この方法は、以下の方法による。

case 「faceCellZoneMesh」を選択し、ポップアップメニューを表示し、「コピー」を選択する。この後、case 「damBreakZone」を選択してポップアップメニューの「mesh 貼り付け」を選択して、mesh 貼り付けの画面を表示させる。この画面上で、メッシュをコピー、貼り付けを行う。



コピーする mesh と貼り付け先を確認して「貼り付け開始」ボタンをクリックして、mesh を貼り付ける。

コピー mesh: faceCellZoneMesh/constant/polyMesh


貼り付け先: damBreakZone/constant

mesh 貼り付け後は、「閉じる」ボタンで、画面を閉じておく。

以上の操作で、新たに作成した mesh が damBreakZone 側に貼り付けられた事になる。

これにより、damBreakZone 側の、internalField と baoundaryField の内容は、全てクリアされる。

## 7-2-6. setFields で値をセット

「alpha.water」フィールドに setFields コマンドで値をセットする為、TreeFoam 上の  ボタンをクリックして、「field へのデータセット」画面を表示させる。

setFields コマンドは、OF-13 から書式と作動が変わっている。

OF-13 cellZone の領域に値がセットされる

OF-13 以外 cellSet の領域に値がセットされる

OF-13 は、cellZone 領域に値がセットできる為、次の topoSetEditor の操作は必要ない。

OF-13 以外の場合、topoSetEditor を使って、cellZone の waterLo と waterHi から cellSet の waterLo と waterHi を作り出す必要がある。

これらを作り出す為に、「field へのデータセット」画面上の「cellSet 作成」ボタンをクリックする。

「mesh 抽出」画面 (topoSetEditor) が現れるので、この画面上で

<Action>	new	新しく object を作り出すコマンド
<Source>	cellZone	cellZone 内の
	waterHi	waterHi と
	waterLo	waterLo を選択
<Result>	sets	複数の source から複数の set (cellSet) を作り出す

を選択して、「code 出力」ボタンをクリックする。この操作により、画面下部のテキストボックス中に、この処理を行うための topoSet のコマンド群が作成される。

この後、「クリア・追加・実行」ボタンをクリックする事で、topoSetDict の内容をクリア、作成した topoSet コマンドを Dict に追加、topoSet を実行し、最終的に選択した cellZone から cellSet を作り出す。

最後に「閉じる」ボタンをクリックして、「mesh 抽出」画面を閉じ、「field にデータセット」画面に戻る。

The image shows two screenshots of the TreeFoam software interface. The top screenshot shows the main TreeFoam window with the 'Fieldへのデータセット' (Field Data Set) dialog open. The bottom screenshot shows the 'topoSet Editor' window.

**Fieldへのデータセット (Field Data Set) Dialog:**

- timeFolder内の各Fieldへのデータセット (クリア)
- 編集する folder: time (startTime:0), region (region0)
- setFieldsによるデータセット: cellSet作成 (highlighted), field内容確認
- fields list: U, alpha.water (highlighted), alpha.water.orig, epsilon, k, nuTilda, nut
- setFieldsDict作成... cellSet, fieldを選択後、クリックして、setFieldsDict用のデータを作成する。

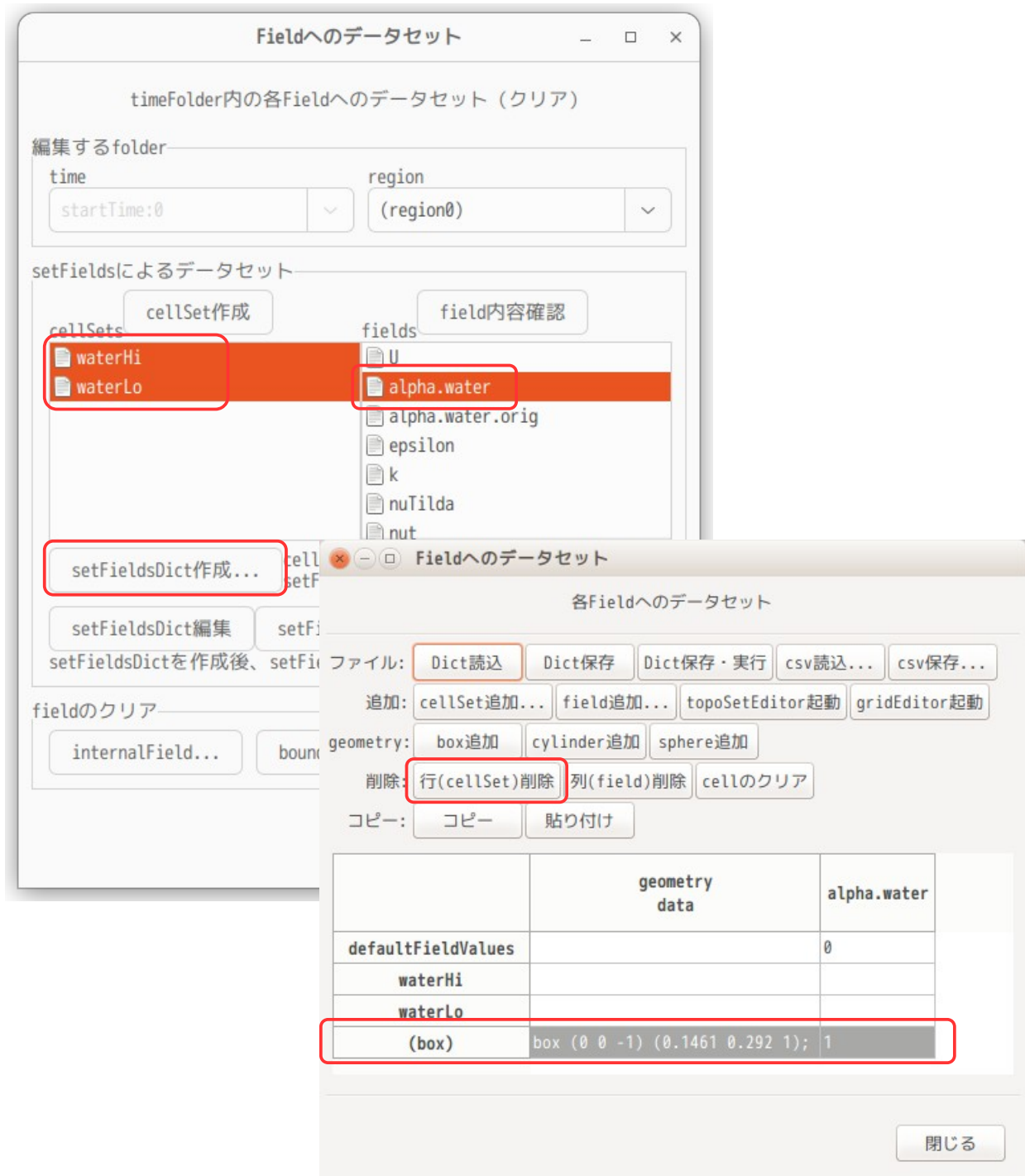
**topoSet Editor (topoSetDictを作成し、meshを抽出)**

- time: startTime :0, region: (region0) (-regionを設定して、topoSetを実行)
- <Action> コマンド: new (highlighted), add, delete, subset, no source, clear, invert, remove, combined, renameSetZone, newAddsSet, newCellToFace, newZonesToSets
- <Source> 入力 mesh:constant/. type: cellSet (highlighted), faceSet, pointSet, sets, zones, cellZone (highlighted), faceZone, pointZone, 幾何図形: box, cylinder, sphere, その他: surface, rotatedBox, targetVolume, searchableSurface, region, boundary, field, patch, label, shape, normal, nearest
- name: waterHi, waterLo (highlighted)
- <Result> 出力 mesh:constant/. type: sets (highlighted), cellSet, faceSet, pointSet, zones, cellZoneSet, faceZoneSet, pointZoneSet, zones
- code出力 (highlighted)
- クリア, 追加, 実行 (highlighted)
- コード確認, 編集, topoSetDictクリア, topoSetDict追加, topoSet実行
- コード確認, 編集:
 

```
// new To cellSet
{
  name    waterHi;
  type    cellSet;
  action  new;
  // Cells in cell zone
  source  zoneToCell;
  sourceInfo
  {
    name "waterHi" ;    // Name of cellZone, regular expressions allowed
  }
}
```
- <使用法>
  - ・ action (コマンド) を選択。
  - ・ source type, nameを選択。
  - ・ result type, nameを決定。
  - ・ 「topoSetDictに追加」
- topoSetDict編集, paFoam起動, 閉じる (highlighted)

「fieldへのデータセット」画面に戻ると、cellSets内に「waterHi」と「waterLo」が取得できているので、これらと、値をセットする field (alpha.water) を選択し、「setFieldsDict 作成...」ボタンをクリックする。

OF-13 の場合、topoSetEditor を起動しなくても、下図の様に waterLo と waterHi のセット領域が確認できる。



現れた画面上には、現在設定されている setFieldsDict の内容が表示されている。(damBreak で使用している setFieldsDict は、box を使ってデータをセットしているので、(box)行が表示されている。)

この画面中で、まず、不要な(box)行を削除する。削除方法は、削除したい行を選択して「行(cellSet)削除」ボタンをクリックする事で行が削除できる。削除後、alpha.water 列中の waterHi と waterLo に「1」を

入力する。最終的に以下になる。



データ入力後、「Dict 保存」ボタンをクリックすると、この内容で setFieldsDict が作成される。この後、「閉じる」ボタンで画面を閉じておく。

以下が、でき上がった setFieldsDict になる。(以下は、0F-13 以外の setFieldsDict になる。)

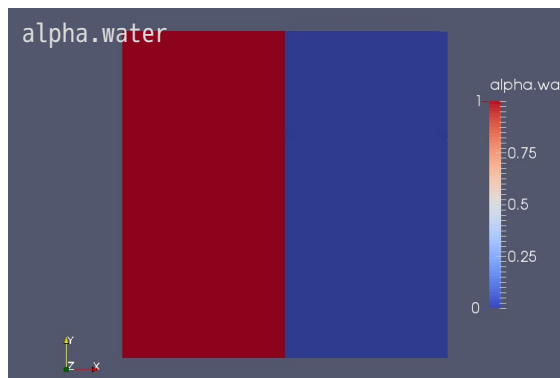
```
// ***** //
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);
regions
(
    cellToCell
    {
        set waterHi;
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
    cellToCell
    {
        set waterLo;
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
);
// ***** //
```

setFieldsDict ができ上がったので、画面上の「setFields 実行...」ボタンをクリックして、setFields コマンドを実行する。





データセット状況を paraFoam で確認すると、下図の様に、alpha.water フィールドに値がうまく設定できている。(waterHi、waterLo 領域に「1」がセットされている。)



#### 7-2-7. データセット状態の確認

今の状態は、baffle (内部パッチ) が設定されていない為、tutorials の damBreak と形状は異なるが同じ状態。ここで solver (interFoam) を実行してみる。

実行にあたっては、境界条件が設定されていないので、これを設定する。境界条件は、tutorials の「damBreak」と今回の「damBreakZone」の内容 2 種類を gridEditor で表示させ、お互いに copy & paste で貼り付ければ済む。下図参照。



gridEditor: damBreak/0/. (0:0) damBreak 側

ファイル(F) 編集(E) 表示(V)

	define patch at constant/. (boundary)	U	alpha.water	alpha.water.org	p_rgh
field type dimensions		volVectorField; [0 1 -1 0 0 0 0];	volScalarField; [0 0 0 0 0 0 0];	volScalarField; [0 0 0 0 0 0 0];	volScalarField; [1 -1 -2 0 0 0 0];
internal Field		uniform (0 0 0);	nonuniform List<scalar> 2769 ( 1 1...	uniform 0;	uniform 0;
leftWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; value uniform 0;
rightWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; value uniform 0;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; value uniform 0;
atmosphere	type patch;	type pressureInletOutletVelocity; value uniform (0 0 0);	type inletOutlet; inletValue uniform 0; value uniform 0;	type inletOutlet; inletValue uniform 0; value uniform 0;	type totalPressure; p0 uniform 0; U U; phi phi; rho rho; psi none; gamma 1; value uniform 0;
defaultFaces	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;

この部分（壁の境界条件）を選択し、コピーする。  
（ポップアップメニューから「cell コピー」を選択）

damBreakZone 側

	define patch at constant/. (boundary)	U	alpha.water	alpha.water.org	p_rgh
field type dimensions		volVectorField; [0 1 -1 0 0 0 0];	volScalarField; [0 0 0 0 0 0 0];	volScalarField; [0 0 0 0 0 0 0];	volScalarField; [1 -1 -2 0 0 0 0];
internal Field		uniform (0 0 0);	nonuniform List<scalar> 4500 ( 1 1...	uniform 0;	uniform 0;
atmosW	type patch; inGroups 1(patches);	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradient;
frontBackW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradient;
sideW	type wall; inGroups 1(wall);	type zeroGradient;			

この部分を選択し、貼り付ける。  
（ポップアップメニューから「cell 貼り付け」を選択）

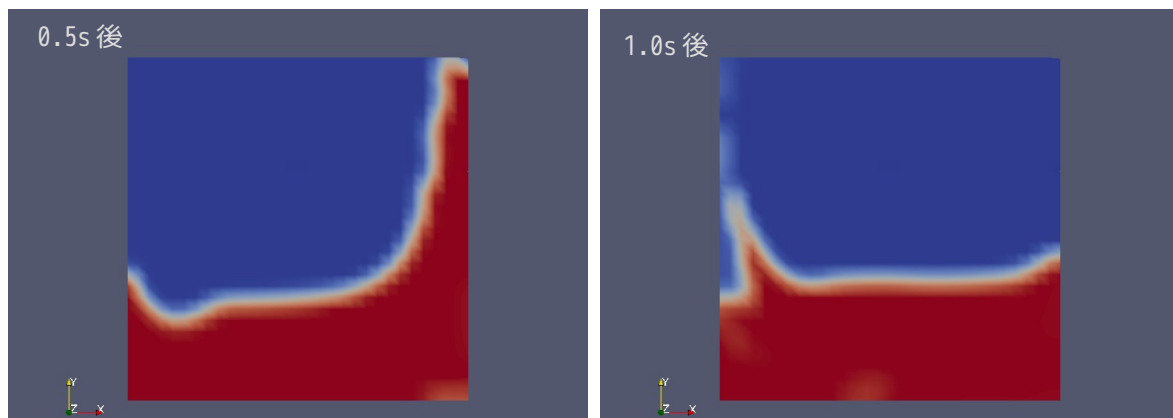
最終的に以下の状態（frontBackWは、「slip」に設定している。）になる。

gridEditor: damBreakZone/0/. (0:0)


ファイル(F) 編集(E) 表示(V)

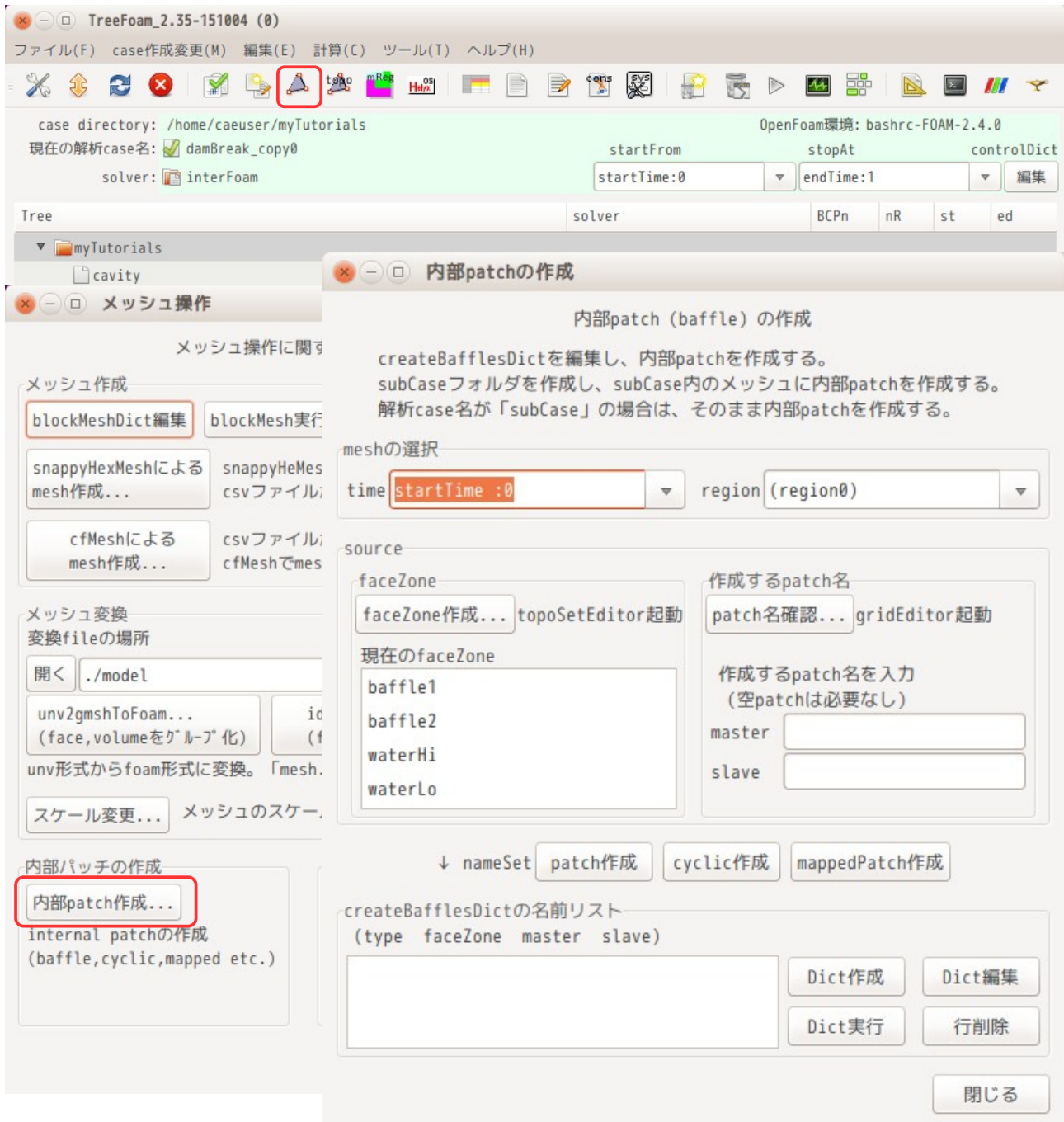
	define patch at constant/. (boundary)	U	alpha.water	alpha.water.org	p_rgh
field type dimensions		volVectorField; [0 1 -1 0 0 0];	volScalarField; [0 0 0 0 0 0];	volScalarField; [0 0 0 0 0 0];	volScalarField; [1 -1 -2 0 0 0];
internal Field		uniform (0 0 0);	nonuniform List<scalar> 4500 ( 1 1...	uniform 0;	uniform 0;
atmosW	type patch; inGroups 1(patch);	type pressureInletOutletVelocity; value uniform (0 0 0);	type inletOutlet; inletValue uniform 0; value uniform 0;	type inletOutlet; inletValue uniform 0; value uniform 0;	type totalPressure; p0 uniform 0; U U; phi phi; rho rho; psi none; gamma 1; value uniform 0;
frontBackW	type wall; inGroups 1(wall);	type slip;	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; value uniform 0;
sideW	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; value uniform 0;

この条件で計算させた結果が以下になる。baffle が無いため、tutorials の damBreak とほぼ同じ結果。



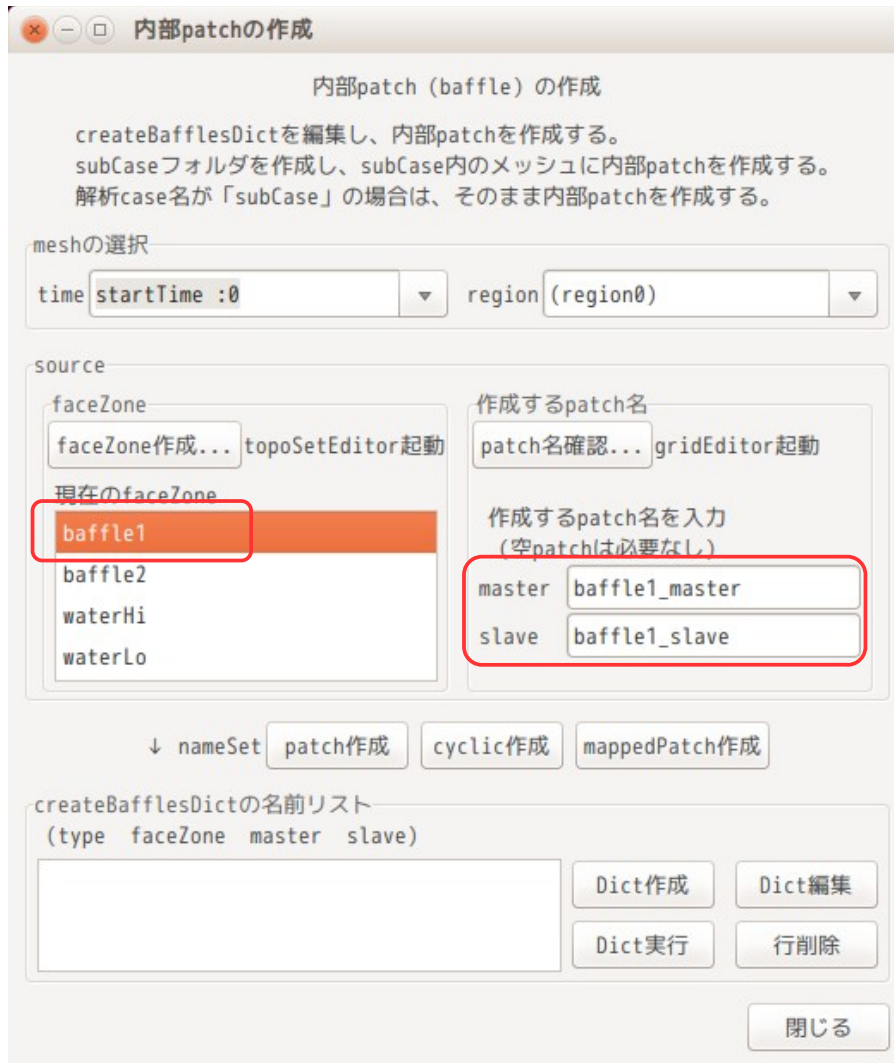
## 7-2-8. baffle (内部パッチ) 作成

このモデルには、faceZone を作成しているので、これを使って、baffle (内部パッチ) を作成してみる。この為には、TreeFoam 上の  ボタンをクリックして、「メッシュ操作」画面を表示させ、「内部 patch 作成...」ボタンをクリックして、「内部 patch の作成」画面を表示させる。この画面上で、内部 patch を作成する事になる。



内部パッチ (baffle) は、上図「内部 patch の作成」画面上から、作成したい faceZone を指定して baffle を作り出すことになる。今回は、faceZone 「baffle1」「baffle2」を使って、baffle 「baffle1」「baffle2」を作成する。

その作成方法は、下図のように、faceZone 「baffle1」を選択する。この選択により、patch 名の master, slave に、各々「baffle1\_master」「baffle1\_slave」が入力される。



今回の場合、baffleを作成するので、以下の様に、master、slaveとも、同じ名前「baffle1」に変更する。変更後、「patch作成」ボタンをクリックして、名前リストに追加する。

名前リストは、下表の意味を持っている。

区分	faceZone 名	master 名	slave 名
patch	baffle1	baffle1	baffle1



同様に、baffle2の方も操作し、以下の様な名前リストを作成する。  
この後、「Dict 作成」ボタンをクリックして createBafflesDict を作成し、「Dict 実行」ボタンをクリックして createBaffle コマンドを実行し、baffle1 と baffle2 の内部パッチを作成する。



尚、内部パッチが作成される case は、現在の解析 case 内に新しく「subCase」フォルダが作成され、この case 内に内部パッチを追加したメッシュが作られることになる。この為、以後の操作は、解析 case (マーク付き) を「subCase」に変更して、操作する。

また、解析 case 名が「subCase」で内部パッチを作成する場合は、subCase フォルダは作成されず、その case 内のメッシュに内部パッチが追加される。

尚、ここで作成された createBafflesDict ファイルは、以下の内容で作成されている。「Dict 実行」ボタン



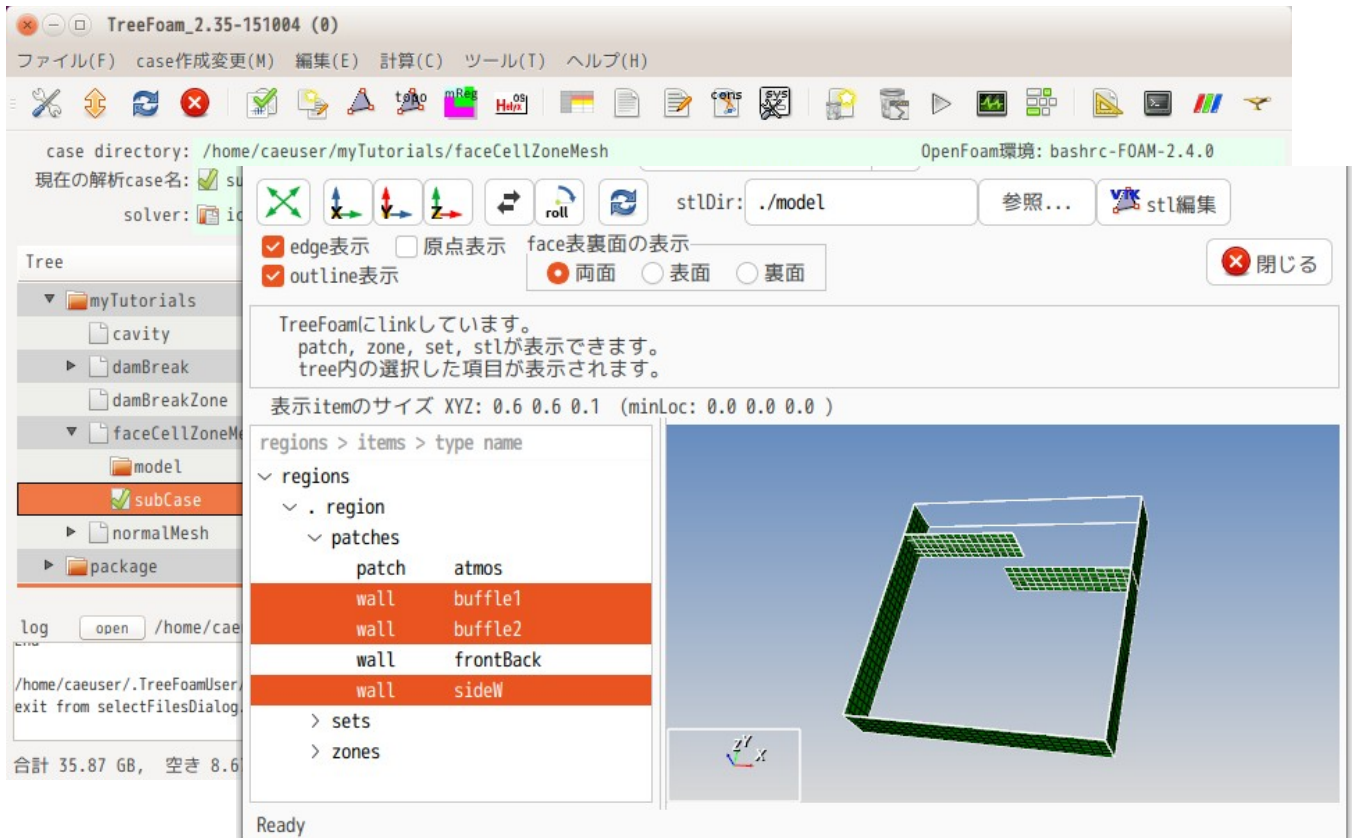
をクリックする前に、この内容を修正して「Dict 実行」ボタンをクリックすると、修正した createBafflesDict の内容で、baffle を作成する事ができる。

```
// * * * * *
// Whether to convert internal faces only (so leave boundary faces intact).
// This is only relevant if your face selection type can pick up boundary
// faces.
internalFacesOnly true;

// Baffles to create.
baffles
{
    baffle1      //baffles is created
    {
        //- Use predefined faceZone to select faces and orientation.
        type      faceZone;
        zoneName   baffle1;
        patches
        {
            master
            {
                //- Master side patch
                name      baffle1;
                type      patch;
            }
            slave
            {
                //- Slave side patch
                name      baffle1;
                type      patch;
            }
        }
    }
    baffle2      //baffles is created
    {
        //- Use predefined faceZone to select faces and orientation.
        type      faceZone;
        zoneName   baffle2;
        patches
        {
            master
            {
                //- Master side patch
                name      baffle2;
                type      patch;
            }
            slave
            {
                //- Slave side patch
                name      baffle2;
                type      patch;
            }
        }
    }
}

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


でき上がったメッシュを paraFoam で確認すると、以下になり、内部パッチが追加されている事がわかる。



尚、今回の様な単純な baffle のみ追加するのであれば、7-2-4 項で作成した csv ファイル中で baffle1 と baffle2 の区分を、faceZone ではなく、wall に設定することで、内部パッチが作成できる。しかし、cyclic や mappedPatch の様に表裏 (master, slave) のパッチを作る必要がある場合は、今回の方法で作成する事になる。

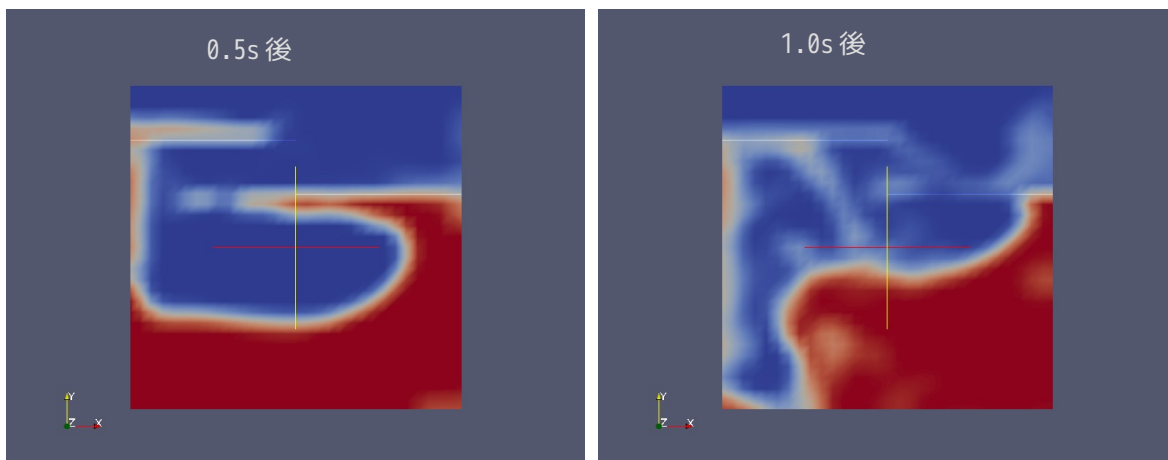
この case を実行してみる。

内部パッチを追加しても、既に設定されている internalField や boundaryField は、そのまま残っているが、新たに作成した内部パッチ (baffle1, baffle2) には、境界条件が設定されていないので、これを設定する。baffle1, baffle2 とともに壁の為、sideW と同じ設定にすれば済む。

TreeFoam 上の  ボタンをクリックして、groEditor を起動して、sideW の境界条件を選択し、cell コピー、cell 貼り付けする事で、同じ条件が設定できる。(下図参照)

	define patch at constant/. (boundary)	U	alpha.water	alpha.water.org	p_rgh
			1...		
atmosW	type wall; inGroups 1(wall);	type pressureInletOutletVelocity; value uniform (0 0 0);	type inletOutlet; inletValue uniform 0; value uniform 0;	type inletOutlet; inletValue uniform 0; value uniform 0;	type totalPressure; rho rho; psi none; gamma 1; p0 uniform 0; value uniform 0;
frontBackW	type wall; inGroups 1(wall);	type slip;	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; gradient uniform 0; value uniform 0;
sideW	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; gradient uniform 0; value uniform 0;
baffle1	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; gradient uniform 0; value uniform 0;
baffle2	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;	type fixedFluxPressure; gradient uniform 0; value uniform 0;

境界条件が設定できたので、計算を開始する。下図が計算させた結果になる。baffleを追加した事によって、流れが変わっている。



### 7-3. cfMeshによる通常メッシュの作成

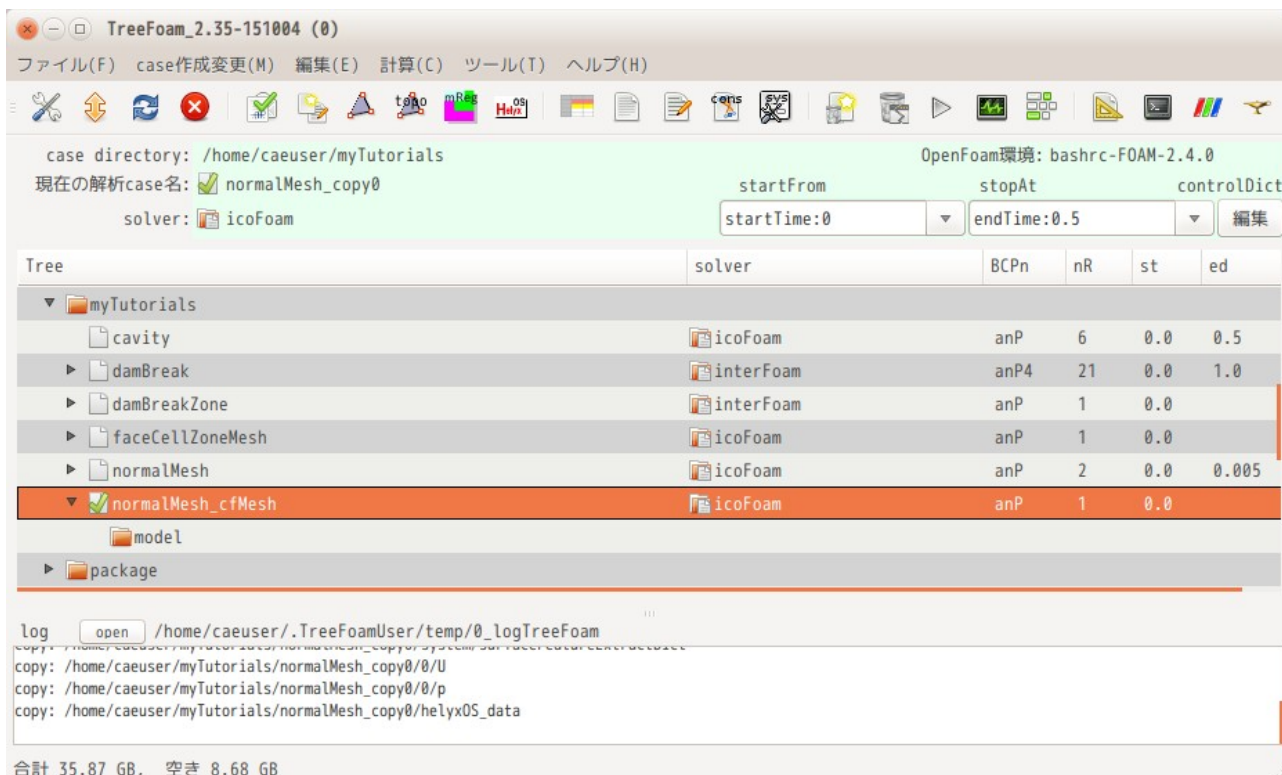
ここで、cfMeshを使ってTreeFoam上でメッシュを作成してみる。cfMeshは、faceZoneやcellZoneを作る事ができないが、通常のメッシュを作る場合は、snappyHexMeshに比べて容易にメッシュを作る事ができるメリットがある。

TreeFoam上でcfMeshを作成する場合、cfMesh専用のdialog「cfMeshによるmesh作成」を準備しているので、これを使ってメッシュを作成する。

#### 7-3-1. caseの作成

sanappyHexMeshで作成した通常メッシュと比較する為に、snappyHexMeshで作成したcase「normalMesh」をコピーして、新しいcase「normalMesh\_cfMesh」を作成する。(7-1項と同様な方法で新しいcaseを作成する。)

フォルダ構成は、最終的に以下の状態になる。noamalMesh\_cfMesh が追加され、このフォルダ内に「model」フォルダが追加されている。




### 7-3-2. レイヤ付きメッシュ作成用の csv ファイル作成

まず、cfMeshを使って stl ファイルからメッシュを作成する時の手順を確認すると、以下の手順で作成する。

- 1) stl ファイルを準備  
patch 毎の stl ファイルを準備する。  
部分的にメッシュサイズを変更する場合は、その部分の形状の stl ファイルを準備する。  
ここまでは、snappyHexMesh の場合と同じ。
- 2) patch 用 stl ファイルの結合  
patch 用に作成した全ての stl ファイルを 1 ヶの stl ファイルに結合する。
- 3) 特徴線の抽出と cfMesh 読み込み用の fms ファイルの作成  
stl ファイルから特徴線を抽出するコマンド (cfMesh のユーティリティ) を実行することで、cfMesh が読み込む形状データ (fms ファイル) も同時に作成できる。以下の様なコマンドを実行することで、特徴線を抽出した fms ファイルが作成できる。  
\$ surfaceFeatureEdges -angle 30 assy.stl model.fms  
  
上記コマンドは、assy.stl ファイルから featureAngle 30 で特徴線を抽出し、その結果を model.fms で書き込む事になる。
- 4) meshDict 作成  
surfaceFile の設定と各 patch の cellSize の設定、部分的に cellSize を設定する場合は、その cellSize と範囲を示す stl ファイルを設定して、meshDict ファイルを作成する。
- 5) メッシュ作成  
meshDict が完成した後は、以下のコマンドを実行してメッシュを作成する。  
\$ cartesianMesh

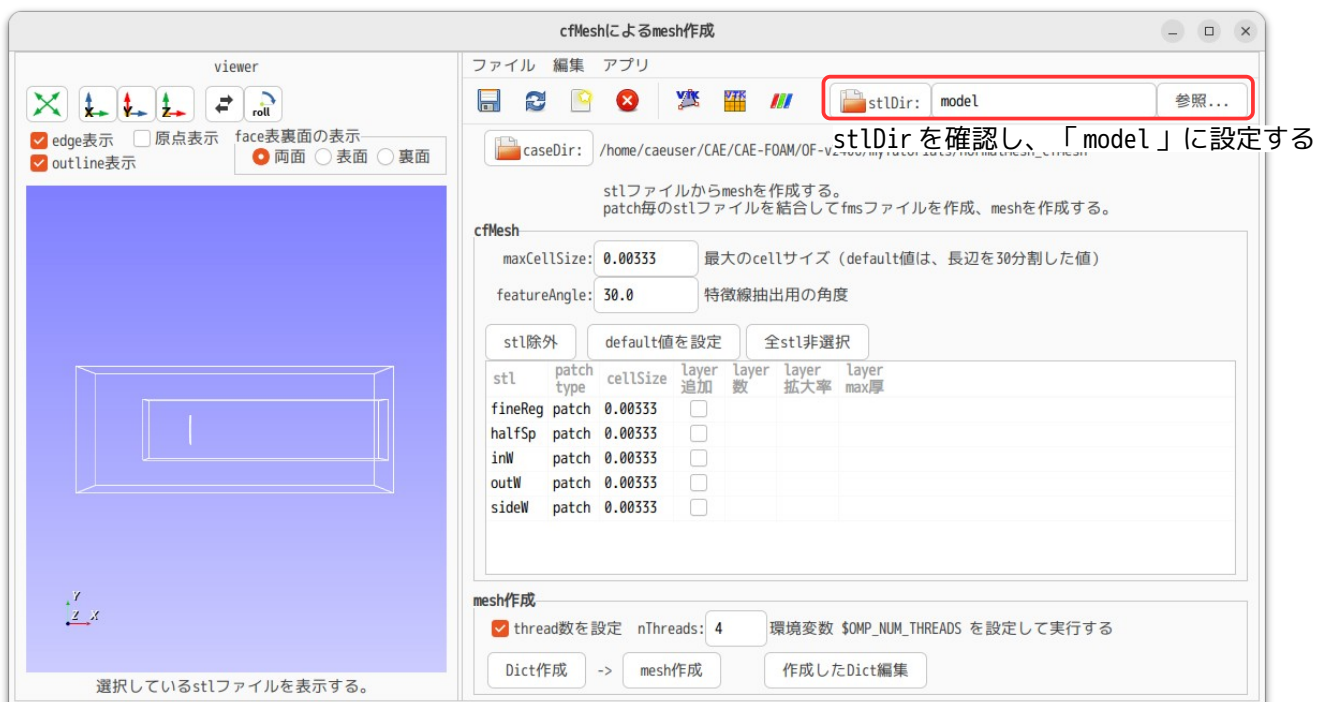
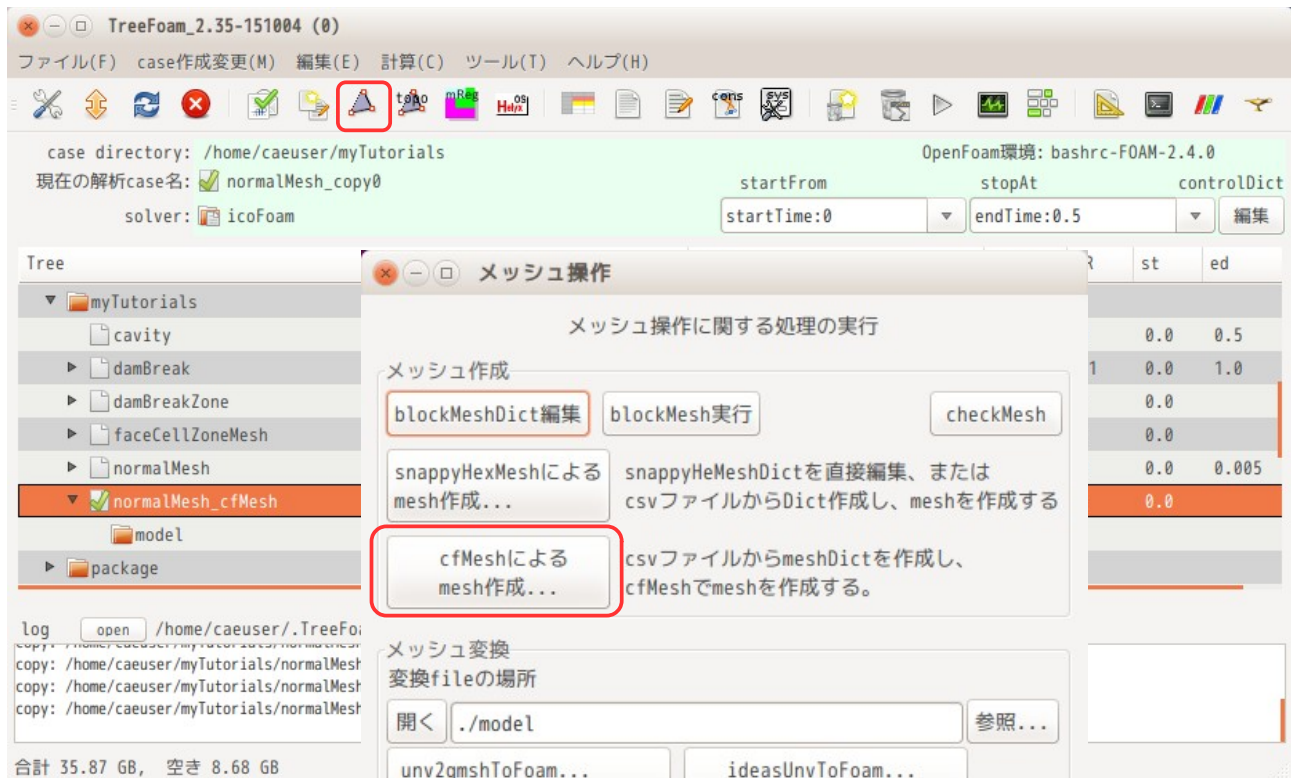
これらの操作が、TreeFoam 上で楽に行える様にしている。stl ファイルを準備した後は、2)~4) までの手続きを csv ファイルで指示し、後は、その指示に従って、stl ファイル結合、特徴線抽出、meshDict を作成する。meshDict 完成後は、「cartesianMesh」コマンドを実行してメッシュを作成する事になる。

以下に、具体的に説明する。

まず、TreeFoam上で、case「normalMesh\_cfMesh」が解析 caseとして設定されている事を確認後、TreeFoam上の  ボタンをクリックして、「メッシュ操作」画面を表示して、「cfMeshによる mesh 作成...」ボタンをクリックする。これによって、「cfMeshによる mesh 作成」画面が表示される。

メッシュ作成は、この画面上を操作して、meshを作成する事になる。

まず、最初に stlDir が「model」フォルダになっているか確認し、model フォルダに設定されていない場合は、「model」に設定し直す。





今回の場合、stl ファイルは、既に「model」フォルダ内に準備されている。また、この stl ファイルは、snappyHexMesh で使用した stl ファイルそのものである為、これらファールの編集の必要はない。

「cfMesh による mesh 作成」画面で設定する項目は、以下になる。（既に default 値が設定されている）

項目	内容	default 値
maxCellSize	最大の cellSize を設定	モデル長辺を 30 分割した値
featureAngle	特徴線を抽出する時の角度を設定	30
各 stl の設定	patchType, stl 毎の cellSize, layer 数等	-

stl の詳細設定は、以下を準備している。

patchType	内容
patch	:patch として設定
wall	:wall として設定
empty	:empty として設定
symmetry	:symmetry として設定
symmetryPlane	:symmetryPlane として設定
regBox	:直方体領域 (stl ファイルで指定) の cellSize を設定
regSph	:球領域 (stl ファイルで指定) の cellSize を設定
face	:面領域 (stl ファイルで指定) の cellSize を設定

まず、maxCellSize をきりの良い「0.004」に設定する。featureAngle は、default のまま。maxCellSize を確定 (enter を入力) すると、その値を基準にして値が再設定される。

この後、stl の詳細設定を行う。以下の様に設定した。（snappyHexMesh と同じ設定）



### 7-3-3. メッシュ作成

meshDict を作成する為に「Dict 作成」ボタンをクリックする。  
この操作によって、分散している stl ファイルが結合され「\_assy.stl」ファイルが作成される。  
このファイルから、model.fms ファイルが作成される。  
meshDict は、前項の設定内容から作成される。

この後、「mesh 作成」ボタンをクリックすることで「cartesianMesh」コマンドを実行し、メッシュを作成する。



cfMeshの場合は、process 並列ではなく、thread 並列でメッシュを作成する。  
thread 並列数を設定して実行するには、「thread 数を設定」をチェックし、「nThreads」に thread 数を入力して実行する。  
尚、「thread 数を設定」をチェックしない場合は、全コアを使って thread 並列してしまう。

今回の csv ファイルから作り出した meshDict は、以下の内容になっている。レイヤを追加しているにも関わらず、snappyHexMeshDict に比べて非常にシンプルに記述できている。

```
-----meshDict の内容-----
// * * * * *
surfaceFile "model/model.fms";
maxCellSize 0.004;
//cellSize of surfaces
surfaceMeshRefinement
{
}
//cellSize of Objects
objectRefinements
{
    fineReg
    {
        cellSize 0.001;
        centre (0.06 0.0 0.0);
        lengthX 0.08;
        lengthY 0.02;
        lengthZ 0.02;
        type box;
    }
}
//cellSize of patches
localRefinement
{
    halfSp
    {
        cellSize 0.001;
    }
    inW
    {
        cellSize 0.004;
    }
    outW
    {
        cellSize 0.004;
    }
    sideW
    {
        cellSize 0.004;
    }
}
```

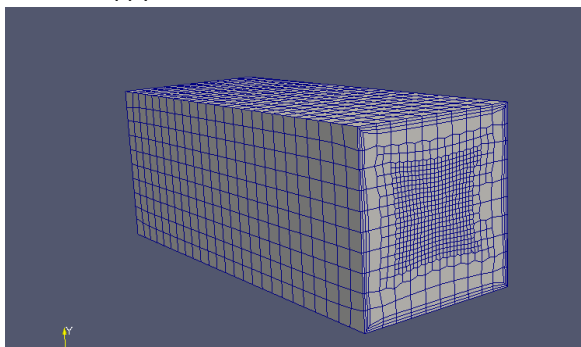
```

    }
}
//set patchName and patchType
renameBoundary
{
    newPatchNames
    {
        halfSp
        {
            newName halfSp;
            type wall;
        }
        inW
        {
            newName inW;
            type patch;
        }
        outW
        {
            newName outW;
            type patch;
        }
        sideW
        {
            newName sideW;
            type wall;
        }
    }
}
//set layers
boundaryLayers
{
    patchBoundaryLayers
    {
        halfSp
        {
            nLayers 3;
            thicknessRatio 1.2;
        }
        sideW
        {
            nLayers 3;
            thicknessRatio 1.2;
        }
    }
}
}
// *****

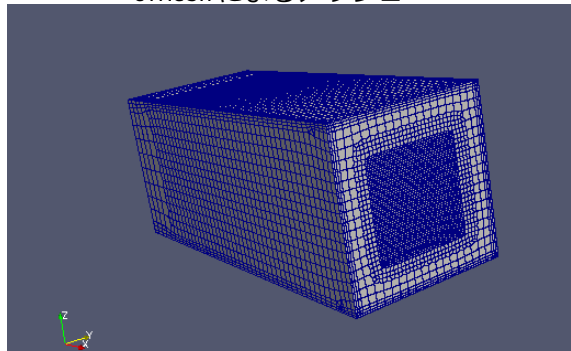
```

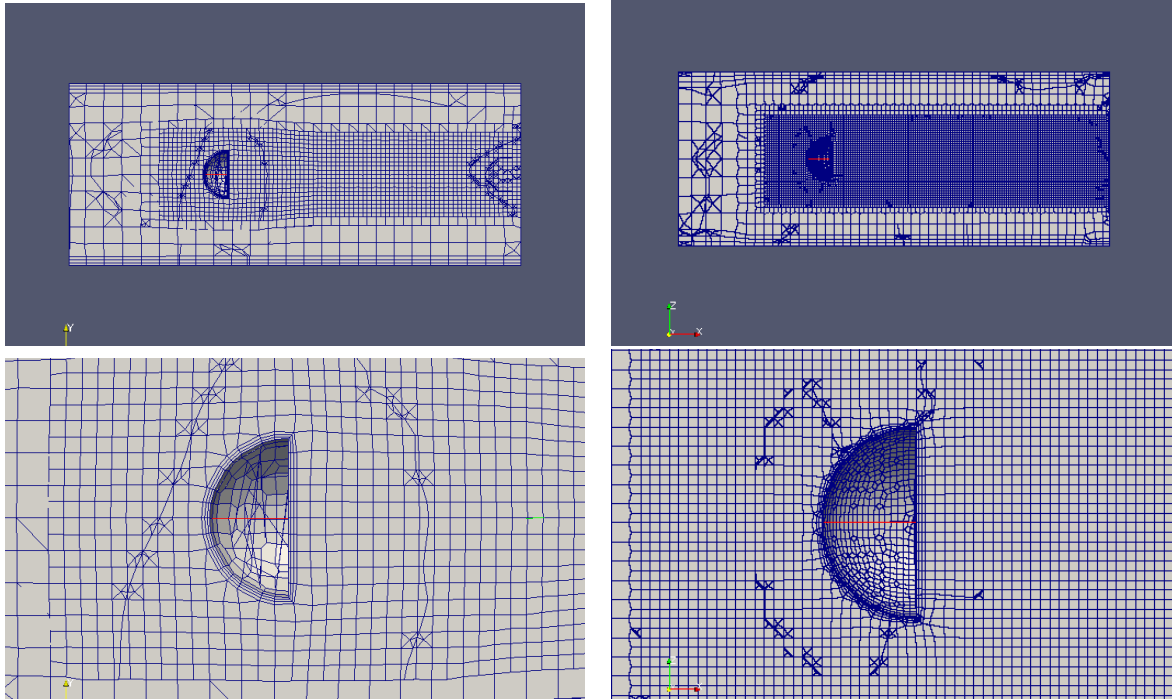
上記 meshDict を使って、作成されたメッシュを確認する。このメッシュは、既に boundary の整合がとれているので、直ぐに paraFoam でメッシュの状況を確認できる。以下は、今回の結果と snappyHexMesh を比較した結果になる。

snappyHexMesh によるメッシュ



cfMesh によるメッシュ



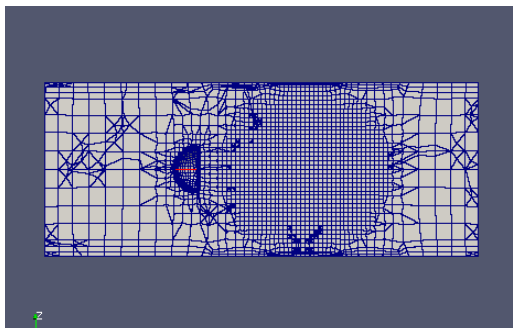


できあがったメッシュを snappyHexMesh と比較すると、fineReg 内のメッシュは、cfMeshの方が細かくなっている。半球の halfSp の直径は 10mm の為、1mm でメッシュを切ると、10 分割されるはずだが、cfMeshの方は、倍の 20 分割されている。この状態は、モデルの大きさを 1000 倍に拡大してメッシュを切り直しても同じ状態。この為、メッシュサイズに関しては、指定した通りのメッシュサイズが必ずしも実現できていないので、メッシュ作成後、望みのメッシュサイズになっているか確認する事が必要か。

部分的にメッシュサイズを変更する為に、上記は、区分「regBox」を使って変更した。  
regBox は、指定された stl ファイルから各方向の最大最小値を算出し、これらの値から中心座標と各方向の長さ求め、直方体の領域を決めている。  
領域設定の方法として、区分「regBox」の他に、区分「regSph」と「face」準備しているの、これらを指定して確認してみる。

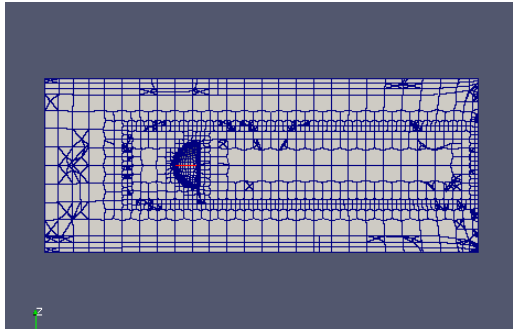
#### <regSph を指定>

stl ファイル「fineReg」は直方体だが、これを区分「regSph」として設定する。regSph は、球領域の設定になる為、球の中心座標と半径を設定するが、regBox と同様に、指定された stl 形状から各方向の最大最小値を読み込み、これら値から球の中心座標を求めている。半径は、各方向の最大最小値から各方向の長さを求め、この平均値から半径を算出して、球領域を設定している。以下の csv ファイルからメッシュを作成したが、設定通り球領域のメッシュが細かくなっている。



#### <face>

stl ファイル「fineReg」を区分「face」として設定する。区分「face」は、stl 形状の面の領域を指定しているので、stl ファイルの形状を忠実に再現して、メッシュを細分化してくれる。  
以下の csv ファイルからメッシュを作成した結果は、設定通り fineReg 領域の面が細かくなっている。



#### 7-4. salome で作成したメッシュを FOAM 形式に変換する例

salome で作成した unv 形式のメッシュを TreeFoam 上で FOAM 形式に変換できるようにしている。  
変換方法は、salome で作成したメッシュを unv 形式で保存し、このメッシュを FOAM 形式に変換する。


また、unv 形式の mesh 内で内部の face や volume をグループ化しておく、変換時にこれらが faceZone や cellZone として作成してくれる。

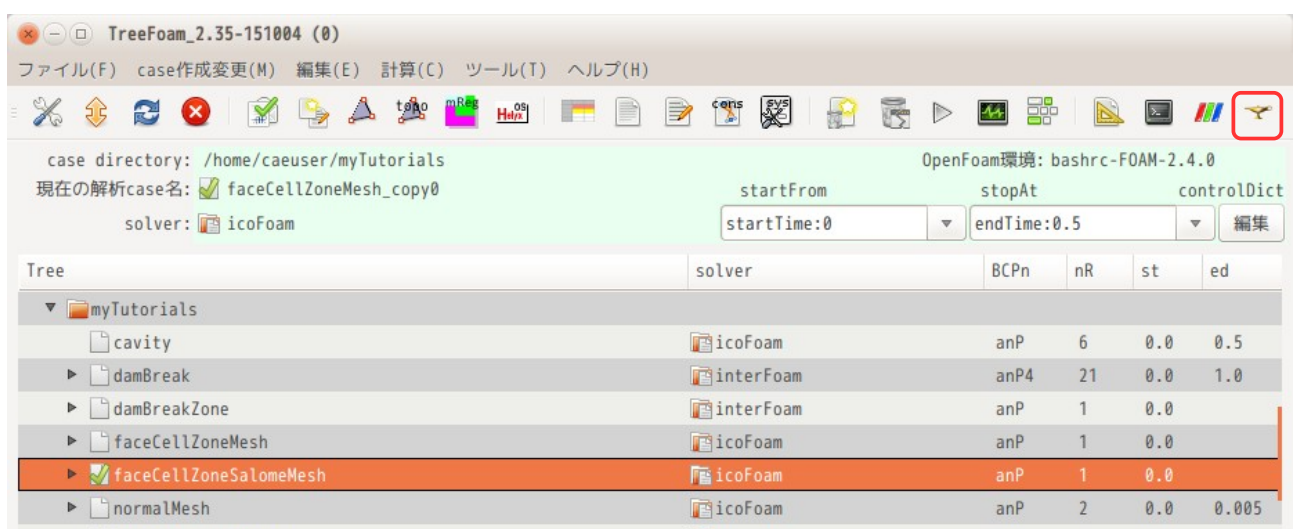
##### 7-4-1. case の作成

モデル形状を faceZone や cellZone を含むモデルとする為、7-2 項で作成したモデルを使うことにする。この為、TreeFoam 上で前項の case 「faceCellZoneMesh」をコピー、case 貼り付けして、新たな「faceCellZoneSalomeMesh」を作成する。最終的に以下の様なフォルダ構成とする。

```
myTutorials
├── faceCellZoneSalomeMesh    メッシュ作成用 case
│   ├── 0
│   ├── constant
│   ├── model                salome が作成したメッシュの保存先
│   └── system
```

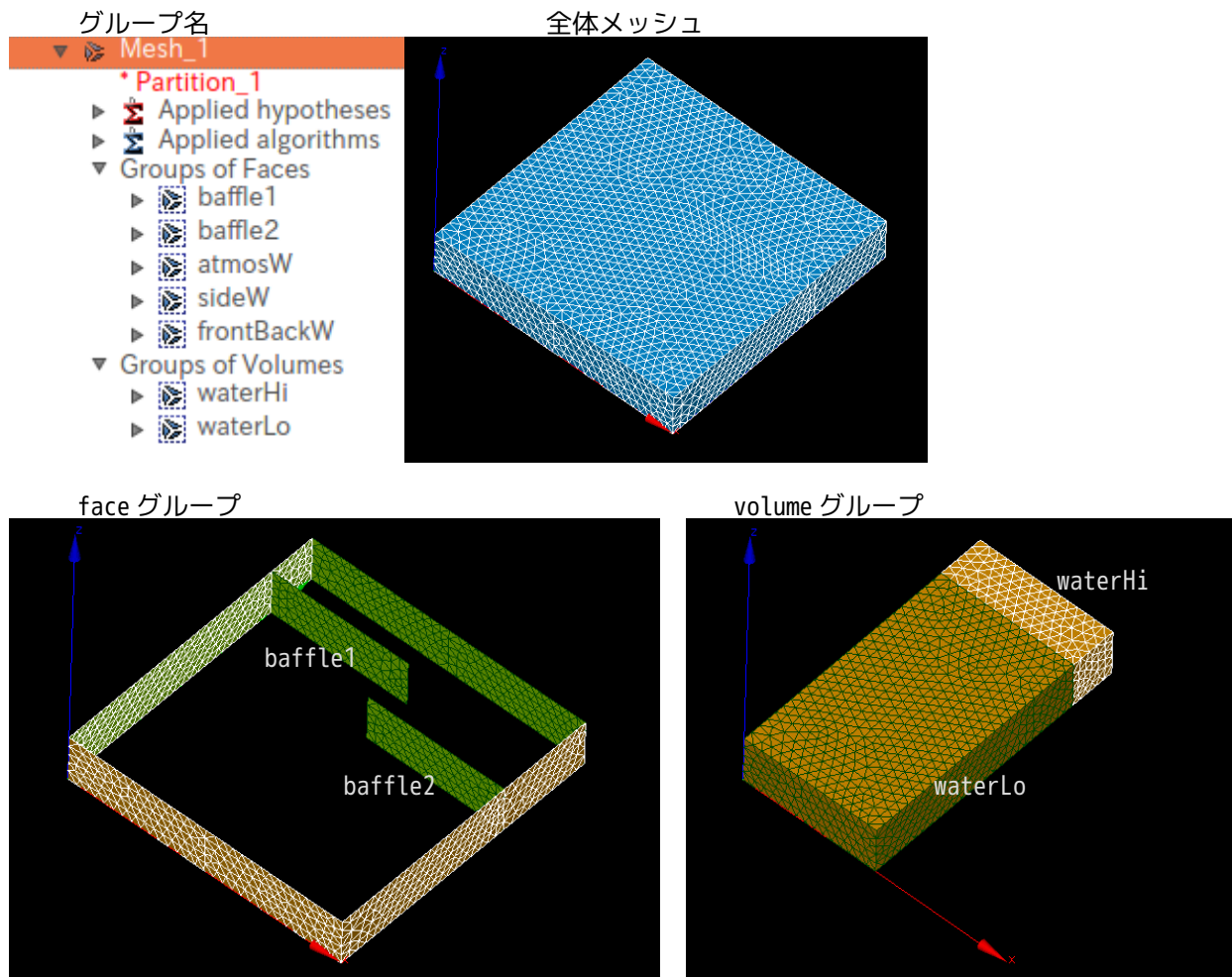
##### 7-4-2. salome によるメッシュ作成

このモデルは、元々 salome で作成しているので、salome 上でメッシュを作成する。  
salome の起動は、TreeFoam 上から、 ボタンをクリックする事で、起動できる。  
(2-3 項の configTreeFoam で salome の項目を設定しておく必要がある。)






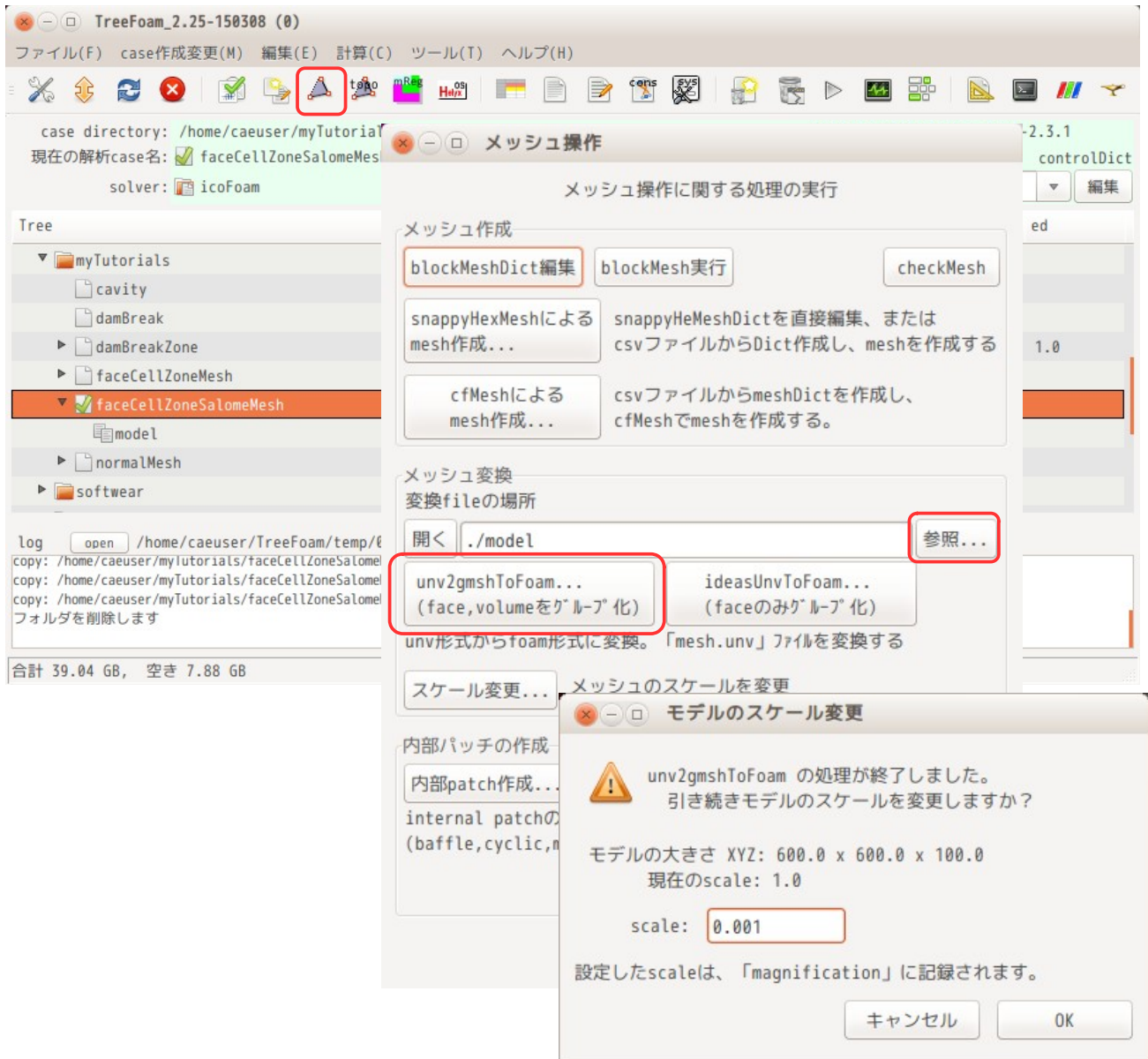
起動後、salome 上で、メッシュを切る。  
 メッシュは、Netgen-1D-2D-3D でメッシュサイズ最大 20mm、最小 10mm で作成している。  
 このメッシュを face と volume でグループ分けし、7-2 項と同じ名称にしている。  
 下図参照。



でき上がった Mesh\_1 を、faceCellZoneSalomeMesh/model フォルダ内に、ファイル名を「mesh.unv」として保存する。(mesh.unv は、\$TreeFoamPath/data/stlFiles/faceZoneSalomeMesh 内に保存しているので、ここから取得できる。)

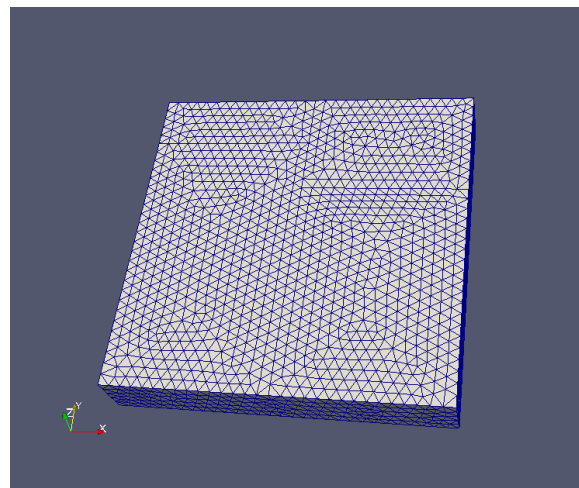
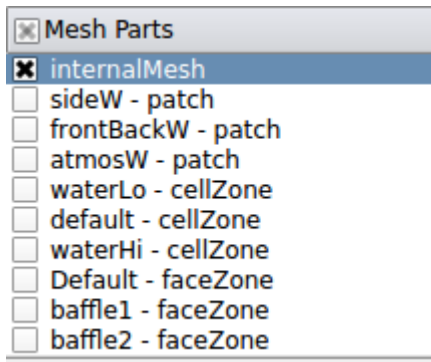
この後、TreeFoam 上の  ボタンをクリックして、「メッシュ操作」画面を表示させ、「参照...」ボタンで、「mesh.unv」が保存されているフォルダ「faceCellZoneSalomeMesh/model」を指定後、「ideasUnvToFoam...(face,volumeをグループ化)」ボタンをクリックする。

変換が終了すると、「モデルのスケール変更」画面が現れるので、画面上に表示されている「モデルの大きさ」を確認し、倍率を入力する。今回は、モデルが mm 単位で作成されている為、倍率を「0.001」に設定した。



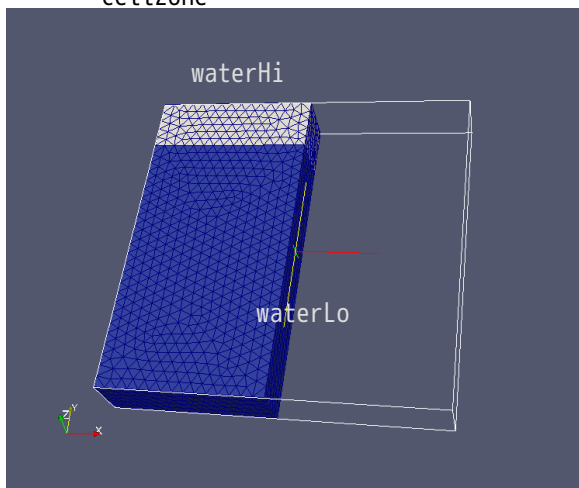
このメッシュ変換は、変換と同時に boundaryField の整合も行っているなので、変換後、直ぐに paraFoam でメッシュが確認できる。

でき上がった FOAM 形式のメッシュを paraFoam で確認すると、以下になる。  
paraFoam から見た「Mesh Parts」にも patch、cellZone、faceZone が確認できる。各々のメッシュを確認した結果、その形状も確認できる。

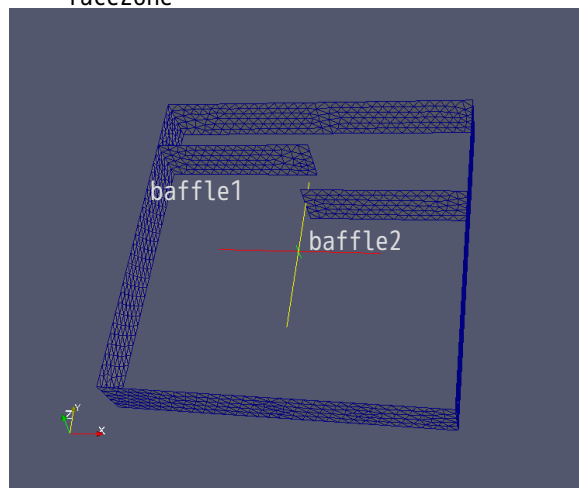


全体メッシュ

cellZone



faceZone



## 8. TreeFoam 内の主なアプリケーション


### 8-1. gridEditor

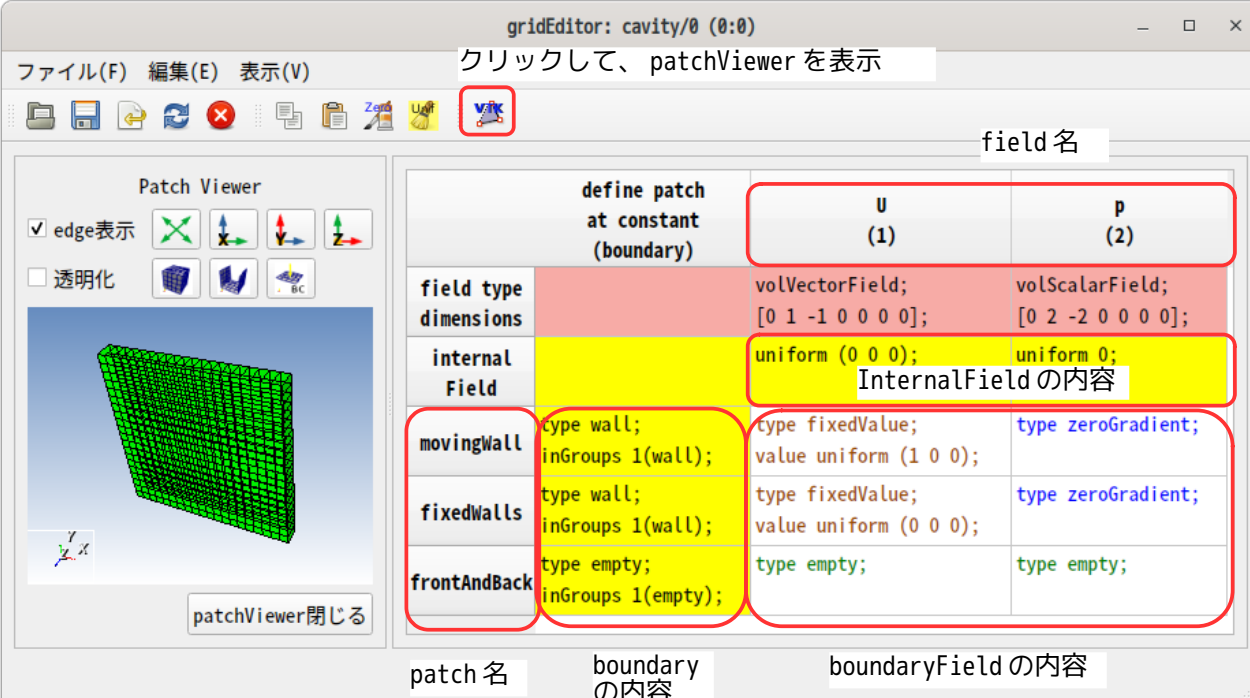
境界条件 (boundary の patch 名と patchType、各 field の internalField や boundaryField) が表形式で編集できる GUI ツールで以下の特徴がある。

- ・ patchViewer 上で patch 形状、場所を確認しながら境界条件が編集できる。
- ・ 表形式の為、セル内のデータを他のセルに copy & paste ができる。また、copy & paste は、gridEditor を複数起動して、この間でも copy & paste が可能になっている。
- ・ field に欠陥 (boundary の整合が取れていない) があっても、読み込むことができ、それを修復 (整合をとる) する事ができる。
- ・ field の書式が ascii、binary、圧縮形式でも、その field データを読み込んで編集でき、同じ形式で保存が可能になっている。
- ・ OpenFOAM-2.1以降 field 内で「\*.」の様な wildCard や「#include」文が使える、OpenFOAM-2.2以降では「\$:wall.U」の様な変数や「inGroups(wall)」の様な patchGroup が使える様になっているが gridEditor でもこれらが扱える様にしている。また、「#includeEtc」文も読むことができる。

#### 8-1-1. 起動画面

gridEditor を起動すると、以下の画面が表示される。以下の例は、tutorials の cavity の境界条件を表示させたものになる。この様に境界条件 (boundary の patch 名と patchType、各 field の internalField と boundaryField) の内容が一望できる。また、下図の赤枠内は編集ができる。

gridEditor 起動後、 ボタンをクリックすると、patchViewer が起動し、画面が現れる。gridEditor の起動は、patch メッシュを読み込まずに起動する (patchViewer が非表示) ので、大規模メッシュの様にメッシュ読み込みに時間が掛かる場合でも gridEditor は、スムーズに起動できる。gridEditor の選択 patch と patchViewer の選択 patch は、連動しているの、patchViewer 上で patch 場所、形状を確認しながら、選択 patch の境界条件が設定できる。




gridEditor: cavity/0 (0:0)

ファイル(F) 編集(E) 表示(V) クリックして、patchViewer を表示

field 名

	define patch at constant (boundary)	U (1)	p (2)
field type		volVectorField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;
movingWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (1 0 0);	type zeroGradient;
fixedWalls	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;


patch 名 boundary の内容 boundaryField の内容

field は、各 timeFolder 内に存在しており、gridEditor が表示している field 内容 (internalField、boundaryField の内容) の読み込み場所は、gridEditor のタイトルバー内に表示されている。今回の場合、「cavity/0/.」から読み込んでいる。読み込む timeFolder を変更する場合は、gridEditor のツールバー内の  ボタンをクリックし、timeFolder を指定して読み込む事になる。

boundary ファイルも constant フォルダだけでなく、各 timeFolder 内に存在する場合もある。gridEditor が表示している boundary の読み込み場所は、field を読み込む timeFolder から constant までさかのぼって polyMesh/boundary を検索し、その timeFolder における最新の boundary を読み込む。今回の読み込み場所は、patchType の列ラベルに表示されており「constant/.」から読み込んでいる。

### 8-1-2. gridEditor の起動

起動方法は、以下の方法で起動できる。

- 1) TreeFoam 上のメニューバー、ツールバーから起動 (  ボタンで起動)

この場合は、TreeFoam 上で解析 case として設定されている case の境界条件を読み取り、gridEditor が起動する。

各 field の internalField と boundaryField の内容を読み取る時、controlDict 内の startFrom を確認し、その時間 (firstTime、startTime、latestTime) に応じた timeFolder 内にある field から internalField と boundaryField の内容を読み取り、表示する。

boundary ファイルの読み込みも同様に、controlDict 内の startFrom を確認し、その時間 (firstTime、startTime、latestTime) から constant までさかのぼって、polyMesh/boundary を検索して最新の boundary ファイルを読み込んでいる。

- 2) TreeFoam 上のポップアップメニューから起動 (「gridEditor 起動...」を選択して起動)

この場合は、選択している case の境界条件を読み取るので、表示させたい case が自由に選択できる。

ポップアップメニューの「gridEditor 起動...」を選択すると、読み込む timeFolder をきいてくるので、これを指定して、境界条件 (各 field や boundary) を読み込む事になる。

boundary の読み込みは、読み込む timeFolder から constant までさかのぼって、polyMesh/boundary を検索して、最新の boundary を読み込む。

### 8-1-3. メニュー構造と内容

gridEditor は、メニューバー、ツールバー、ポップアップメニュー、ダブルクリック操作、ショートカットキーを備えている。

ダブルクリック操作は、マウスカーソルをその場所に合わせると、toolTip が表示され、ここにダブルクリックした時の処理内容が表示される。

ショートカットキーは、プルダウンメニューやポップアップメニュー表示内に、ショートカットキーが可能なメニューについて、その項目の右端に「ctrl-C」の様なショートカットキーが表示される。

gridEditor の基本的操作としては、ポップアップメニューを充実させているので、ポップアップメニューを多用した方がスムーズに操作できる。

#### 8-1-3-1. メニューバーとツールバー

下図の様なメニューバーとツールバーを備えている。



これらの項目と内容は以下。

- 1) ファイル (E)



開く (O)




現在の case 内のフォルダ (timeFolder や regionFolder) を指定して、baoundary と field データを読み込み、新しい gridEditor を起動する。





保存 (S)

表示されている cell のデータ (boundary と field データ) を保存する。



-  CSV 保存 (V)  
表示されている gridEditor のイメージ (ラベル名と全ての cell データ) を csv 形式で保存する。
-  再読み込み (R)  
boundary と各 field データを再読み込みし、再表示する。cell データを初期化できる。
-  閉じる (Q)  
gridEditor を終了する。  
終了時は、このボタンをクリックして終了させる。

## 2) 編集 (E)

- patch 名変更  
選択した行の patch 名を変更する。
- patch 削除  
選択した空パッチ (face が「0」のパッチ) を削除する。選択した行が空パッチでない場合は、削除できない。
-  cell コピー (C)    ctrl-C  
選択した cell データを TreeFoam の clipBoard にコピーする。  
使用頻度が高いのでショートカットキーを割り当てている。
-  cell 貼り付け (P)    ctrl-V  
TreeFoam の clipBoard にコピーされた cell データを cell に貼り付ける。  
使用頻度が高いのでショートカットキーを割り当てている。
- cell 内容を Editor で編集  
cell には、決められた行数しか表示できていない。全て表示しきれていない cell データの場合、データの最後が「...」で終わっている。  
このような cell データの全てを確認したい、編集したい場合には、このメニューを選択する。  
cell に表示する行数は、「cell 内の表示行数・データ数変更」メニューで決定される。

## 3) 表示 (V)

- 全表示/非表示 field の切り替え (A)  
非表示設定した field を隠す、または表示する (全表示) の切り替えを行う。非表示の場合は、ラベルフォントの色が濃い青に変わるので、今の画面が非表示なのか全表示なのかは、判断できる。
- 選択した field を非表示 (H)  
選択した field を非表示設定にする。
- field の再表示、表示順変更 (R)  
非表示設定した field を表示設定に変える、また field の表示順を変更する。(任意の順番で表示させることができる。)  
この設定は、firstTime のフォルダ内に「displayField」の隠しファイルが作成され、ここにその設定が保存されるので、次回起動時には、その設定を読み込み同じ設定で起動する。
- cell 内の表示行数・データ数変更  
ここで、cell 内に表示する行数を指定する。  
計算結果が入っている timeFolder を指定して gridEditor を起動した時、internalField や boundaryField には、nonuniform 形式 (List 形式) のデータが入っており、データの行数は膨大な量になる。この為、全てのデータが cell 内に表示できないので、ここで表示行数を設定し、その行数のみ表示させる設定としている。  
また、読み込む field が binary 形式の場合は、そのデータを ascii 変換して表示させているが、binary 形式はファイルサイズが小さくなるので、大規模モデルの場合が多く、データ量も多くなる。(圧縮ファイルも同様。) この為、binary 形式のデータ全てを ascii に変換せずに、指定したデータ数 (行数) のみ変換するようにしている。この変換するデータ数をここで指定する。  
ascii でも同様な処理を行い、指定した行数以上の余分なデータは取り込まない様にしている。これにより、gridEditor が扱うデータ数が減少するので、処理速度が早く、軽快に動作する。ここで指定するデータ数の値は、cell で表示させる行数以上のデータを指定しておく。



#### 空白 cell に zeroGradient をセット

boundary の整合が取れていない field ファイルを gridEditor で読み込んだ場合、必要な patch 名のデータが存在しない（例：他の case から field をコピーした場合、mesh を入れ替えた場合に相当）為、該当する cell は「空白」で表示される。  
この様な場合、このメニューの実行により、全ての空白 cell を boundary の patchType に応じて以下の様に cell 内容を設定する為、boundary の整合を図る事ができる。

boundary の patchType	cell 内容
empty	type empty;
symmetry	type symmetry;
symmetryPlane	type symmetryPlane;
cyclic	type cyclic;
cyclicAMI	type cyclicAMI;
cyclicACMI	type cyclicACMI;
cyclicSlip	type cyclicSlip;
wedge	type wedge;
その他	type zeroGradient;



#### internalField のクリア

選択した internalField の内容をクリアする。  
指定した internalField が「uniform」形式の場合は、クリアせずそのまま。  
「nonuniform」形式（List 形式）の場合、データが scalar、vector、symmTensor、tensor 等のデータタイプを判断して、値を「0」設定にクリアする。

#### 4) patchViewer



#### patchViewer 起動

patch メッシュを VTK を使って読み込み、3D 表示する。  
patchViewer の選択 patch と表中の選択 patch は、連動しており、patchViewer（表中）の patch を選択すると、表中（patchViewer）側の patch が選択される。

#### 8-1-3-2. ポップアップメニュー

右クリックで表示されるポップアップメニューは、右クリックする場所によって、メニューが異なる。場所ごとのメニュー内容は以下。  
一部のメニュー項目は、メニューバーやツールバーのメニュー項目と同じ項目が存在する。

列名を右クリック			
	define patch at constant/. (boundary)	U	p
field type		volVectorField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;
		cell 内を右クリック	
movingWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (1 0 0);	type zeroGradient;
fixedWalls	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;
行名を右クリック			

#### 1) cell 内のポップアップメニュー

**cell コピー ctrl-C**

選択した cell データを TreeFoam の clipBoard にコピーする。  
使用頻度が高いのでショートカットキーを割り当てている。

**cell 貼り付け ctrl-V**

TreeFoam の clipBoard にコピーされた cell データを cell に貼り付ける。  
使用頻度が高いのでショートカットキーを割り当てている。

**cell 内容を Editor で編集**

cell には、決められた行数しか表示できていない。全て表示しきれていない cell データの場合、データの最後が「...」で終わっている。  
このような cell データの全てを確認したい、編集したい場合には、このメニューを選択する。  
cell に表示する行数は、「cell 内の表示行数・データ数変更」メニューで決定される。

**internalField のクリア**

選択した internalField の内容をクリアする。  
指定した internalField が「uniform」形式の場合は、クリアせずそのまま。  
「nonuniform」形式 (List 形式) の場合、データが scalar、vector、symmTensor、tensor 等のデータタイプを判断して、値を「0」設定にクリアする。

**空白 cell に zeroGradient をセット**

boundary の整合が取れていない field ファイルを gridEditor で読み込んだ場合、必要な patch 名のデータが存在しない (例: 他の case から field をコピーした場合、mesh を入れ替えた場合に相当) 為、該当する cell は「空白」で表示される。  
このような場合、このメニューの実行により、全ての空白 cell を boundary の patchType に応じて以下の様に cell 内容を設定する為、boundary の整合を図る事ができる。

boundary の patchType	cell 内容
empty	type empty;
symmetry	type symmetry;
symmetryPlane	type symmetryPlane;
cyclic	type cyclic;
cyclicAMI	type cyclicAMI;
cyclicACMI	type cyclicACMI;
cyclicSlip	type cyclicSlip;
wedge	type wedge;
その他	type zeroGradient;

**cell 内容をクリア (空白 cell 作成)**

選択した cell 内容をクリア (空白 cell で埋める) する。このコマンドは、上記の「空白 cell に zeroGradient をセット」コマンドと併用すると、cell 内容を default の状態に設定することができる。  
「delete」キーでも同様に空白 cell を作成する事ができる。

**全表示/非表示 field の切り替え**

非表示設定した field を隠す、または表示する (全表示) の切り替えを行う。非表示の場合は、ラベルフォントの色が濃い青に変わるので、今の画面が非表示なのか全表示なのかは、判断できる。

**選択した field を非表示**

選択した field を非表示設定にする。

**field の表示順変更**

非表示設定した field を表示設定に変える、また field の表示順を変更する。(任意の順番で表示させることができる。)  
この設定は、firstTime のフォルダ内に「.displayField」の隠しファイルが作成され、ここにその設定が保存されるので、次回起動時には、その設定を読み込み同じ設定で起動する。

**cell 内の表示行数・データ数変更**

ここで、cell 内に表示する行数を指定する。  
計算結果が入っている timeFolder を指定して gridEditor を起動した時、internalField や boundaryField には、nonuniform 形式 (List 形式) のデータが入っており、データの行数は膨大な量になる。この為、全てのデータが cell 内に表示できないので、ここで表示行数を設定し、その行数のみ表示させる設定としている。  
また、読み込む field が binary 形式の場合は、そのデータを ascii 変換して表示させているが、binary 形式はファイルサイズが小さくなるので、大規模モデルの場合が多く、データ量も多くなる。(圧縮ファイルも同様。) この為、binary 形式のデータ全てを ascii に変換せずに、指定したデータ数 (行数) のみ変換するようにしている。この変換するデー

タ数をここで指定する。  
ascii でも同様な処理を行い、指定した行数以上の余分なデータは取り込まない様にしている。これにより、gridEditor が扱うデータ数が減少するので、処理速度が早くなっている。  
ここで指定するデータ数は、cell で表示させる行数以上のデータを指定する。

## 2) 列名のポップアップメニュー

### 全表示/非表示 field の切り替え

非表示設定した field を隠す、または表示する（全表示）の切り替えを行う。非表示の場合は、ラベルフォントの色が濃い青に変わるので、今の画面が非表示なのか全表示なのかは、判断できる。

### 選択した field を非表示

選択した field を非表示設定にする。

### field の表示順変更

非表示設定した field を表示設定に変える、また field の表示順を変更する。（任意の順番で表示させることができる。）

この設定は、firstTime のフォルダ内に「.displayField」の隠しファイルが作成され、ここにその設定が保存されるので、次回起動時には、その設定を読み込み同じ設定で起動する。

### field コピー

選択している field を TreeFoam の clipBoard にコピーする。（複数をコピーしたい場合は、コピーしたい field を複数個選択して、右クリックする。）  
貼り付ける対象は、自身の gridEditor でも、他の case の gridEditor でも貼り付けはできる。

### field 貼り付け（挿入）

TreeFoam の clipBoard にコピーされている field を選択している列に貼り付ける。（挿入する。）貼り付ける field 名が存在する場合は、field 名を数字付き field 名に変更して挿入する。（同名の field が存在しても、置き換える様な貼付けはしない。）

### field 名変更

選択している field 名を変更する。

### field 削除

選択している field を削除する。

## 3) 行名のポップアップメニュー

### 行コピー

選択している行を TreeFoam の clipBoard にコピーする。

### 行貼り付け

TreeFoam の clipBoard にコピーされた行を貼り付ける。

### patch 名 sort する/しない切り替え

patch 名を sort してアルファベット順に並べ替えて、表示する。これは、表示方法を変更するのみで、保存の順番は変わらない。

### cell 内の表示行数・データ数変更

ここで、cell 内に表示する行数を指定する。

計算結果が入っている timeFolder を指定して gridEditor を起動した時、internalField や boundaryField には、nonuniform 形式 (List 形式) のデータが入っており、データの行数は膨大な量になる。この為、全てのデータが cell 内に表示できないので、ここで表示行数を設定し、その行数のみ表示させる設定としている。

また、読み込む field が binary 形式の場合は、そのデータを ascii 変換して表示させているが、binary 形式はファイルサイズが小さくなるので、大規模モデルの場合が多く、データ量も多くなる。（圧縮ファイルも同様。）この為、binary 形式のデータ全てを ascii に変換せずに、指定したデータ数（行数）のみ変換するようにしている。この変換するデータ数をここで指定する。

ascii でも同様な処理を行い、指定した行数以上の余分なデータは取り込まない様にしている。これにより、gridEditor が扱うデータ数が減少するので、処理速度が早くなっている。  
ここで指定するデータ数は、cell で表示させる行数以上のデータを指定する。

### patch 名変更

選択した行の patch 名を変更する。

#### 新しい空 patch 追加

追加したい空 patch の個数分の行を選択し、メニューを実行すると、選択した個数分の空 patch が追加される。

#### 空 patch 削除

削除したい空 patch を選択して、メニューを実行すると、選択した空 patch が削除される。

#### 全ての空 patch 削除

gridEditor が表示している全ての空 patch (internalField を除く黄色の行) を削除する。

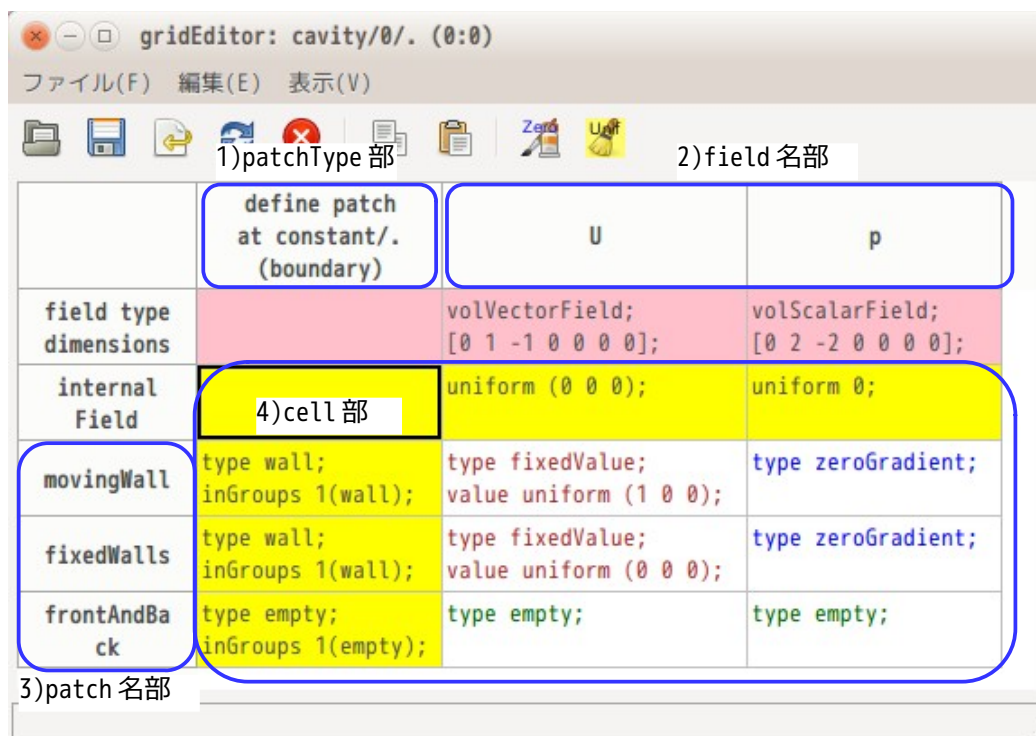
#### 変数定義行 (空) の表示/非表示切り替え

field 内で「\$iniTemp」等の変数を定義する行 (水色の行) を追加する。ただし、既に変数を定義している field を読み込んだ場合は、この設定にかかわらず、変数行は、表示される。(この場合は、変数行を消す事ができない。)

変数行は、boundaryField 内と外に定義できるので、この設定を「表示する」にするとこの行が 2 行現れる。

### 8-1-3-2. ダブルクリック操作

ダブルクリックする場所によって、その処理が異なってくる。



#### 1) field 名部をダブルクリック

ダブルクリックした field を editor で開く。

field が binary 形式の場合は、データを ascii 変換した後で表示する。

データ形式が nonuniform 形式 (List 形式) の場合は、データ量が膨大になるので、ascii、binaryにかかわらず、List 形式のデータは、「cell 内の表示行数・データ数変更」で設定したデータ数に省いて、editor で開く様にしている。この為、editor で編集できるデータは、List 形式以外の項目が編集できることになる。

#### 2) patchType 部をダブルクリック

boundary ファイルを editor で開く。

#### 3) patch 名部をダブルクリック

ダブルクリックした patch 名をが変更できる。

#### 4) cell 部をダブルクリック



cell 内容が編集できる。cell 内容の最後が「...」で終わっている場合、全ての内容が表示されていない状態になっており、この場合は、ダブルクリックにより、その cell 内容の全てを editor で開き、確認・編集ができる。

#### 8-1-4. field 内変数や patchGroup、include 文の扱い

field 内で「\$iniTemp」や「\$:wallBC.U」(又は「\$!wallBC/U」) の様な変数、「".\*"' の様な wildCard、boundary で設定した patchGroup (inGroups) が使えるが、この内「\$:wallBC.U」の変数タイプ、wildCard、patchGroup は、構造が複雑で、その設定内容が直感的に理解できない。この為、gridEditor 上では、これらを解釈した結果を表示させるようにしている。

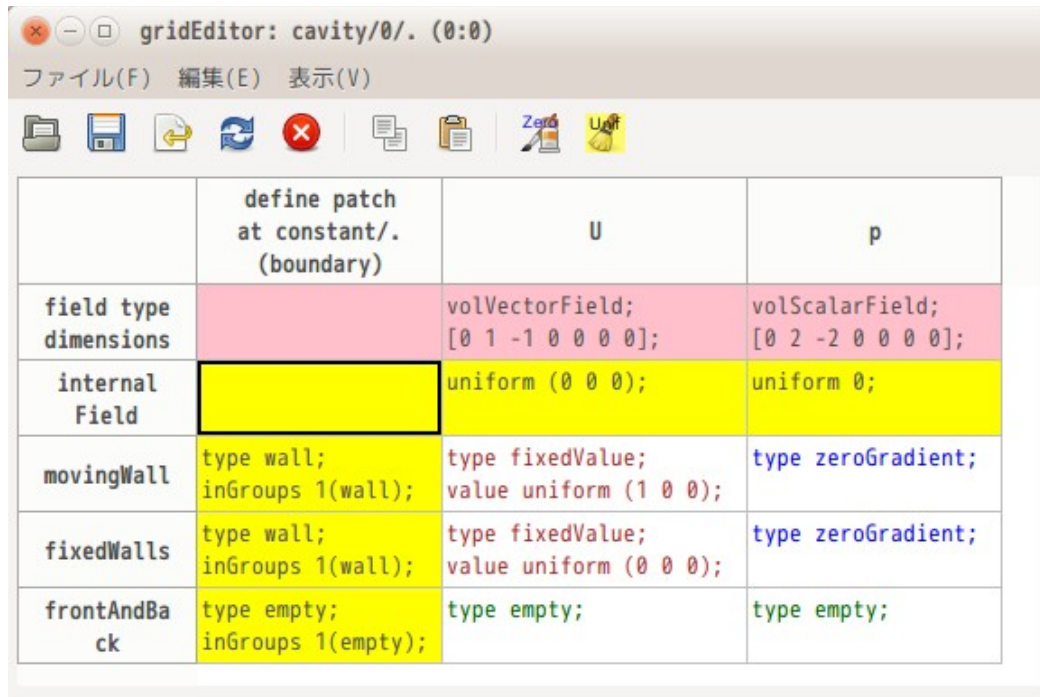
ただし、「\$iniTemp」の変数タイプについては、階層がなく直感的に理解できる事と、この変数がある事によって修正が楽に行える為、gridEditor 上では解釈せずそのまま残して表示させている。field 内の「#include」や「#includeEtc」の内容についてもその内容を読み込み、同様な処理を行っている。

保存は、gridEditor が表示している内容 (変数を解釈した結果) を保存する。「#include」や「#includeEtc」を含む行を保存する場合は、これを解釈した結果を保存する為、include 行は不要になる為、これらの行は、削除される。

##### 8-1-4-1. gridEditor による field 内変数の入力例

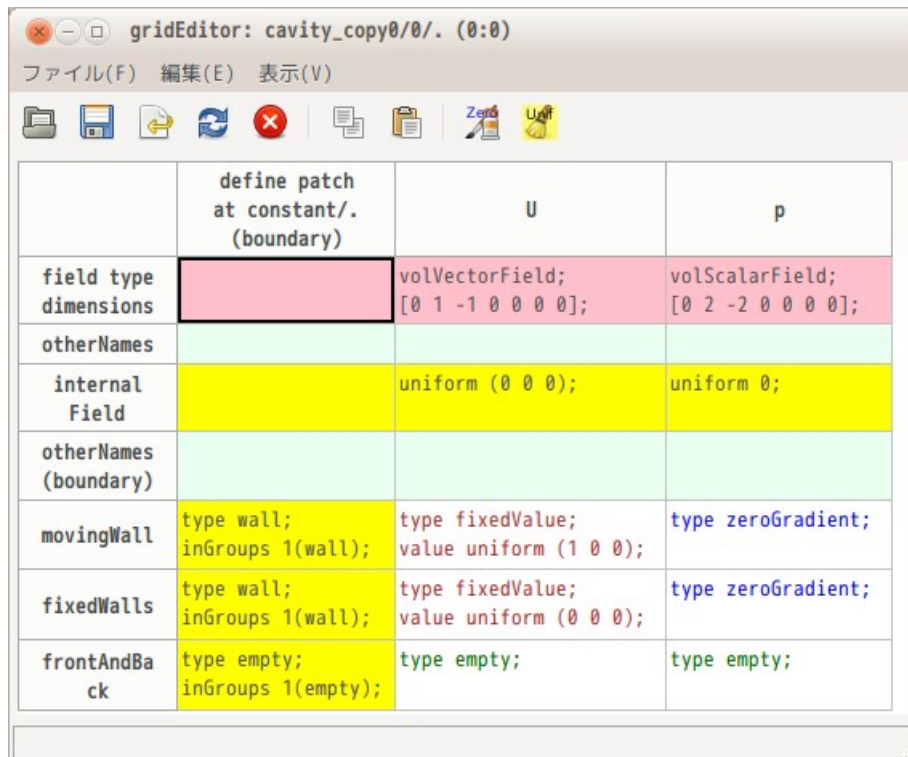
tutorials の cavity を例にとって、gridEditor 上で field 内変数を入力してみる。以下のリストは、オリジナルの boundary と U field の内容になる。この内容を gridEditor で表示させると以下になる。

<pre> &lt; U field &gt; // * * * * * //  dimensions      [0 1 -1 0 0 0]; internalField    uniform (0 0 0); boundaryField {     movingWall     {         type      fixedValue;         value      uniform (1 0 0);     }      fixedWalls     {         type      fixedValue;         value      uniform (0 0 0);     }      frontAndBack     {         type      empty;     } }  //***** // </pre>	<pre> &lt; p field &gt; // * * * * * //  dimensions      [0 2 -2 0 0 0]; internalField    uniform 0; boundaryField {     movingWall     {         type      zeroGradient;     }      fixedWalls     {         type      zeroGradient;     }      frontAndBack     {         type      empty;     } }  //***** // </pre>
---	---



この内容に対し、gridEditor 上で field 変数を追加して保存し、結果を確認してみる。

field 変数を追加するためには、patch 名（行ラベル）部を右クリックしてポップアップメニューを開き「変数定義行（空）の表示/非表示切り替え」を選択する。この操作により、下図の様に gridEditor 上に変数定義行（水色の行）が現れる。




水色の変数定義行が 2 本表示されているが、これは上段が boundaryField の外で変数を定義し、下段が boundaryField の中で変数を定義する為のもの。上段で定義した変数は、internalField や boundaryField 内で使用できる。下段は、boundaryField 内のみでしか使用できない。

ここで、以下の様に変数を定義してみる。上段で初期値 iniU、iniP を定義してこの変数を internalField

で使い、下段で moveU と zeroU を定義してこの変数を boundaryField で使う設定にしている。

	define patch at constant/. (boundary)	U	p
field type		volVectorField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
otherNames		iniU (0 0 0);	iniP 0;
internal Field		uniform \$iniU;	uniform \$iniP;
otherNames (boundary)		moveU (1 0 0); zeroU (0 0 0);	
movingWall	type wall; inGroups 1(wall);	type fixedValue; value uniform \$moveU;	type zeroGradient;
fixedWalls	type wall; inGroups 1(wall);	type fixedValue; value uniform \$zeroU;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;

この状態で、 ボタンをクリックして、保存する。保存後、U, p field の内容を確認すると、以下になる。

```

< U field >
// * * * * *

dimensions      [0 1 -1 0 0 0 0];
iniU (0 0 0);

internalField    uniform $iniU;

boundaryField
{
    moveU (1 0 0);
    zeroU (0 0 0);

    movingWall
    {
        type      fixedValue;
        value      uniform $moveU;
    }
    fixedWalls
    {
        type      fixedValue;
        value      uniform $zeroU;
    }
    frontAndBack
    {
        type      empty;
    }
}
//*****

< p field >
// * * * * *

dimensions      [0 2 -2 0 0 0 0];
iniP 0;

internalField    uniform $iniP;

boundaryField
{
    movingWall
    {
        type      zeroGradient;
    }
    fixedWalls
    {
        type      zeroGradient;
    }
    frontAndBack
    {
        type      empty;
    }
}
//*****

```

各々の field に変数が boundaryField の内外に追加されている事が判る。

今の設定は、値を変数に置き換えたのみの為、このまま実行しても結果は変わらない。しかし、変数が定義できる事は、変数の値を修正する事で、その変数を使っている場所の値を全て変更できるメリットがある。

#### 8-1-4-2. patchGroup と wildCard の使用例

前項と同様に tutorials の cavity を使って、patchGroup と wildcard を使って設定してみる。  
尚、wildCard については、正規表現であり、gridEditor 側も正規表現でマッチングを確認している。

U field について、patchGroup と wildcard を使って以下の様に        内を書き換え、gridEditor で読み込み表示させた結果が以下になる。patchGroup と wildcard を解釈した結果が表示されている。

```
< boundary >
// * * * * *

3
(
    movingWall
    {
        type            wall;
        inGroups         2(move wall);
        nFaces           20;
        startFace        760;
    }
    fixedWalls
    {
        type            wall;
        inGroups         1(wall);
        nFaces           60;
        startFace        780;
    }
    frontAndBack
    {
        type            empty;
        inGroups         1(empty);
        nFaces           800;
        startFace        840;
    }
)

//*****

< U field >
// * * * * *

dimensions      [0 1 -1 0 0 0];

internalField    uniform (0 0 0);

boundaryField
{
    move
    {
        Type            fixedValue;
        Value            iniform (1 0 0);
    }
    "fix.*"
    {
        type            fixedValue;
        value            uniform (0 0 0);
    }
    frontAndBack
    {
        type            empty;
    }
}

//*****
```

gridEditor で表示させた結果 (解釈した結果が表示)

	define patch at constant/. (boundary)	U	p
field type		volVectorField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;
movingWall	type wall; inGroups 2(move wall);	type fixedValue; value uniform (1 0 0);	type zeroGradient;
fixedWalls	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;

この状態を保存すると、解釈した結果を保存するので、結果的に field の内容は、表示通りのイメージの状態が保存される。(wildCard や move の patch 名は削除される。)

## 8-1-4-3. tutorials 内での使用例

tutorials 内で field 内変数を多用している「compressible/rhoPimpleFoam/RAS/annularThermalMixer」の内容を確認してみる。(0F-11, 12, 13 の場合「foamRun/fluid/annularThermalMixer」)

まず、該当する tutorials をコピーして新しい case を作成し、「./Allrun」を実行して case を完成させる。この後、U field を確認すると、以下の様に設定されている。非常にシンプルに記述されているが、patch 内容はこの他に include ファイルの内容や boundary の内容を確認しないと解らない。

また、include ファイル中に include 文を含む場合もあり、さらに「\$:outlet.U」タイプの変数は、「\$:ini.outlet.U」の様に、ネスティングさせて記述ができる様になっているので、patch 内容を理解する事が難しくなっている。gridEditor では、これらの解釈方法を再帰的呼び出しで解釈させている為、これらのネスティング深さの制限なく解釈して表示できる。

```
<U field>
// ***** //
#include "${FOAM_CASE}/constant/caseSettings"
dimensions      [0 1 -1 0 0 0];
internalField    uniform (0 0 0);
boundaryField
{
    innerInlet
    {
        type      fixedValue;
        value      uniform $:innerInlet.U;
    }
    outerInlet
    {
        type      fixedValue;
        value      uniform $:outerInlet.U;
    }
    outlet        { $:outlet.U; }
    staticWalls   { $:wall.U; }
    movingWalls   { $:movingWall.U; }
    #includeEtc "caseDicts/setConstraintTypes"
}
// ***** //
```

この内容をそのまま gridEditor で表示させた結果が以下になる。

U field を editor で開いた結果は難解であるが、これを gridEditor で表示させた場合は、以下の様に表示され、各 patch の境界条件が素直に理解できる状態になっている。  
(#include や #includeEtc 文も解釈できている。)

尚、gridEditor 上の boundary の内容はそのまま表示させると、行数が多くなってしまうので、inGroups の行を 1 行にまとめ直して、表示させている。



gridEditor: annularThermalMixer/0/. (0:1)			
ファイル(F) 編集(E) 表示(V)			
	define patch at constant/. (boundary)	T	U
field type dimensions		volScalarField; [0 0 0 1 0 0 0];	volVectorField; [0 1 -1 0 0 0 0];
internal Field		uniform 293;	uniform (0 0 0);
innerInlet	type patch; inGroups 1(inlet);	type fixedValue; value uniform 233;	type fixedValue; value uniform (0 0 0.2);
outerInlet	type patch; inGroups 1(inlet);	type fixedValue; value uniform 293;	type fixedValue; value uniform (0 0 0.1);
innerOutlet	type patch; inGroups 1(outlet);	type inletOutlet; inletValue uniform 293; value \$inletValue;	type pressureInletOutletVelocity; value uniform (0 0 0);
outerOutlet	type patch; inGroups 1(outlet);	type inletOutlet; inletValue uniform 293; value \$inletValue;	type pressureInletOutletVelocity; value uniform (0 0 0);
rotorBlades	type wall; inGroups 2(movingWalls wall);	type zeroGradient;	type movingWallVelocity; value uniform (0 0 0);
rotorBlades_slave	type wall; inGroups 2(movingWalls wall);	type zeroGradient;	type movingWallVelocity; value uniform (0 0 0);
shaft	type wall; inGroups 2(movingWalls wall);	type zeroGradient;	type movingWallVelocity; value uniform (0 0 0);
statorBlades	type wall; inGroups 2(staticWalls wall);	type zeroGradient;	type fixedValue; value uniform (0 0 0);
statorBlades_slave	type wall; inGroups 2(staticWalls wall);	type zeroGradient;	type fixedValue; value uniform (0 0 0);
walls	type wall; inGroups 2(staticWalls wall);	type zeroGradient;	type fixedValue; value uniform (0 0 0);
AMI1	type cyclicAMI; inGroups 2(cyclicAMI baffleFaces); matchTolerance 0.0001; transform noOrdering; neighbourPatch AMI2;	type cyclicAMI;	type cyclicAMI;
AMI2	type cyclicAMI; inGroups 2(cyclicAMI baffleFaces); matchTolerance 0.0001; transform noOrdering; neighbourPatch AMI1;	type cyclicAMI;	type cyclicAMI;

この結果を保存すると、変数を解釈した状態が保存される為、以下の様書き直されて保存される。

include ファイルは、不要になる為、削除される。

```

// * * * * *
// #include "${FOAM_CASE}/constant/caseSettings"
dimensions      [0 1 -1 0 0 0];
internalField    uniform (0 0 0);
boundaryField
{
    innerInlet
    {
        type      fixedValue;
        value      uniform (0 0 0.2);
    }
    outerInlet
    {
        type      fixedValue;
        value      uniform (0 0 0.1);
    }
    innerOutlet
    {
        type      pressureInletOutletVelocity;
        value      uniform (0 0 0);
    }
    outerOutlet
    {
        type      pressureInletOutletVelocity;
        value      uniform (0 0 0);
    }
    rotorBlades
    {
        type      movingWallVelocity;
        value      uniform (0 0 0);
    }
    rotorBlades_slave
    {
        type      movingWallVelocity;
        value      uniform (0 0 0);
    }
    shaft
    {
        type      movingWallVelocity;
        value      uniform (0 0 0);
    }
    statorBlades
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    statorBlades_slave
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    walls
    {
        type      fixedValue;
        value      uniform (0 0 0);
    }
    AMI1
    {
        type      cyclicAMI;
    }
    AMI2
    {
        type      cyclicAMI;
    }
}
// * * * * *

```

変数と wildCard (又は変数と wildCard の組み合わせ) を記述順に解釈していき、この結果を patchGroup または patch 名に入れる。その後、patchGroup を解釈し、この結果を patch 名に入れる。最後に、patch 名の内容を記述順に確認し、その内容に書き換える。

この様に、解釈していく途中で patch 名の内容を逐次書き換えて行くが、最終的に「patch 名に直接記述」したものがあれば、この内容で確定する。(wildCard や patchGroup でその patch 内容が決定されていても、それに関係なく「patch 名に直接記述」があれば、その内容で確定する。)

同じタイプの中では、記述順に内容を書き換えていく。（最後の記述内容で決定される。）

patchGroupについては、inGroups内で定義された順で値を書き換えていく。

例:「finGroups 3(moveWall1, fixWall, sideWall)」の場合、moveWall→fixWall→sideWallの順で確認し、最後に確定した内容で決定される。

タイプ	例	解釈順
変数、wildCard patchGroup patch 名に直接記述	\$:outlet.U、".*" inGroups 1(wall) movingWall	↓

### 8-1-5. binary 形式の扱い


gridEditor では、field の書式が binary でも、それを読み込み表示して、編集できる。

binary 形式の編集・保存は、binary を ascii 形式に変換して編集し、保存時は ascii を binary に変換して保存する。

### 8-1-5-1. OpenFOAM の binary 形式

実際に tutorials の cavity を使って、binary 形式のファイルの内容を確認してみる。

この case の要素数を減らすために、blockMesh の分割数を (3 x 3 x 1) に変更してメッシュを切り直す。controlDict 内の writeFormat を「binary」に変更して、solver を走らせ、binary 形式のファイルを作成する。でき上がった timeFolder 内の U field を editor で開き内容を確認すると、以下が表示される。

ファイルの書式は、FoamFile 中の「format」の内容で、そのファイルが ascii か binary か判断できる。binary 部は、 部のみで他は ascii で記述されている。

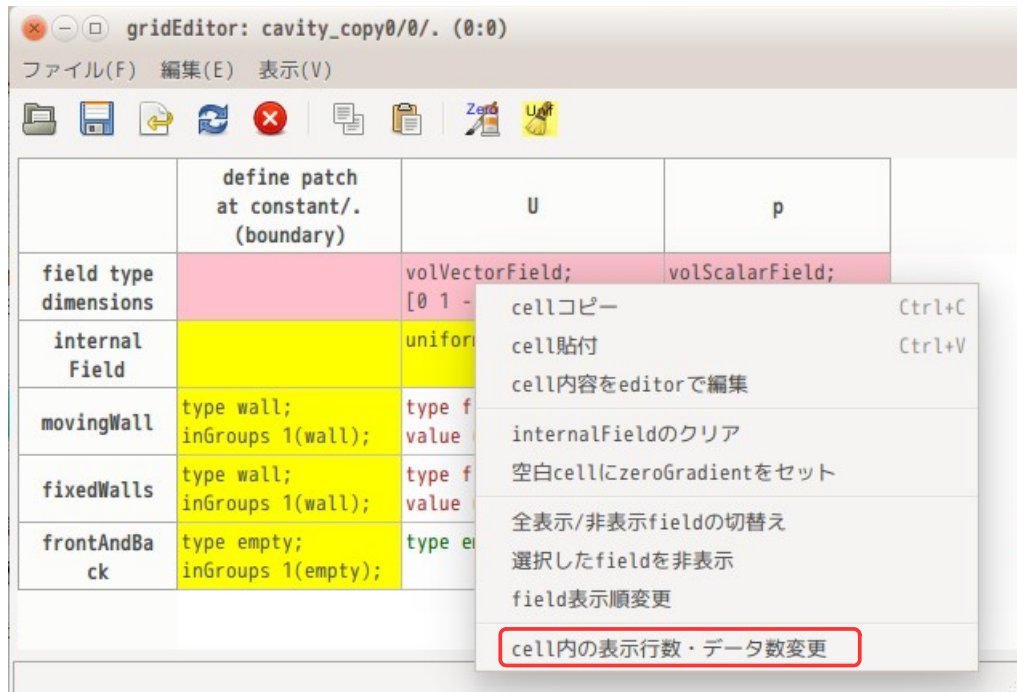
[illegible]

binary 部を読む為には、直前の変数 (vector 等) とその組数 (今回の場合、vector が 9 組) を確認した上で、その個数分の binary データを読み込む事になる。変数は以下のものを読み込む事ができる。

变数	type	byte 数
scalar	double	1 x 8 byte
vector	double	3 x 8
symmTensor	double	6 x 8
tensor	double	9 x 8
label	int	1 x 4
faceList	int	1 x 4
bool	bool	1 x 1

binary ファイル自体に ascii 文字と binary データが混在しているので、ascii 部と binary 部に分けて読み込む。binary 部は、予め設定されているデータ数分を ascii 変換する。

ascii 変換するデータ数は、以下の様に cell のポップアップメニューの「cell 内の表示行数・データ数変更」を選択して表示される「表示行数・データ数の設定」画面内で設定する。







この後、分けて読み込んだ ascii 部のデータに、ascii 変換した binary 部を挿入して、ascii ファイルとして完成させる。でき上がった ascii ファイルを「TreeFoam/temp/」フォルダに「U.0/U」として保存し、このファイルを editor で読み込み編集する方法をとっている。

gridEditor 上から、binary 形式のファイルを実際に開いてみる。下図が、cavity を binary で計算させた後、timeFolder「0.5」を gridEditor で表示させた結果になる。binary データが ascii 変換されて表示されている。

gridEditor: cavity_copy0/0.5/. (0:3)				
ファイル(F) 編集(E) 表示(V)				
	define patch at constant/. (boundary)	U	p	phi
field type		volVectorField;	volScalarField;	surfaceScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];	[0 3 -1 0 0 0];
internal Field		nonuniform List<vector> 9 ( (-0.0462046515435 0.0408637298982 0.0) (-0.0972003750338 0.00480091815208 0.0)...	nonuniform List<scalar> 9 ( 4.09366552723e-09 0.0769624380778...	nonuniform List<scalar> 12 ( -2.6185383348e-05 2.61853814353e-05...
movingWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (1 0 0);	type zeroGradient;	type calculated; value uniform 0;
fixedWalls	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty; value nonuniform 0;

この後、U field を editor で開く為に、field 名「U」(列ラベル「U」) をダブルクリックする。この操作で、editor で U field を開き、その内容が確認できる。

以下が editor で U field を開いた結果になる。binary 部 (部) が ascii 変換されて、数字が確認できる。基本的に binary データ部は編集できないが、ascii 部は編集できる。ascii 部を編集後、保存すると、binary 部に元の binary データを挿入し保存される。

```

/*-----* C++ *-----*/
|=====|
| \      / | F ield      | OpenFOAM: The Open Source CFD Toolbox
|  \    /  | O peration | Version: 2.3.0
|   \  /   | A nd       | Web: www.OpenFOAM.org
|    \/    | M anipulation|
|-----|
FoamFile
{
    version      2.0;
    format       binary;
    class        volVectorField;
    location     "0.5";
    object       U;
}
// *****

dimensions      [0 1 -1 0 0 0];

internalField    nonuniform List<vector>
9
(
(-0.0462046515435 0.0408637298982 0.0)
(-0.0972003750338 0.00480091815208 0.0)
(-0.0434501396518 -0.0425435040631 0.0)
(-0.0449110781197 0.144268555566 0.0)
(-0.116741501046 -0.00457649800858 0.0)
(-0.0216978539534 -0.152800969997 0.0)
(0.226799172932 0.112710025279 0.0)
(0.174552642635 0.00922281531272 0.0)
(0.240340455548 -0.100962433547 0.0)
...U.0...
);

boundaryField
{
    movingWall
    {
        type          fixedValue;
        value          uniform (1 0 0);
    }
    fixedWalls
    {
        type          fixedValue;
        value          uniform (0 0 0);
    }
    frontAndBack
    {
        type          empty;
    }
}
// *****

```

ascii 変換された binary 部  
20 行分あるはずだが、データは全部で  
9 ケしかない為、9 行を表示している。  
(この部分は編集できない。)

## 8-2. topoSetEditor

特定のメッシュを抽出して、加工する OpenFOAM ユーティリティの topoSet を GUI 上で操作できる様にしたもの。このメッシュ操作は、topoSetDict を作成し topoSet を実行するだけだが、topoSetDict の作成が重要で、ここが GUI 上で行える様にしている。

OF-13 では、topoSet コマンドがなくなっている為、topoSetEditor は、使えない。

### 8-2-1. topoSet のコマンド構造

例として、cellZone「waterHi」から cellSet「waterHiSet」を作り出すコマンドを考えると、以下になる。  
 以下の様に、topoSetDict の actions 内に、この処理を書き込むことになる。

```
// * * * * * //
actions
(
  {
    name    waterHiSet;      //作り出す名前
    type    cellSet;        //作り出すタイプ (今回は、「cellSet」)
    action  new;            //新しく作り出す為、actionは「new」
    source  zoneToCell;     //作り出す方法 (今回は「zoneToCell」)
    sourceInfo
    {
      name  "waterHi";      //元となる cellZone 名
    }
  }
);
// * * * * * //
```

この構造を分解すると、下記の様に3ヶに分類できる。

name	waterHiSet;	}	1) result : 結果のタイプと名称
type	cellSet;		
action	new;	}	2) action : 処理内容
source	zoneToCell;	}	3) source : source と result の組み合わせと、source 名
sourceInfo	{		
	name "waterHi";		
	}		


topoSet は、action 内容によって source 不要の action もあるが、全てこの構造になっている。  
 この為、GUI 上で

action を選択    どうか  
 source を選択    何から  
 result を選択    何を作り出すか

を操作する事によって、topoSetDict を作り出すように設定している。(直感的に操作できる)

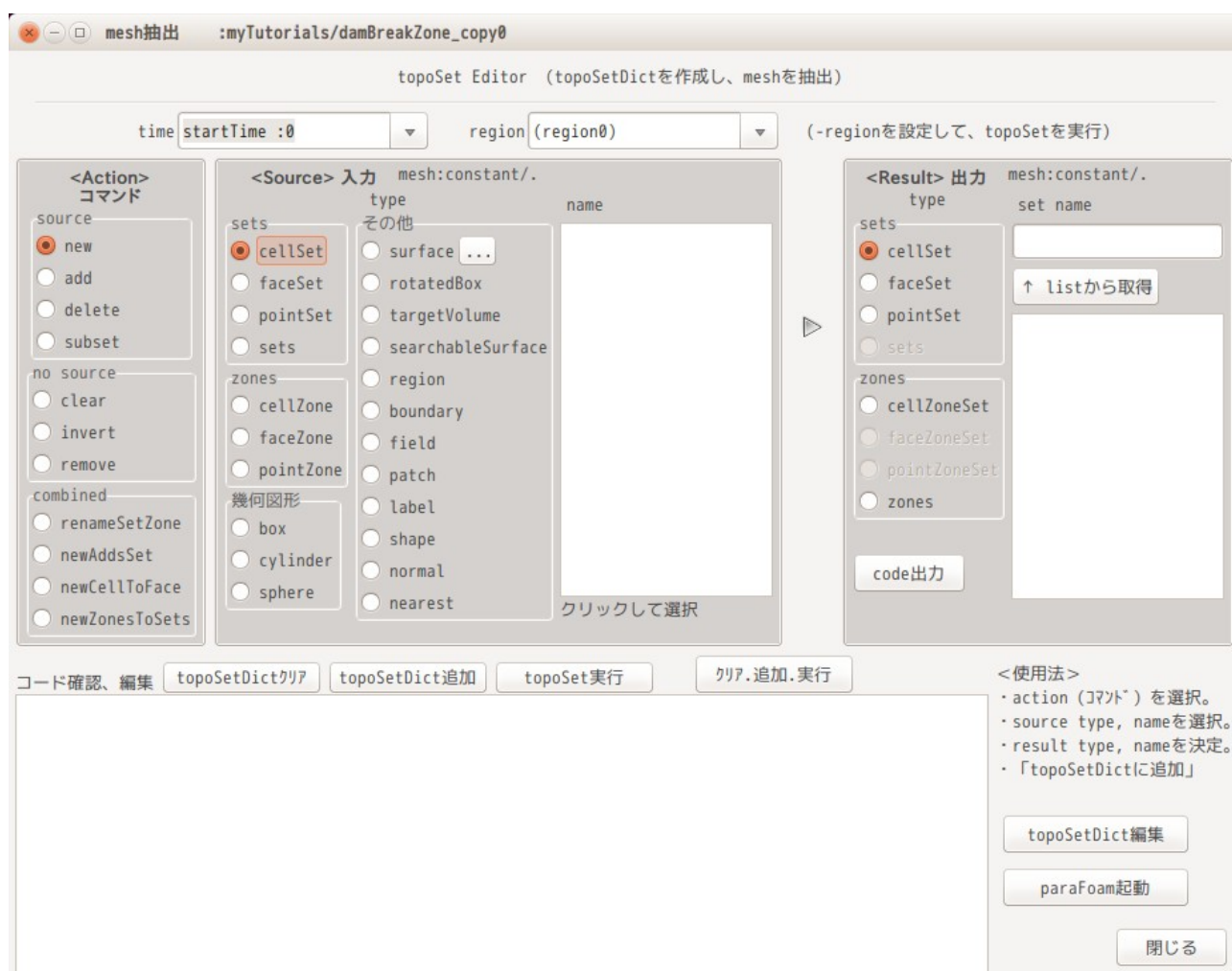
また、0F-5.0 系と 0F-v1806 系は、当初 topoSetDict の書式が同じだったが、0F-v1812 から書式が変わっている。この為、TreeFoam ではこれらに対応する為、0F-5.0 系は、\$TreeFoamPath/data/0F-5.0/system 内に topoSetDict を、0F-v1812 系は、\$TreeFoamPath/data/0F-v1812/system 内に topoSetSourceDict を保存しており、TreeFoam 側は、0F のバージョンからそのバージョンに近い topoSetDict の内容を読み取り、action の内容を取得している。

## 8-2-2. topoSetEditor の画面

TreeFoam 上の  ボタンをクリックして topoSetEditor を起動すると、以下の画面が現れる。この画面は、「Action」「Source」「Result」の 3 ブロックに分かれており、各々のブロックから項目を選択するだけで、topoSetDict の内容(下部のテキストボックス中に表示される)が作成できる。

テキストボックス中に表示された内容は、ボタン操作で topoSetDict にコピーして、topoSet を実行する事ができ、メッシュ操作が楽に行える。

また、Result の Type に「sets」と「zones」があるが、これは、複数の source 名を選択して、繰り返しの処理が行える様に準備している。詳細は、8-2-6 項を参照。



## 8-2-3. topoSet コマンドの内容

topoSet コマンドの構成を OpenFOAM-2.4.0 で調べた結果、下表の構成となっている。

action 「clear」「invert」「remove」は、source 不要で、直接 result を指定する。  
action 「new」「add」「delete」「subSet」については、source とそれに対する result の組み合わせを指

定する。

topoSetEditor 画面上で、action と source、result を選択すると、topoSetEditor は、それに対応するコマンド内容を検索してテキストボックス中に表示する。存在しない組み合わせの場合、source の内容が表示されない。(result の内容は表示される。)

#### 8-2-4. topoSet コマンドの抽出について

topoSetEditor は、前項の様に与えられた「Action」「Source」「Result」の情報から、該当する topoSet コマンドを抽出してくる必要がある。

この方法は、手本となる topoSetDict を読み込み、ここから該当する部分を抜き出している。手本となる topoSetDict は、\$TreeFoamPath/data/OFDict フォルダ内に各 version 毎に保存されている。この為、OpenFOAM のバージョンによってコマンドの書式が変わっていても、そのバージョンに応じたコマンド内容が抽出できることになる。(実際にバージョンによって書式が変わっているコマンドもある。)

以下は、そのファイルの一部であるが、コマンドを抜き出す部分は、「//」でコメントアウトされた部分を使って抜き出している。

```
/*----- C++ -----*/
=====
\\      F ield
\\      O peration
\\      A nd
\\      M anipulation
OpenFOAM: The Open Source CFD Toolbox
Version:  2.4.0
Web:      www.OpenFOAM.org
/*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    object       topoSetDict;
}
// *****

// List of actions. Each action is a dictionary with e.g.
// // name of set
// name      c0;
//
// // type: pointSet/faceSet/cellSet/faceZoneSet/cellZoneSet
// type      cellSet;
//
// // action to perform on set. Two types:
// // - require no source : clear/invert/remove
// //   clear : clears set or zone
// //   invert : select all currently non-selected elements
// //   remove : removes set or zone
// // - require source : new/add/delete/subset
// //   new : create new set or zone from source
// //   add : add source to contents
// //   delete : deletes source from contents
// //   subset : keeps elements both in contents and source
// action    new;
//
// The source entry varies according to the type of set:
//
// cellSet
// ~~~~~
//
// // Select by explicitly providing cell labels
// source    labelToCell;
// sourceInfo
// {
//     value (12 13 56); // labels of cells
// }
//
// // Copy elements from cellSet
```



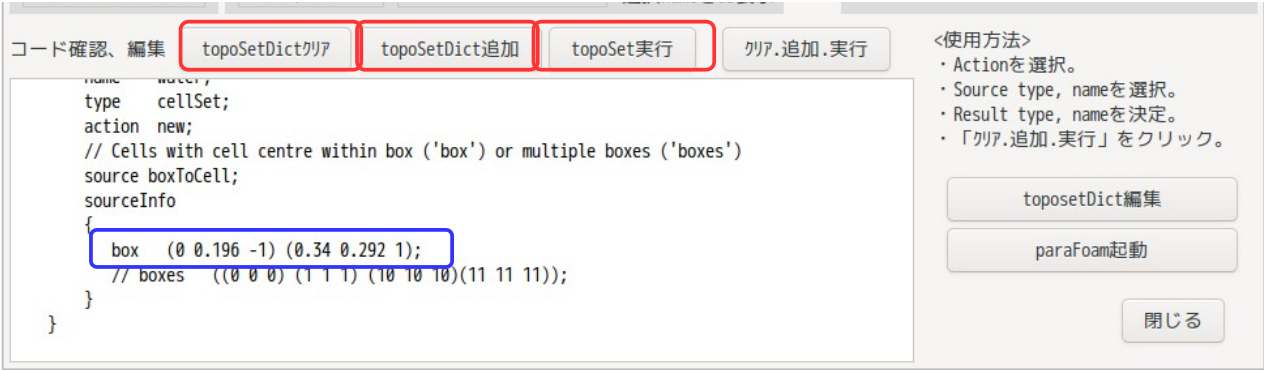
以下省略

実施例として「box で cellSet を抽出」、「cellZone から cellSet を抽出する」事を行ってみる。

tutorials の damBreak を使って、box 領域の cellSet を抽出してみる。

この後、TreeFoam 上の  ボタンをクリックして topoSetEditor を起動する。  
起動後、画面上で以下を選択する。

ここまでの操作で topoSetDict の内容は、殆ど出来上がっているが、抽出する為の box の座標が default の値になっているので、この値を修正する。修正は、テキストボックス内に表示されている座標を直接編集する。

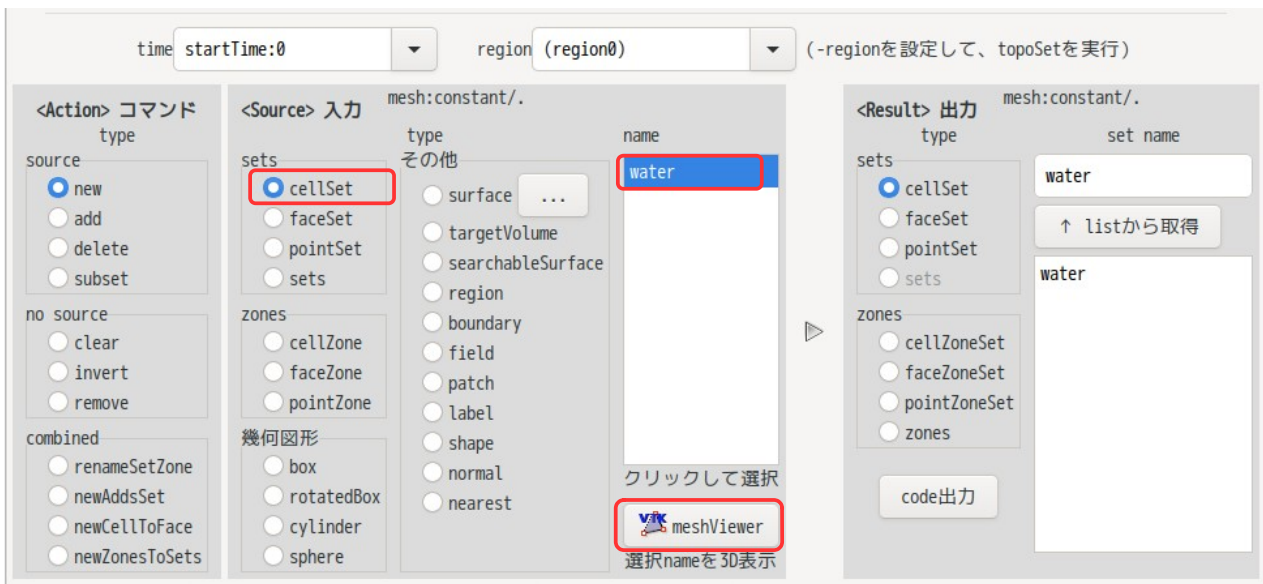


```

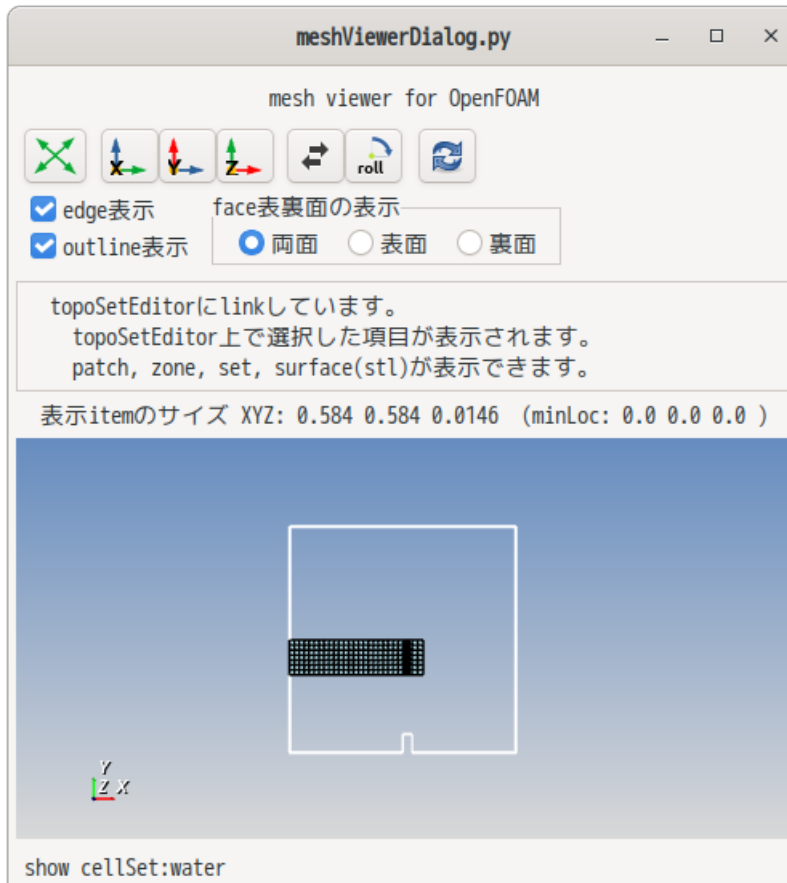
actions
(
    // new To cellSet
    {
        name    water;
        type    cellSet;
        action  new;
        // Cells with cell centre within box ('box') or multiple boxes ('boxes')
        source  boxToCell;
        sourceInfo
        {
            box    (0 0.196 -1) (0.34 0.292 1);
            //boxes ((0 0 0) (1 1 1) (10 10 10)(11 11 11));
        }
    }
);
// *****

```

抽出された、cellSet「water」の内容は、paraFoamでも確認できるが、TopoSetEditor 上の「meshViewer」ボタンをクリックして、viewer を起動して確認した方が、素早く確認できる。その方法は、下図の様に、作成した cellSet「water」が見えるようにラジオボタン「cellSet」を選択し、cellSet「water」を選択して、「meshViewer」ボタンをクリックする。



meshViewer が選択した cellset「water」を表示した状態で起動する。(下図参照)



meshViewer は、paraFoam よりも起動は早い。また、選択した item (patch, zones, sets, stl) を表示した状態で起動する。また、meshViewer が起動した後でも、topoSet 上で選択 item を変更してもリアルタイムで表示が変更される。この為、使いやすい。

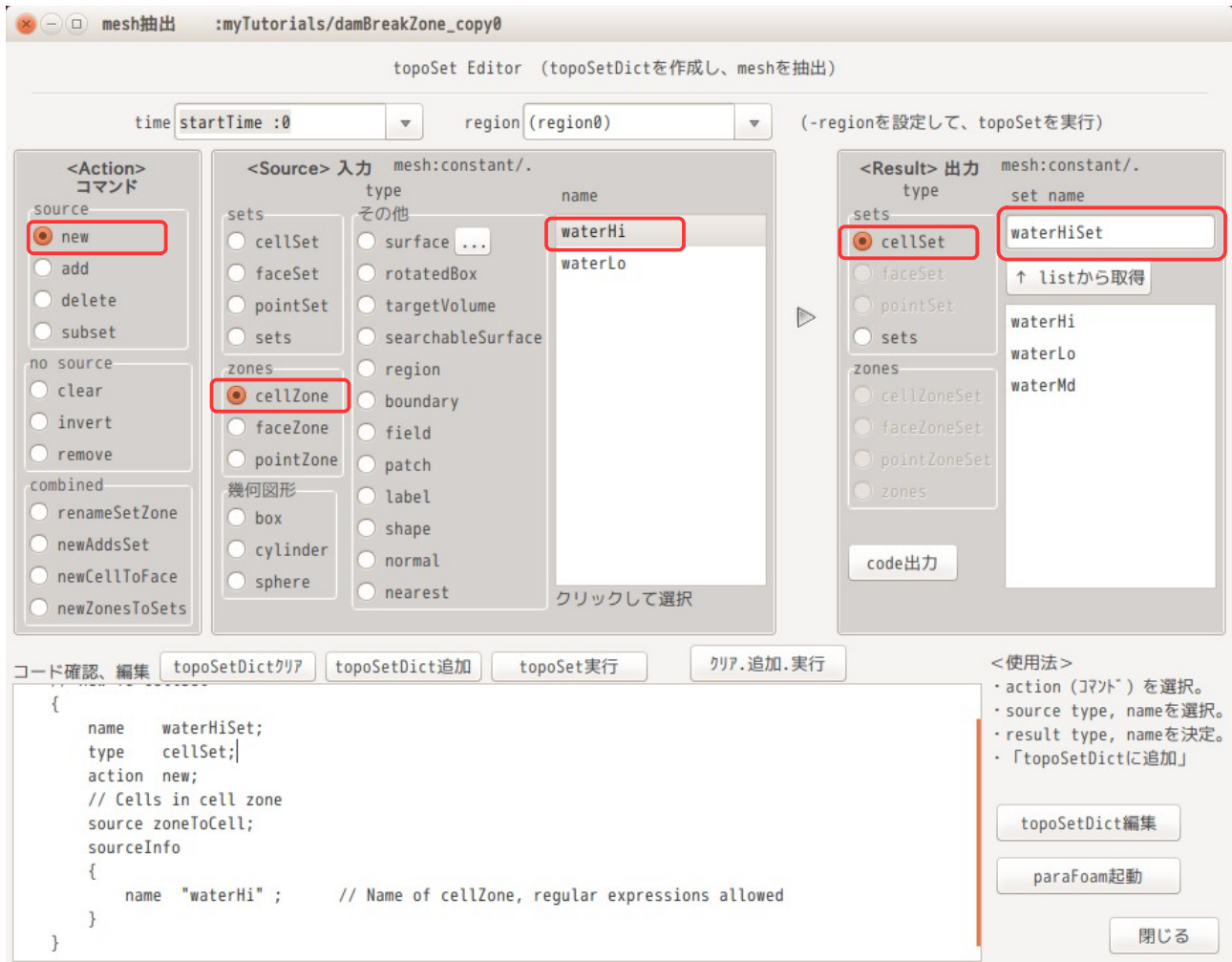
#### 8-2-5-2. cellZone を cellSet として抽出

メッシュ作成時に cellZone を作った場合等を想定して、その cellZone から cellSet を作り出してみる。

まず、case を作成する必要がある為、7-2 項で作成した case 「faceCellZoneMesh」をコピーして、メッシュ操作 (cellZone から cellSet を抽出) してみる。コピーした case を解析 case に設定 (☑マーク付き) した上で、TreeFoam 上の  ボタンをクリックして topoSetEditor を起動する。

topoSet の処理は、cellZone 「waterHi」から cellSet 「waterHiSet」を作成してみる。この処理を行う為に、topoSet の画面上で以下を選択し、cellSet の名称を入力すると、テキストボックス中に、その処理を行う為の topoSetDict の内容が表示される。

action	new
source	cellZone
	name 「waterHi」を選択
result	cellSet
	set neme 「waterHiSet」を入力



今回の場合、表示された topoSetDict の内容は修正する必要がなく、このまま topoSetDict として使えるので、1)項と同様に「topoSetDict クリア」「topoSetDict 追加」「topoSet 実行」ボタンを順にクリックする事で、cellSet「waterHiSet」が作成できる。

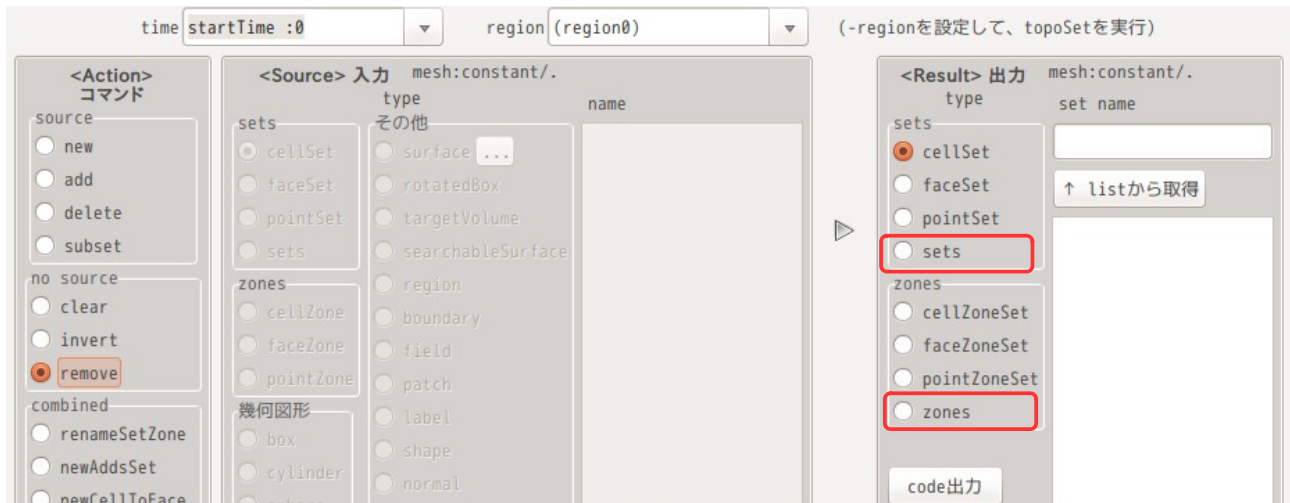
今回の操作は、cellZone から cellSet を作り出す操作だが、source と result を入れ替えて操作すると、cellSet から cellZone を作り出すことができる。

また、ボタンをクリックする時、topoSetDict をクリアせずに、「topoSetDict 追加」「topoSet 実行」ボタンのみをクリックした場合は、今回の topoSetDict の内容を、既にある system/topoSetDict に追加して topoSet を実行する。この為、topoSetEditor 上で、単一処理の topoSetDict を追加しながら、最終的に複数の処理を行う topoSetDict を作成する事ができる。

## 8-2-6. 繰り返しの Action について

複数の cellZone (又は faceZone, pointZone) を一括して同名の cellSet (又は faceSet, pointSet) を作成したり、逆に複数の cellSet を一括して同名の cellZone を作り出せる様に、Result type に「sets」と「zones」を準備している。下図参照。





Result type で「sets」または「zones」を選択した場合は、source 名を複数選択でき、選択した Action を繰り返して処理する事ができる。

この時の Action は、

new	
delete	
clear	no source
invert	no source
remove	no source

が使える。

上記 Action の内、clear, invert, remove は、source を必要としない Action の為、複数の source 名が選択できない事になるが、この場合は、複数の Result 名を選択して、繰り返し処理を行う事になる。これらの繰り返しの処理は、対象がたくさんある場合、楽に処理でき、非常に便利になる。

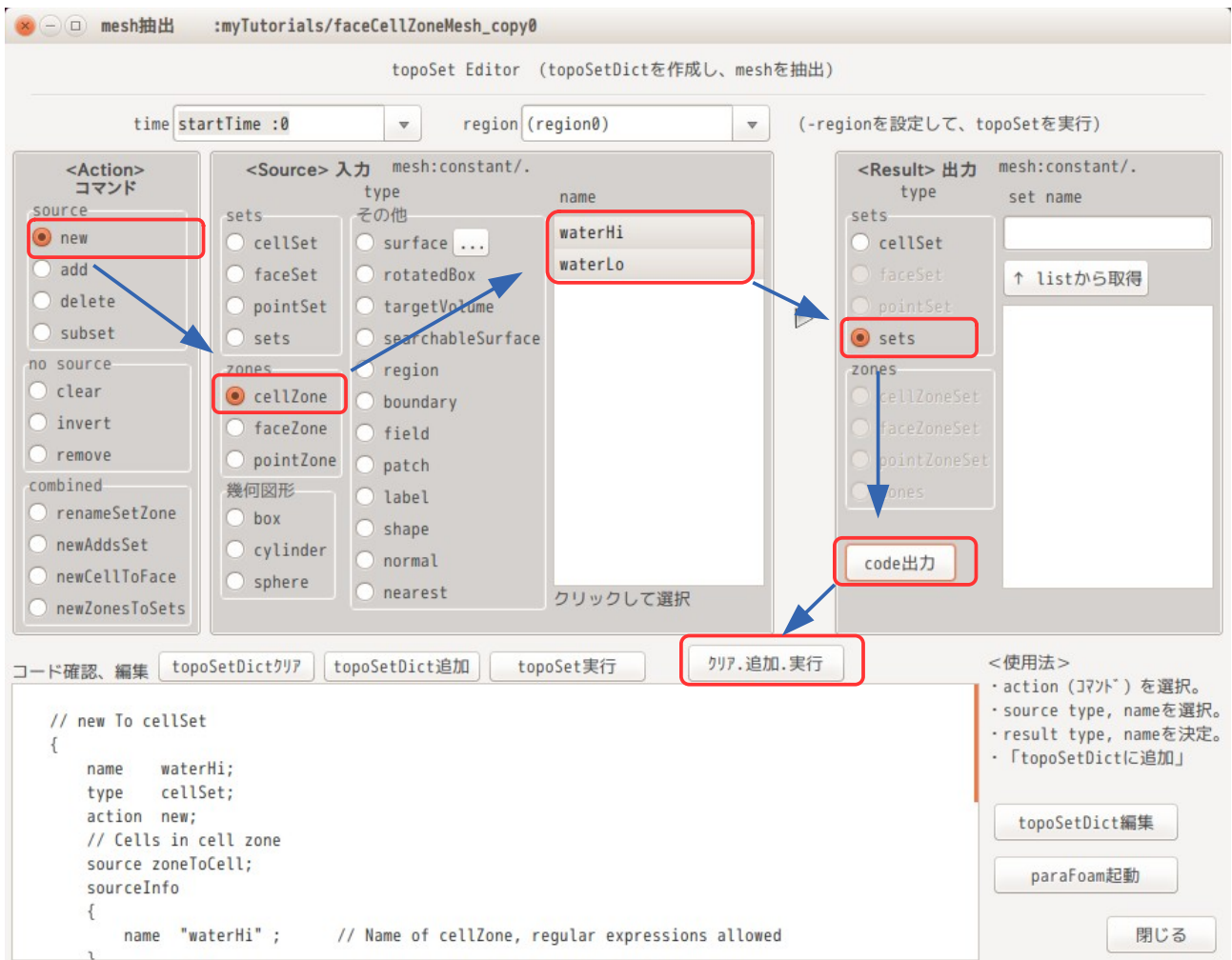
次項以降に new, remove Action についての例を示す。

#### 8-2-6-1. new Action の繰り返し処理の例

一例として、7-2-7 項と同じ処理を行ってみる。

この処理は、faceZone 「waterHi」 「waterLo」 から、同名の cellSet 「waterHi」 「waterLo」 を作り出す処理になる。new Action を 2 回繰り返す処理を行う事になる。

具体的には、以下の様に選択する。(Source 名を複数選択し、Result type で sets を選択する。)



Result typeとして「sets」を選択しており、これだけではResult typeがcellSet, faceSet, pointSetなのか確定しない事になるが、Source typeでcellZoneを選択している為、Result typeは、「cellSet」に設定される。この様に、Result typeは、Source typeで決定される事になる。

以上の操作により、以下のtopoSetDictができて上がる。

```
// * * * * *
actions
(
    // new To cellSet
    {
        name    waterHi;
        type    cellSet;
        action  new;
        // Cells in cell zone
        source  zoneToCell;
        sourceInfo
        {
            name "waterHi" ;    // Name of cellZone, regular expressions allowed
        }
    }
    // new To cellSet
    {
        name    waterLo;
        type    cellSet;
        action  new;
        // Cells in cell zone
        source  zoneToCell;
        sourceInfo
    }
}
```

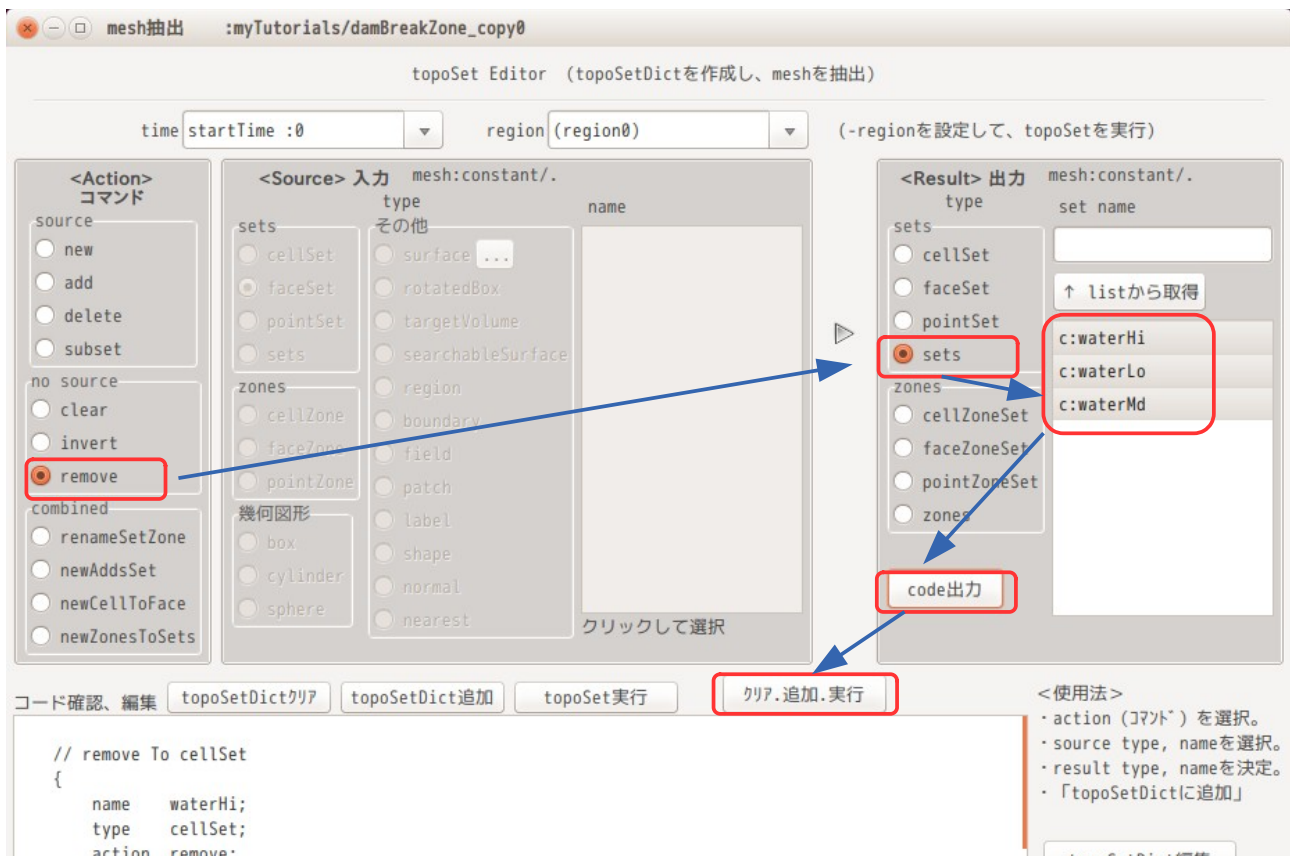
```

    {
        name "waterLo" ;      // Name of cellZone, regular expressions allowed
    }
};
// *****

```

#### 8-2-6-2. remove Action の繰り返し処理の例

cellZone 「waterHi」「waterMd」「waterLo」を全て削除してみる。  
remove Action は、source を必要としない Action の為、result 名を複数選択することになる。  
具体的には、以下の様に選択する。



以上の操作により、以下の topoSetDict ができあがる。

```

// ***** //
actions
(
    // remove To cellSet
    {
        name    waterHi;
        type    cellSet;
        action   remove;
    }
    // remove To cellSet
    {
        name    waterLo;
        type    cellSet;
        action   remove;
    }
    // remove To cellSet
    {
        name    waterMd;
        type    cellSet;
        action   remove;
    }
)

```

```

        name    waterMd;
        type    cellSet;
        action   remove;
    }
};
// *****

```

### 8-2-7. 組み合わせ (combined) Action について

topoSet の action は、単一処理しかできない為、例えば「cellZone の名称を変更する」場合は、topoSet の action を複数組み合わせることで実現することになる。

この為、よく使用する処理については、これらを組み合わせるコマンドを作っているため、これを使用する。下図の   部分参照。

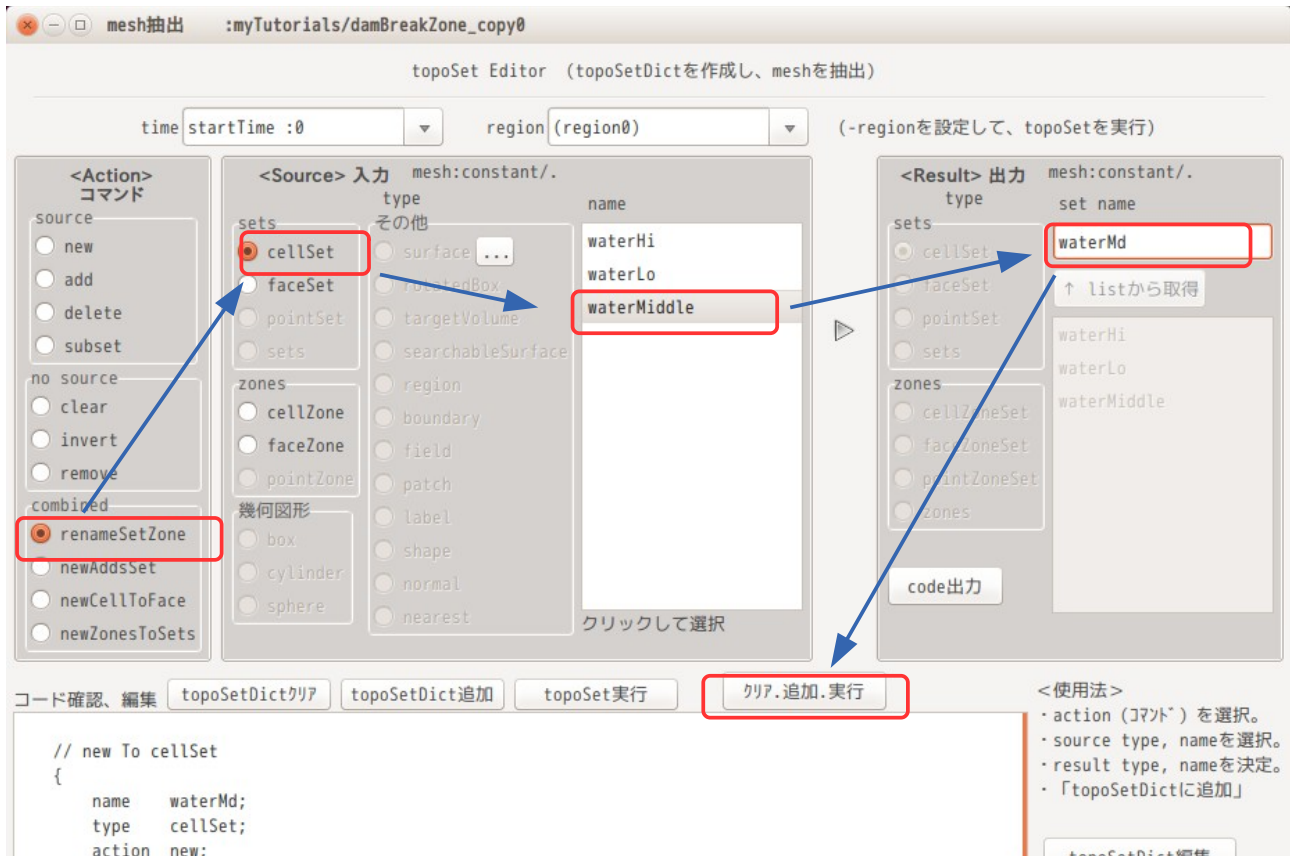


以下に、これら組み合わせ Action の具体的使用例を示している。  
尚、newAddsSet と newZonesToSets Action については、前項の繰り返し処理で同様な事が実現できる為、これらの使用例を省略している。この為、使用例としては、renameSetZone と newCellToFace を載せている。

#### 1) renameSetZone

cellSet、faceSet、cellZone、faceZone の名称を変更したい時に使用する。

以下の例は、cellSet「waterMiddle」を cellSet「waterMd」に rename する操作を示している。（以下の様にクリックするだけで、この操作を行う topoSetDict ができあがる。）



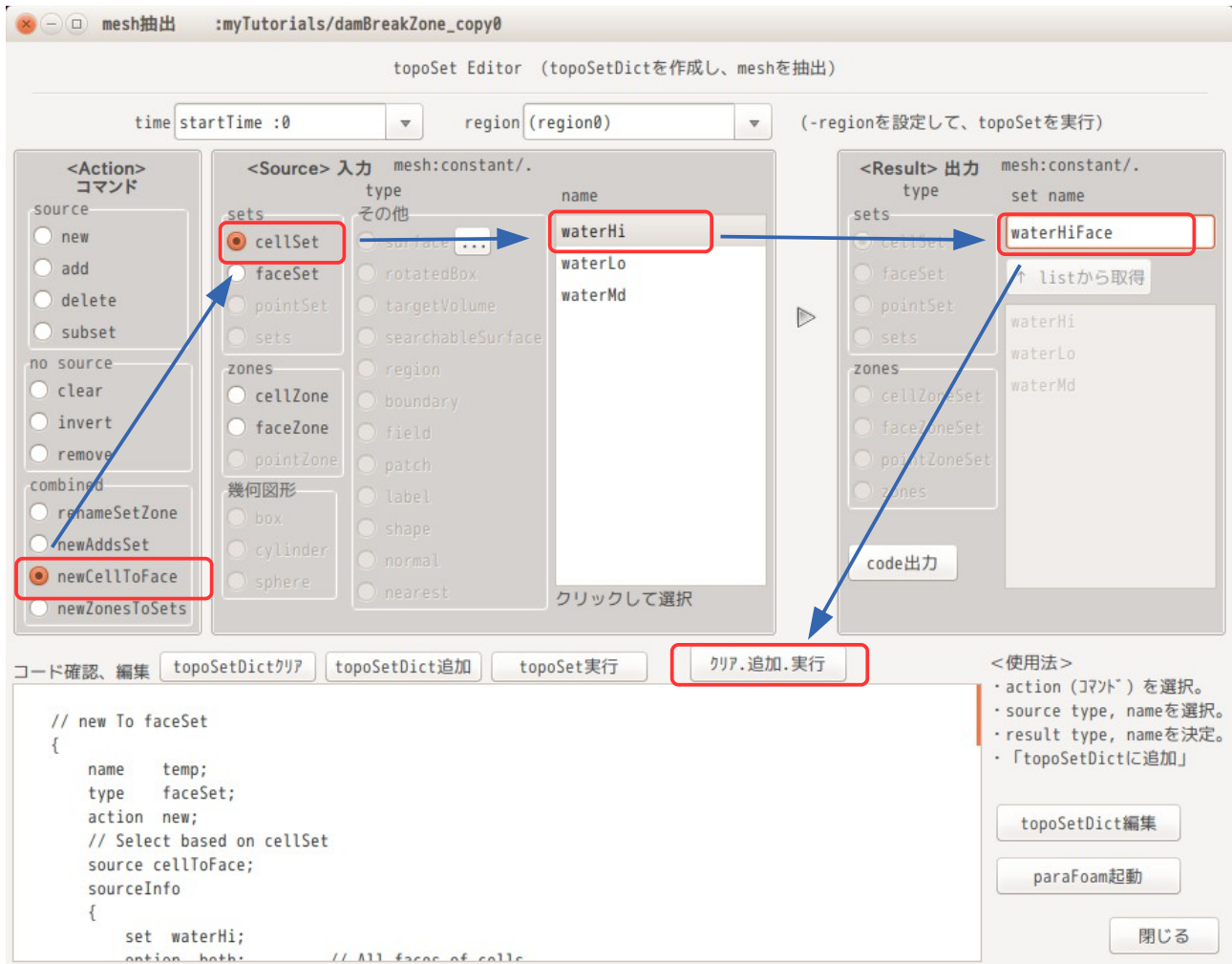
以上の操作により、以下の topoSetDict ができあがり、これを実行する事で、「waterMiddle」が「waterMd」に rename できる。

```
// * * * * * //
actions
(
    // new To cellSet
    {
        name    waterMd;
        type    cellSet;
        action  new;
        // Copy elements from cellSet
        source cellToCell;
        sourceInfo
        {
            set waterMiddle;
        }
    }
    // remove To cellSet
    {
        name    waterMiddle;
        type    cellSet;
        action  remove;
    }
);
// * * * * * //
```

## 2) newCellToFace

選択した cellZone 又は cellSet から、外表面の face を抜き出し、faceZone 又は faceSet を作り出す。以下の例は、cellSet 「waterHi」の外表面を faceSet 「waterHiFace」として取得する例になる。





以上の操作で、以下の topoSetDict ができあがり、これを実行して外周面の faceSet 「waterHiFace」を取得することができる。

```
// * * * * * //
actions
(
    // new To faceSet
    {
        name    temp;
        type    faceSet;
        action  new;
        // Select based on cellSet
        source  cellToFace;
        sourceInfo
        {
            set  waterHi;
            option both;           // All faces of cells
            //option both;         // Only faces whose owner&neighbour are in cellSet
        }
    }
    // new To faceSet
    {
        name    waterHiFace;
        type    faceSet;
        action  new;
        // Select based on cellSet
        source  cellToFace;
        sourceInfo
        {
            set  waterHi;
```

```

        option all;          // All faces of cells
        //option both;       // Only faces whose owner&neighbour are in cellSet
    }
}
// delete To faceSet
{
    name    waterHiFace;
    type    faceSet;
    action  delete;
    // Copy elements from faceSet
    source  faceToFace;
    sourceInfo
    {
        set  temp;
    }
}
// remove To faceSet
{
    name    temp;
    type    faceSet;
    action  remove;
}
};
// *****

```

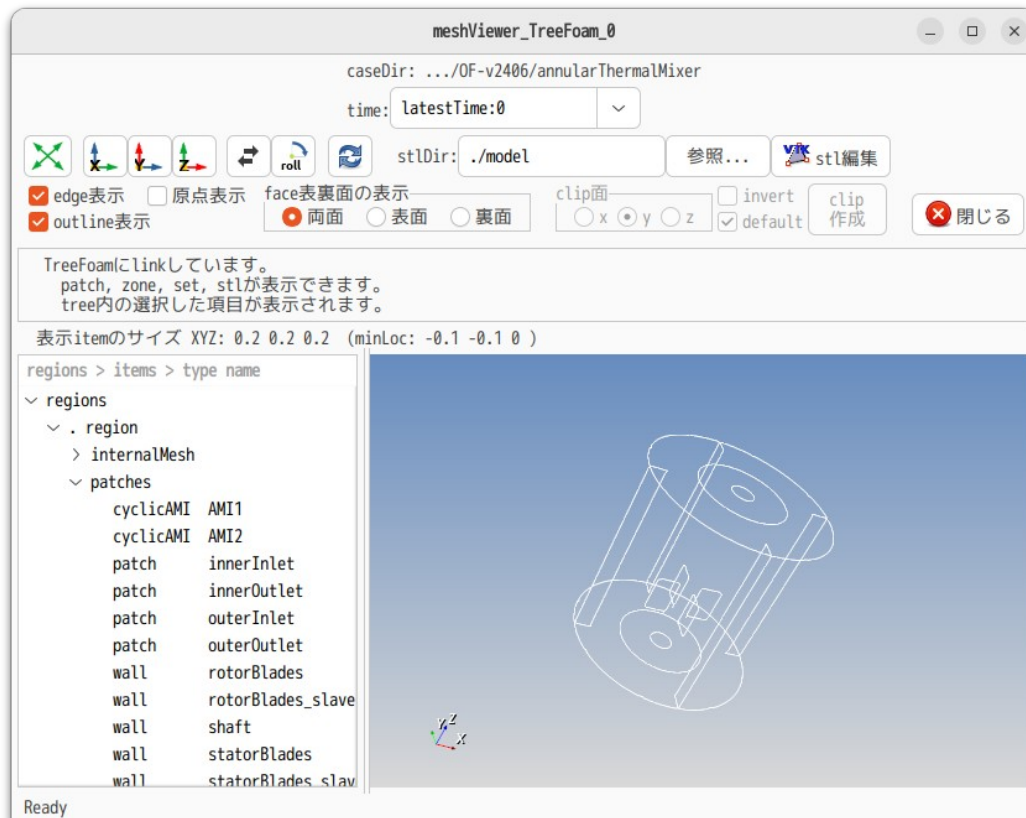
### 8-3. meshViewer

case 内のメッシュ形状が速やかに確認できる様にする為に作成した。起動時間（メッシュ表示までの時間）は、paraView よりも早い時間で起動、表示する事ができる。  
 これは、メッシュ情報しか読み込まない為。（初期値や計算結果が入る field データを読み込まない。）  
 また、meshViewer の表示は、internalField、patch、zones、sets、stl に限定している。  
 これにより、起動時間が早くなっている。

meshViewer は、解析 case 内のメッシュを表示する。また、meshViewer は TreeFoam 側と link している為、meshViewer が起動した状態で、TreeFoam の解析 case を変更すると、meshViewer がそれを検知して、変更した解析 case のメッシュを読み込んで、再表示する。  
 この為、多くの case の中から目的の case を探す時には、メッシュ形状が速やかに確認できるので、威力を発揮する。

また、meshViewer から、stlViewer が起動できるので、stl ファイルの拡大縮小、移動、回転などの編集が行える。この為、stl ファイルを使ったメッシュを作成する時には、便利。

下図は、meshViewer が起動した状態になる。起動時では、モデルの outline（メッシュから抽出したモデルの edge）を表示している。（tutorials の rhoPimpleFoam/RAS/annularThermalMixer のメッシュを表示）

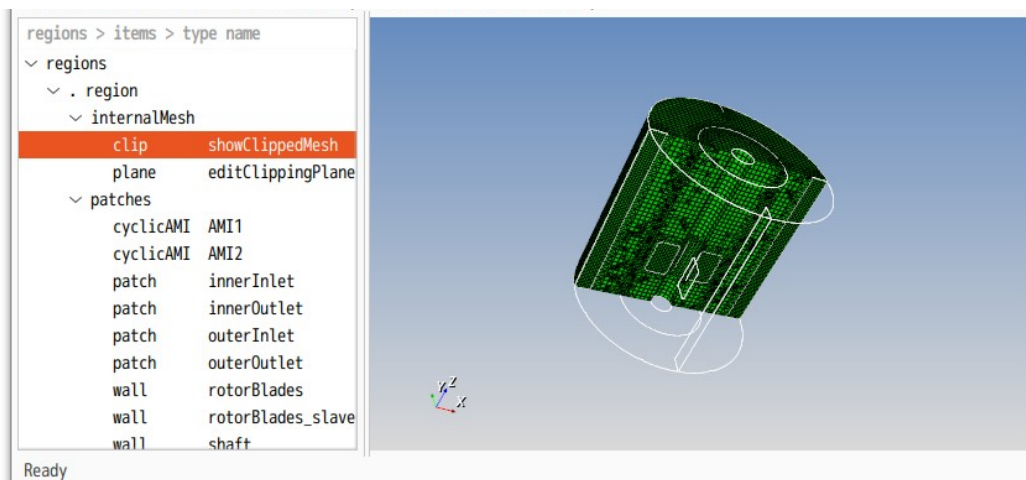


### 8-3-1. internalMesh の表示

メッシュにレイヤを追加した時などは、メッシュ内部の確認が必要になる事がある。メッシュ内部の確認は、断面カットで確認できる。meshViewer でも任意の位置で断面カットができる。

#### 8-3-1-1. デフォルト面でカット

デフォルト面は、メッシュの中心座標を通る Y 方向のベクトルの垂直面に設定されている。この面で断面カットする場合は、下図の様に、Tree 内の「internalMesh」>「clip」を選択すると、デフォルト面で断面カットしたメッシュが確認できる。

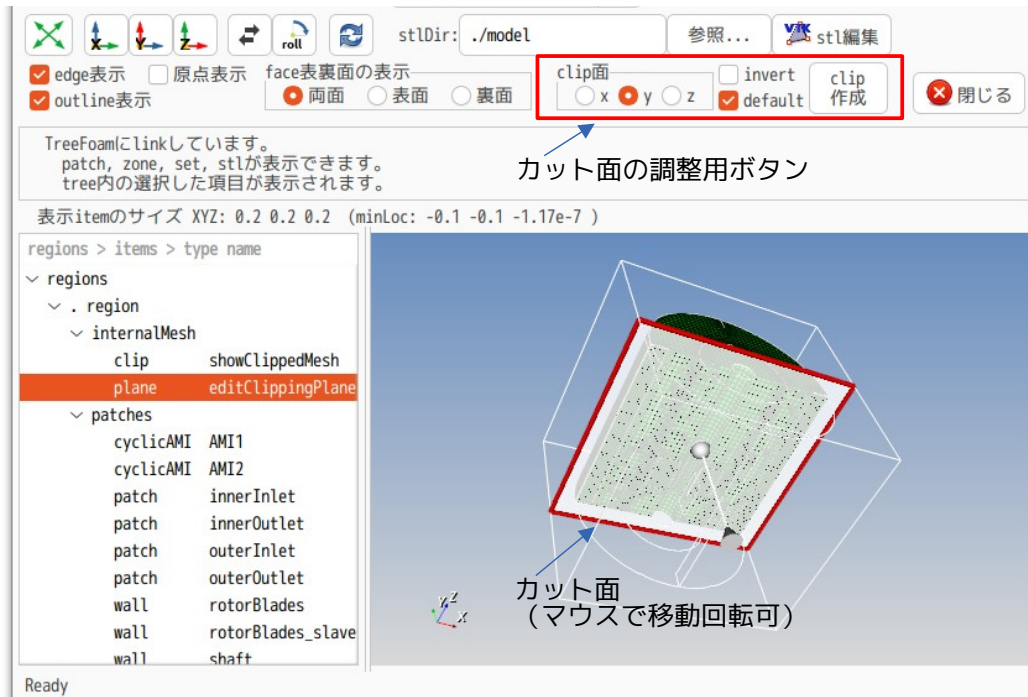


### 8-3-1-2. 任意面でカット

任意の面で断面カットするには、Tree の「internalMesh」>「plane」を選択する。ここを選択すると、赤枠で囲まれた面が表示される。この面がカット面になる。  
また、カット面を調整する各ボタンが表示され、これらのボタンが使用できる状態になる。

赤枠で囲まれたカット面は、マウスでドラッグして移動、回転が可能になっている。  
各ボタンは、カット面の方向 (xyz 方向)、面の反転 (invert) などでカット面を調整する事ができる。

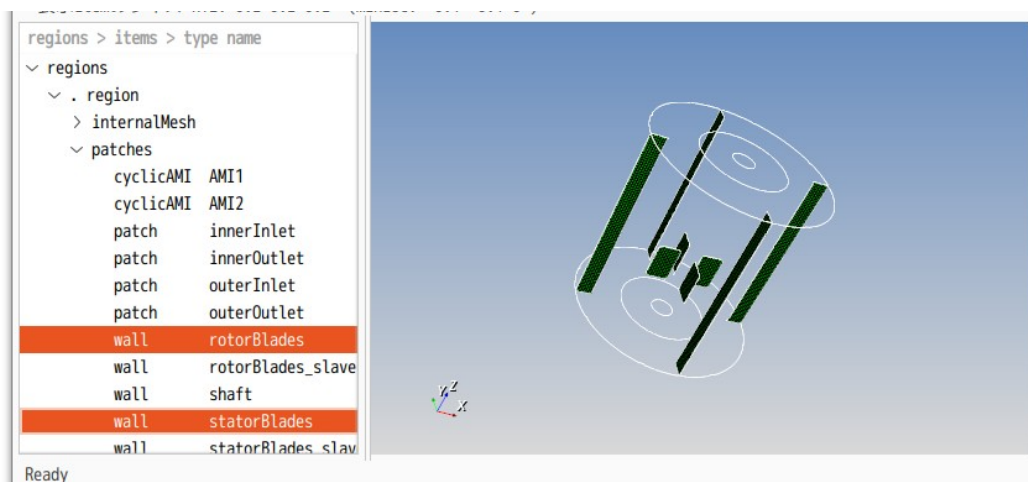
カット面が確定した後は、「clip 作成」ボタンで確定し、メッシュのカットが完了する。  
カット後は、Tree の「internalMesh」>「clip」を選択する事で、余分な赤枠のカット面を消す事ができる。



### 8-3-2. patch の表示

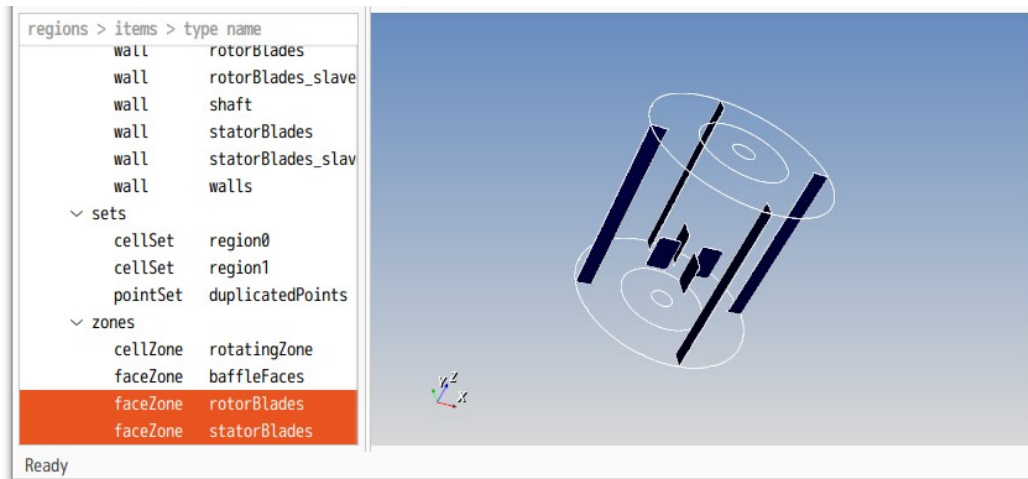
meshViewer は、基本的に画面左の Tree 内の項目を選択した時、その選択された項目 (patch) を outline 上に追加して表示する。

下図は、選択した blade の patch を表示している。



### 8-3-3. sets、zones の表示

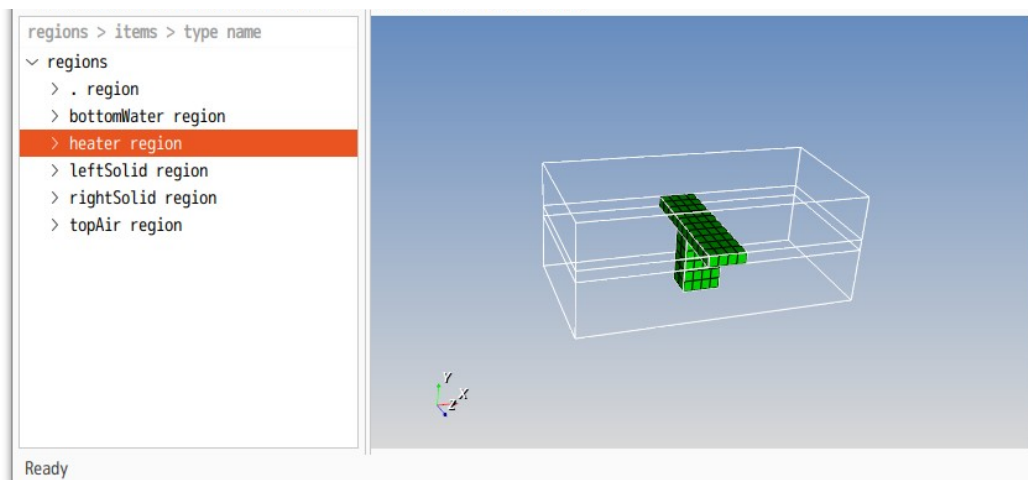
meshViewer の Tree 内に sets、zones の項目がある場合（メッシュ内に sets、zones が存在する）、それらを開くと、その内容が確認できる。前項と同様に、選択した項目が表示される。下図は、faceZone「rotorBlades」、faceZone「statorBlades」を選択して表示している。



### 8-3-4. multiRegion の表示

multiRegion の case の場合は、meshViewer が起動した段階で、全 region を読み込み、全 region の outline を表示する。

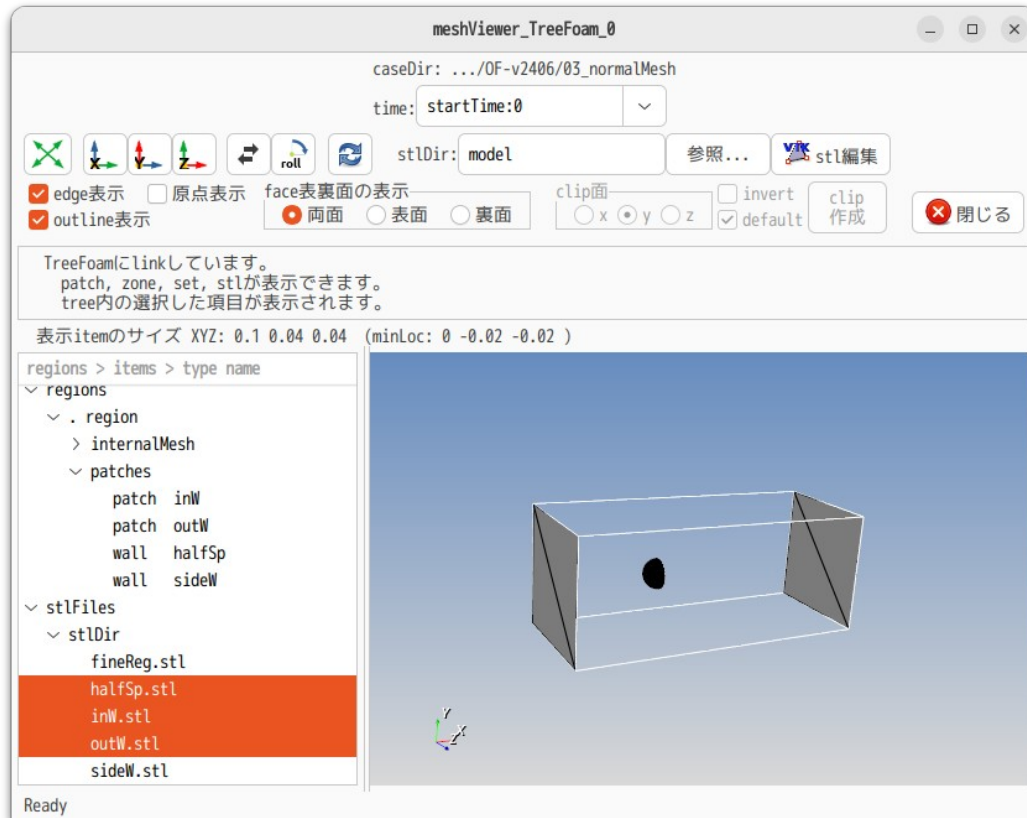
下図は、chtMultiRegionHeater の case を meshViewer で開いた状態で、heater region のメッシュを表示している状態になる。また、この case の region としては、「bottomWater」、「heater」、「leftSolid」、「rightSolid」、「topAir」がある事も容易に確認できる。これら region 内の patches、sets、zones のメッシュ形状も Tree を選択する事で表示できる。



### 8-3-5. stl の表示

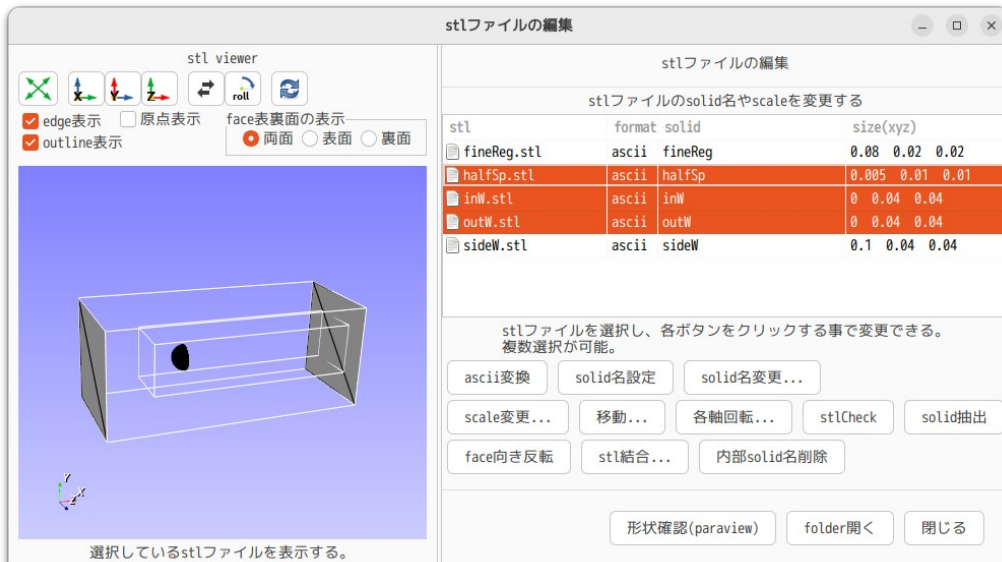
stl ファイルについては、stl ファイルの場所 (stlDir) を指定することで、stl ファイルの形状を meshViewer で表示させることができる。（下図参照。）





また、「stl 編集」ボタンで stlViewer を起動する事ができるので、stl ファイルの拡張、移動、回転を行うことができる。下図が stlViewer の起動画面になる。

stlViewer の詳細は、「9-1-1. stl ファイルの編集」を参照。



## 8-4. FrontISTR との連成解析

OpenFOAM の流体解析と FrontISTR の構造解析・熱解析を連成させて、流体-構造連成や流体固体の熱連成解析が実現できる。

流体構造連成解析の場合、圧力  $p$  field と変位  $\text{pointDisplacement}$  field が扱える solver (pimpleFoam, interFoam 等) に限定される。

流体固体の熱連成解析の場合、温度  $T$  field が扱える solver であれば、基本的に適用できる。熱連成解析に加えて、固体の熱ひずみを扱う場合は、変位  $\text{pointDisplacement}$  field が必要になる。

流体構造連成、熱連成、固体の熱ひずみを扱う解析の場合は、圧力  $p$ , 変位  $\text{pointDisplacement}$ , 温度  $T$  field が扱える solver が必要で、この場合、buoyantPimpleFoam, rhoPimpleFoam 等の solver を使って解析することになる。

流体と構造側のメッシュは、各々独立している為、これらの間で物理量を交換する時、mapping の操作が必要になる。mapping は、平面上の 3 点の物理量から、平面内の任意点の物理量を求めている為、solid モデルに限定される。

シェルやビームの場合は、edge や point に境界条件を設定するが、edge や point には物理量が mapping できない。この為、この連成解析では、シェルやビームは、扱えない。

OpenFOAM は、出力時間の有効桁数が controlDict 内の writePrecision で設定できる。この有効桁数を大きくすると、連成解析の相手側 (FrontISTR) に送信する値の有効桁数も大きくなる。特に時間の有効桁数が大きくなる事により、時間増分 ( $\Delta t$ ) の精度も向上する。

### 8-4-1. 連成計算方法

#### 8-4-1-1. 流体構造の連成計算方法

連成計算は、流体側で計算した圧力を構造側に mapping して、変位を計算。その変位の計算結果を流体側に mapping して mesh の座標を更新する。これを繰り返して連成計算を行っていく。

連成計算する方法として、OpenFOAM と FrontISTR を同時に計算を進めて行く方法と、これらを交互に計算を進めていく方法を準備している。

この流体-構造連成解析を実現させる為に、controlDict 内の functions 内に sourceCode を include している。その処理内容は、以下になる。

#### <同時計算 (高速)>

OpenFOAM 側

FrontISTR 側

- ・ patch の圧力を出力
- ・ 前回の patch 変位を読み、patch にセット
- ・ pythonScript を裏で起動
- ・  $\Delta t$  を更新、流体計算再開

- ・ patch の圧力を SGRP に mapping
- ・ FrontISTR を起動して変位計算
- ・ 変位を patch 座標に mapping
- ・ 次回の patch 変位として出力

} PythonScript  
が処理する

#### <交互計算 (安定)>

OpenFOAM 側

FrontISTR 側

- ・ patch の圧力を出力
- ・ pythonScript を起動
- ・ patch 変位を読み、patch にセット
- ・  $\Delta t$  を更新、流体計算再開

- ・ patch の圧力を SGRP に mapping
- ・ FrontISTR を起動して変位計算
- ・ 変位を patch 座標に mapping
- ・ patch の変位として出力

} PythonScript  
が処理する

同時計算の場合は、OpenFOAM、FrontISTR のお互いの待ち時間が減少するので、計算は早くなるが、 $\Delta t$  は、一定値で計算が進んでいく。

交互計算の場合は、お互いの計算が終了するまで待つ為、計算時間が長くなるが、「adjustTimeStep yes;」として maxCo 制御で計算を勧めていく事ができるので、流速が変動するような場合、流速に応じた  $\Delta t$  で計算が進んでいき、安定して計算する事ができる。

OpenFOAM の controlDict 内の writePrecision を「writePrecision 8;」に修正すると、時間の有効桁数がデフォルトの 6 桁から 8 桁に変更される。これを変更すると、FrontISTR 側へ渡す時間の有効桁数も 8 桁に変

更されるので、OpenFOAM、FrontISTR 側も時間の精度が向上する。

また、計算結果が出力保存されている時間から連成計算が再開できる様にしている。

(startTime、latestTime で計算開始時間が設定できる。)

この機能があると、エラー停止した時、deltaT や writeInterval を変更して連成解析を再スタートさせて、エラーが発生する直前の結果を確認する事ができるので、デバッグには、便利。

#### 8-4-1-2. 流体固体の熱連成（固体の熱ひずみ含む）計算方法

流体固体の熱連成解析は、patch 温度から patch 面の熱流束を計算し、固体側の SGRP に mapping して温度を計算。その温度を流体側の patch 面に mapping して patch 温度を更新する。これを繰り返して計算を進めていく。

連成計算する方法として、流体構造連成と同様に、OpenFOAM と FrontISTR を同時に計算を進めて行く方法と、これらを交互に計算を進めていく方法を準備している。  
これらの処理内容は、以下になる。

##### <同時計算（高速）> OpenFOAM 側

- ・ patch の熱流束を出力
- ・ 前回の patch 温度を読み、patch にセット
- ・ pythonScript を裏で起動
- ・ deltaT を更新、流体計算再開

##### FrontISTR 側

- ・ patch の熱流束を SGRP に mapping
- ・ FrontISTR を起動して温度を計算
- ・ 温度を patch 面に mapping
- ・ 次回の patch 温度として出力

} PythonScript  
が処理する

##### <交互計算（安定）> OpenFOAM 側

- ・ patch の熱流束を出力
- ・ pythonScript を起動
- ・ patch 温度を読み、patch にセット
- ・ deltaT を更新、流体計算再開

##### FrontISTR 側

- ・ patch の熱流束を SGRP に mapping
- ・ FrontISTR を起動して温度を計算
- ・ 温度を patch 面に mapping
- ・ patch の温度として出力

} PythonScript  
が処理する

流体、固体間の熱移動は、上記の様に熱流束 heatFlux を介して行っている。

熱流束は、以下の cell 中心温度  $T_c$  と patch の face 中心温度  $T_f$  から以下の様に計算している。

$$q = -\kappa \frac{dT}{dx} = -\kappa \frac{T_c - T_f}{dx}$$

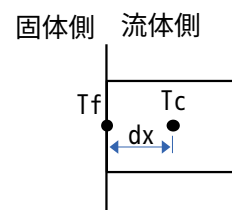
$T_c$ : cell 中心の温度 [K]

$T_f$ : face 中心の温度 [K]

$dT/dx$ : 温度勾配 [K/m]

$\kappa$ : 熱伝導率 [W/mK]

$q$ : 熱流束 [W/m<sup>2</sup>]



固体の熱ひずみに関しては、固体の温度分布を計算した後に熱ひずみを計算する。

#### 8-4-1-3. 連成計算における並列計算方法

OpenFOAM 側は、process 並列 (MPI 並列)、FrontISTR 側は thread 並列を前提としてきたが、TreeFoam ver 3.21 からは、FrontISTR 側も process 並列ができる様に修正した。

FrontISTR の起動は、以下の様に process 並列数と thread 並列数を指定して起動する方法になっているので、これをそのまま連成解析にも同様に設定した。

```
$ mpirun -np 4 fistr -t 2
```

#process 並列数:4, thread 並列数:2, 合計 8 コア使用

FrontISTR を process 並列計算した場合、計算結果が分散しているので、計算結果を OpenFOAM 側の patch に mapping する時、連成面のみ、計算結果を merge して mapping する様にしている。

process 並列計算する場合は、OpenFOAM、FrontISTR 共、予め並列計算用にメッシュ分割しておく。

OpenFOAM: TreeFoam 上で、メッシュ分割する。

FrontISTR: EasyISTR 上で、メッシュ分割する。

fsi, chtss: 同じメッシュファイルを使っている為、どちらかでメッシュ分割する。  
(メッシュファイル: FistrModel.msh)

cht: cht 専用のメッシュを使っており、これをメッシュ分割する。  
(メッシュファイル: FistrModel\_cht.msh)

連成計算は、OpenFOAM、FrontISTR とも、計算開始時にメッシュ分割数を確認して、分割数分の process を起動して、連成計算を開始する。FrontISTR 側は、使用する全 core 数を指定する様にしており、余ったコアは、thread 並列に回される。

例) FrontISTR 側の計算

全 8 コアで、並列計算用でメッシュを 4 分割している場合

process 数: 4

thread 数: 2

全 8 コアで、並列計算用でメッシュ分割していない場合

process 数: 1

thread 数: 8

これらの値の設定は、dialog 内で以下の様に設定する。

並列計算する為には、「並列計算する」をチェックする。この後、FrontISTR 側は、「全 core 数」を入力する。

The screenshot shows a configuration dialog with two main panels: 'OpenFOAMの設定' (OpenFOAM Settings) and 'FrontISTRの設定' (FrontISTR Settings). Both panels have a '並列計算する' (Parallel Calculation) checkbox checked and highlighted with a red box. In the FrontISTR panel, the '全core数' (All Cores) is set to 8, also highlighted with a red box. Other settings include field coefficients, scalar fields, point fields, patch names, and timing parameters like nStepsFsi and nStepsHeat.

## 8-4-2. 具体例

連成解析の tutorials を \$TreeFoamPath/frontIstr/tutorials フォルダ内に準備しているので参考になる。

### 8-4-2-1. 流体構造連成解析の具体例

preCICE の tutorial に設定されている「flap\_perp」を模したモデルを例として説明する。preCICE では、流体側は、2次元モデルで、構造側は3次元モデルとして計算しているが、今回の連成解析では、2次元モデルは、扱えないので、両方とも3次元モデルとして計算する。(3次元モデルとして mesh を作成している。)

この例は、\$TreeFoamPath/frontIstr/tutorials フォルダ内に OpenFOAM の各バージョン毎に「flap\_perp\_<OFver>-FrontISTR.zip」として保存している。該当するファイルを適当な場所で、展開する。(この例は、OpenFOAM-9 用で作成している。)

展開すると以下の folder が確認できる。



```

flap_perp_OpenFOAM-FrontISTR/
  Fluid/                                #TreeFoamの解析 folder に設定する
  0/
  0.org/
  constant/
  coupling_FrontISTR/                  #連成解析の folder
  data/                                #dataFolder
  python/                              #連成解析に必要なスクリプト
  couplingData                          #連成解析に必要な data を保存
  system/
  runFsi                                #連成計算開始スクリプト
  Solid/                                #FrontISTR の case

```

TreeFoam を起動して、Fluid フォルダにし点マークを付け、解析 folder に設定する。  
 この後、メニュー「計算」>「流体-構造連成解析」を選択して、連成解析用の dialog を起動する。

dialog 表示後、まず、赤枠内の連成する項目から fsi を選択する。さらに赤枠内の EasyISTR の実行 file



「easyistr」と FrontISTR の実行 file 「fistr1」の場所を設定する。  
残りの項目は、既に設定済のため、設定は不要。以下にその内容を説明する。

#### <連成解析の内容>

OpenFOAM と FrontISTR の計算方法 流体固体の計算を同時に計算させるか、交互に計算させるかを選択する。  
同時計算は、deltaT 一定で高速。交互計算は、maxCo 制御で deltaT が流速に応じて変動するので、計算が安定する。

#### <OpenFOAM 側の設定>

p field の係数 取得した圧力に乘じる係数  
OpenFOAM の pimpleFoam (非圧縮性流体の solver) の場合、圧力値に流体の密度を乗じる必要がある。扱う流体が空気の場合、密度は「1」なので、1.0 を入力。

p field の shift 量 取得した圧力の shift 量  
圧力が絶対圧で計算されている場合は、相対圧になる様に大気圧分を減じる必要があるが、今回は、相対圧で計算しているので、shift 量 0.0 を入力している。

scalarField 圧力を取得する field を指定する。

pointField 変位を設定する field を指定。

patch 圧力値を取得する patch 名を指定。

並列計算する 並列計算する場合は、チェックする。  
予め並列計算ができる様に、decomposePar で並列計算用にメッシュ分割しておく。  
OpenFOAM と FrontISTR の同時計算を選択している場合は、FrontISTR 側の thread 数を考慮して設定する。

#### <FrontISTR 側の設定>

easyistr EasyISTR の実行 file 「easyistr」の場所を指定する。  
連成解析のためには、EasyISTR は関係ないが、構造側の境界条件を設定する場合には、必要。

fistr1 FrontISTR の実行 file 「fistr1」の場所を指定する。

solidDir FrontISTR の解析 folder を相対 path で指定する。

SGRP 構造側の surfaceGroup 名を指定する。この SGRP 名と OpenFOAM 側の patch 名が対応し、OpenFOAM 側の圧力値が、この face に mapping される。

並列計算する FrontISTR を並列する場合は、チェックし、予めメッシュ分割しておき、全 core 数を入力する。  
OpenFOAM と FrontISTR の同時計算を選択している場合は、OpenFOAM 側の並列数を考慮して FrontISTR 側の process 数と thread 数 (全 core 数) を設定する。

#### <連成計算するタイミング>

nStepsFsi 構造と連成する step 数を指定。  
「1」を入力すると deltaT 時間毎に構造側と連成する。

#### <各 Job の待ち時間>

最大待ち時間 各 Job をキャンセルするまでの時間を指定。  
この時間内で、OpenFOAM、FrontISTR 側とも 1step の計算が終了する必要がある。  
終了しない場合は、Job がキャンセルされ、エラー停止する。

圧縮ファイルを展開した状態は、OpenFOAM 側、FrontISTR 側共に各々単独で動く状態になっている。(まだ、連成計算ができる状態にはなっていない。)

確認の為、今の状態で、OpenFOAM 側、FrontISTR 側がエラーなく単独で動くかどうか確認する。

OpenFOAM 側は、TreeFoam 上で、▶ アイコンをクリックして、計算開始させて、動作を確認する。  
FrontISTR 側は、連成計算 dialog 内の「EasyISTR 起動」ボタンで、EasyISTR を起動し、動解析の設定 (全 step 数、時間増分)、圧力 DLOAD の設定、restart の設定を確認し、計算を開始して動作を確認する。これらの設定内容は、連成計算の中で、内容が逐次修正されていく。(圧力 DLOAD の設定は、該当する SGRP 名の均一な圧力の設定で、作動を確認する。連成計算時は、この内容が分布圧力に修正されて計算が進んでいく。)

また、EasyISTR に関しては、以下の点に注意する。

- EasyISTR の起動は、「EasyISTR 起動」ボタンで起動する。このボタンで起動すると、solidDir を workFolder に設定し、その内容を読み込んだ状態で起動する為。
- EasyISTR 上で、内容を修正した場合は、必ずメニュー「ファイル」>「保存 (temp → dir)」を選択して、実 folder に修正した設定内容を反映させる。

- ・連成解析とセットで使用する場合は、EasyISTR-3.39 以降を使用する。

OpenFOAM、FrontISTR 単独で動く事を確認した後は、以下の手順で、連成計算を開始する。

計算開始は、以下の各ボタンをクリックして計算開始する。

「結果クリア」ボタン	OpenFOAM、FrontISTR 両方の計算結果をクリアする。
「設定取得・保存」ボタン	dialog の設定内容を couplingData ファイルに保存し、連成計算に必要なスクリプトをコピーする。FrontISTR 側の step 数、時間増分などを置き換える。
「連成計算開始」ボタン	連成計算を開始する。(runFsi を起動する)
「Fistr の log」ボタン	FrontISTR の出力 log を表示する。

任意モデルで連成計算させる場合も前記した様に、OpenFOAM の流体計算、FrontISTR 側の構造計算（非線形の陰解法の動解析）が単独でエラーなく、実行できる事を確認しておく。

この時の OpenFOAM 側の該当 patch の境界条件は、以下の設定にしておく。

```
pointDisplacement field :fixedValue (0 0 0);
```

また、constant/dynamicMeshDict の内容を以下の様に修正する。（以下の例は OF-9 の場合）

```
----- constant/dynamicMeshDict の内容 -----
dynamicFvMesh dynamicMotionSolverFvMesh;
motionSolverLibs ("libfvMotionSolvers.so");
motionSolver    displacementLaplacian;
diffusivity    quadratic inverseDistance 1(flap); //連成する patch 名に修正
-----
```

連成計算の基本は、OpenFOAM 側の startTime, latestTime, deltaT, writeInterval の設定で計算を開始する。FrontISTR 側の設定は、この設定に合わせて、リスタート step 数、時間増分が設定される。計算中は、writeInterval 毎に OpenFOAM と FrontISTR の計算結果を保存する。

また、OpenFOAM 側は、writeInterval の時間で結果を保存していくが、FrontISTR 側は、step 数で結果を保存していく。この為、結果を比較する時に、OpenFOAM の時間と FrontISTR の step 数の対応が判り難い。この対応は、FrontISTR のリスタートファイル名で確認できる。Solid フォルダ内にあるリスタートファイル名は、以下の様に時間と step 数を組み合わせているので、このファイル名から、時間と step 数の関係が確認できる。

以下の例では、時間:0.21, step 数:21 であり、0.21 s が 21 step になる。

リスタートファイル名 : FistrModel.restart\_0.21\_21.0

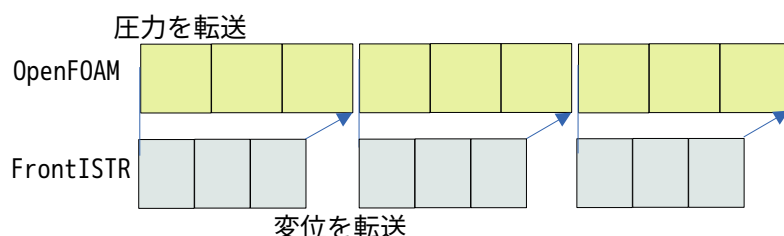
----- --  
時間 step 数

また、流体側の計算に負荷がかかる割に、構造側の変位が小さい様な場合には、流体構造を連成して計算するタイミングを複数回スキップする事ができる。この方法は、連成計算するタイミング (nStepsFsi 値) を変更することで実現できる。

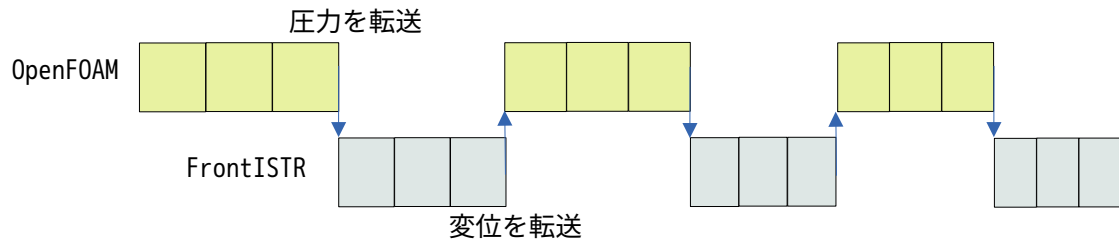
流体の OpenFOAM と構造の FrontISTR を同時に計算させる方法を選択している場合、各々の待ち時間が減少するので、計算時間が短縮できる。さらに、連成するタイミングをスキップさせる事によって、計算時間が短縮できる。

以下の例では、nStepsFsi=3 として、3 回スキップする例になる。

この例では、FrontISTR の方が計算時間が短いので、OpenFOAM 側は、殆どノンストップで計算が進んでいく。

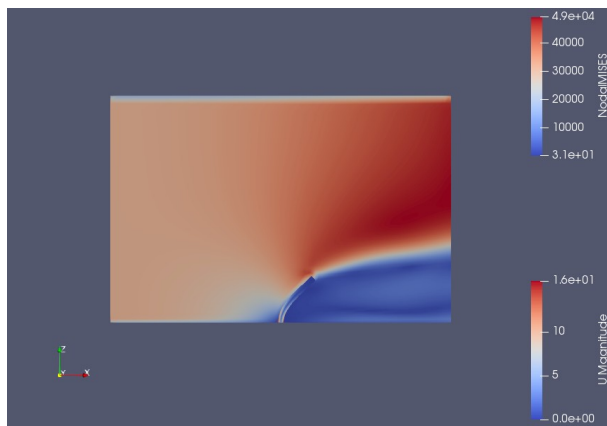


流体と構造を交互に計算させる方法を選択している場合は、下図の様に各々の待ち時間が生じるので、計算時間がかかるが、流速に応じて deltaT を変化させる事ができる（クーラン数一定）為、安定して計算させる事ができる。流速が変動する様なモデルでは、この方法が早くなる場合がある。



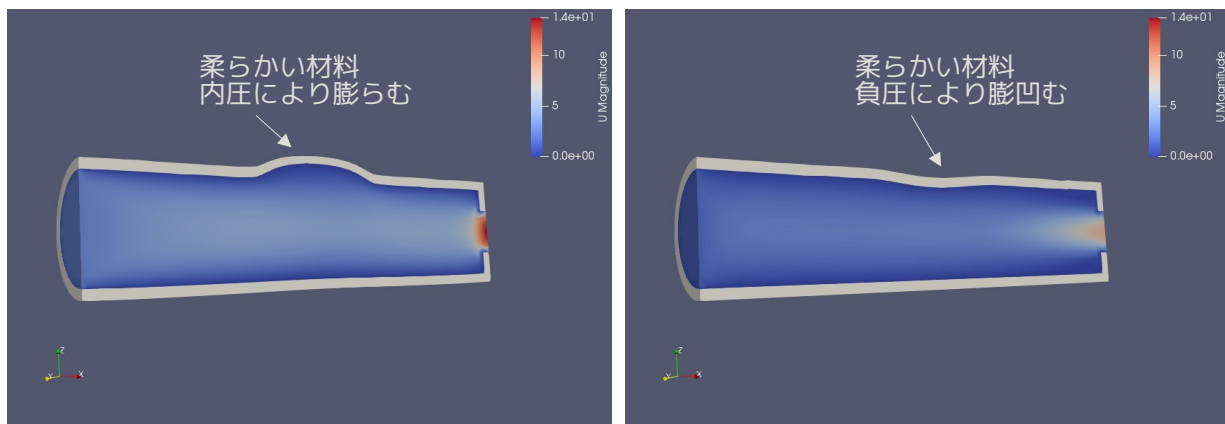
下図は、「flap\_perp\_OpenFOAM-FrontISTR」を 1.5 秒まで計算した結果が以下になる。solid 側は、応力を表示させている。

計算をさらに 5 秒まで継続するのであれば、startFrom を latestTime に変更し、endTime を 5 に変更し、「設定取得・保存」「連成計算開始」ボタンをクリックする事で、計算が継続できる。この時、deltaT や writeInterval を変更して、計算を継続させる事もできる。



softPipe\_OpenFOAM-FrontISTR の計算例 (pimpleFoam)

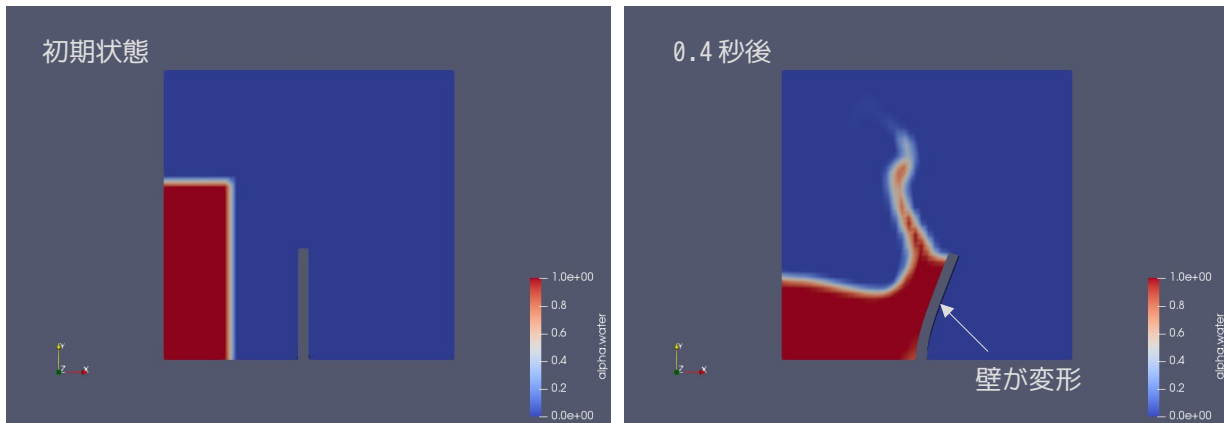
pipe の一部を柔らかい材料に変更したモデルで、流速を変動させている。圧力変動により、pipe が部分的に膨らんだり凹んだりする。



waterToWall\_0F9-FrontISTR の計算例 (interFoam)

damBreak の様に左端の水が壁に衝突し、壁が変形する様子。

この case は、Tutorials 内の「multiphase:多相流」>「interFoam」>「laminar/damBreak」(0F-11 の場合は、「foamRun:一般的なCFD」>「incompressibleVoF」>「damBreakLaminar」) をコピーして、メッシュ、境界条件などを修正して、流体側の case を作成している。

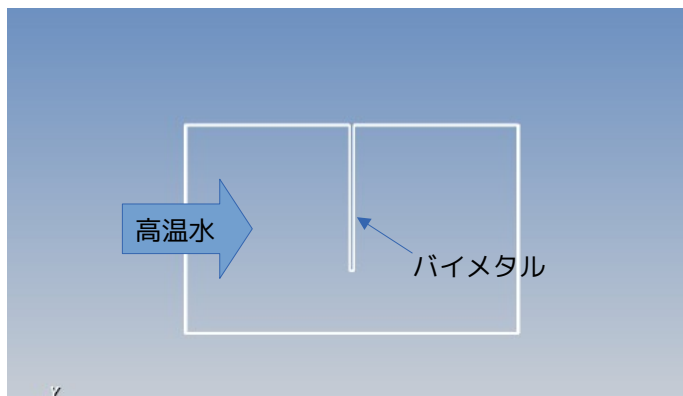


#### 8-4-2-2. 流体構造、熱連成、熱ひずみ解析の具体例

以下の様なモデルを考える。モデル左側から高温水が流入する。モデル中央の天井にバイメタルが接続されている。この為、モデル中央のバイメタルは、流れの圧力による変形と温度上昇による変形が加わって変形していくモデルになる。

また、圧力と温度上昇の変形を確認する為に、  
 流入速度は、 $0 \rightarrow 1\text{s}$  で一定値 ( $0.5\text{m/s}$ ) に到達させる。  
 バイメタルが発振しない様にゆっくり流入速度を上昇させる  
 バイメタルの発振を無くすために、粘性を設定。  
 固有値解析の結果、1次の固有振動数が  $13.7\text{Hz}$  であり、設定した粘性 ( $Rm=0.8$ 、 $Rk=0.8$ ) で  
 周波数応答を確認して、1次の共振をなくしている。  
 を設定してバイメタルの変形を確認する。

この case は、熱移動と圧力を計算する必要がある為、tutorials 内の「heatTransfer:熱輸送と浮力駆動流れ」>「buoyantPimpleFoam」>「hotRoom」(OF11では、「foamRun:一般的なCFD」>「fluid」>「hotRoom」) をコピーして、メッシュ、境界条件を修正し、physicalProperties 内の流体を「ガス」→「水」に変更、momentumTransport 内の simulationType を「RAS」→「laminar」に変更して流体側の case を作成している。



この例は、`$TreeFoamPath/frontIstr/tutorials/OF-9/buoyantPimpleFoam` フォルダ内に「biMetal\_OF9-FrontISTR.zip」として保存している。この圧縮ファイルを適当な場所で展開する。

展開すると以下のフォルダが確認できる。

```
biMetal_OpenFOAM-FrontISTR/
  fluid3/                      #TreeFoam の解析 folder に設定する
  0/
  constant/
  coupling_FrontISTR/          #連成解析の folder
  data/                        #dataFolder
  python/                     #連成解析に必要なスクリプト
```

couplingData	#連成解析に必要な data を保存
system/	
runFsi	#連成計算開始スクリプト
solid3/	#FrontISTR の case

TreeFoamを起動して、fluid3 フォルダにし点マークを付け、解析 folder に設定する。  
この後、メニュー「計算」>「流体-構造連成解析」を選択して、連成解析用の dialog を起動する。

dialog表示後、まず、赤枠内の連成する項目から fsi, cht, chtss を選択する。さらに EasyISTR の実行 file 「easyistr」と FrontISTR の実行 file 「fistr1」の場所を設定する。  
残りの項目は、既に設定済のため、設定は不要。以下にその内容を説明する。

#### <連成解析の内容>

OpenFOAM と FrontISTR の計算方法を同時に計算させるか、交互に計算させるかを選択する。  
同時計算は、deltaT 一定で高速。交互計算は、maxCo 制御で deltaT が流速に応じて変動するので、計算が安定する。

#### <OpenFOAM 側の設定>



p field の係数	取得した圧力に乗じる係数 今顔の solver buoyantPimpleFoam の場合、圧力は、そのまま出力されているので「1.0」を設定。
p field の shift 量	取得した圧力の shift 量 今回の case の場合、圧力が絶対圧で出力されているので、大気圧分を減じる為に「-1e5」を入力している。
scalarField	圧力と熱流束を取得する field を指定する。 今回の case の場合、温度と圧力の両方を取得する必要がある。このような場合は、「p,T」を選択する。
pointField	変位を設定する field を指定。
patch	圧力値や熱流束を取得する patch 名を指定。
並列計算する	並列計算する場合は、チェックする。 予め並列計算ができる様に、decomposePar で並列計算用にメッシュ分割しておく。 OpenFOAM と FrontISTR の同時計算を選択している場合は、FrontISTR 側の thread 数を考慮して設定する。
<FrontISTR 側の設定>	
easyistr	EasyISTR の実行 file 「easyistr」の場所を指定する。
fistr1	FrontISTR の実行 file 「fistr1」の場所を指定する。
solidDir	FrontISTR の解析 folder を相対 path で指定する。
SGRP	構造側の surfaceGroup 名を指定する。この SGRP 名と OpenFOAM 側の patch 名が対応し、OpenFOAM 側の圧力値や熱流束が、この face に mapping される。
並列計算する	FrontISTR を thread 並列する場合は、チェックし、使用する全 core 数を入力する。 OpenFOAM と FrontISTR の同時計算を選択している場合は、OpenFOAM 側の並列数を FrontISTR 側の process 数と thread 数（全 core 数）を考慮して設定する。
<連成計算するタイミング>	
nStepsFsi	構造と連成する step 数を指定。 「1」を入力すると deltaT 時間毎に構造側と連成する。 今回は「2」を入力しているので、deltaT*2 時間毎に連成計算する。
nStepsHeat	熱連成する step 数を指定 今回は、「10」を入力しているので、deltaT*10 時間毎に連成計算する。 熱計算の反応速度は、構造計算の反応速度に比べて、非常に遅い。この為、ここでは、「10」を入力している。
<各 Job の待ち時間>	
最大待ち時間	各 Job をキャンセルするまでの時間を指定。 この時間内で、OpenFOAM、FrontISTR 側とも 1step の計算が終了する必要がある。 終了しない場合は、Job がキャンセルされ、エラー停止する。

圧縮ファイルを展開した状態は、OpenFOAM 側、FrontISTR 側共に各々単独で動く状態になっている。（まだ、連成計算ができる状態にはなっていない。）

確認の為、今の状態で、OpenFOAM 側、FrontISTR 側がエラーなく単独で動くかどうか確認する。

OpenFOAM 側は、TreeFoam 上で、▶ アイコンをクリックして、計算開始させて、動作を確認する。

FrontISTR 側は、連成計算 dialog 内の「EasyISTR 起動」ボタンで、EasyISTR を起動してその作動を確認する。

FrontISTR 側の計算は、fsi:圧力-変位計算、cht:熱計算、chtss:熱ひずみ計算の3種類の計算を行う必要があり、各々設定内容が異なっている。この為、FrontISTR の cnt ファイルを3種類保存できる様にしている。「EasyISTR 起動」ボタンをクリックすると、以下の dialog が表示されるので、ここで、どの cnt ファイルを修正するのかを決定後、EasyISTR を起動して、内容確認、編集を行う。



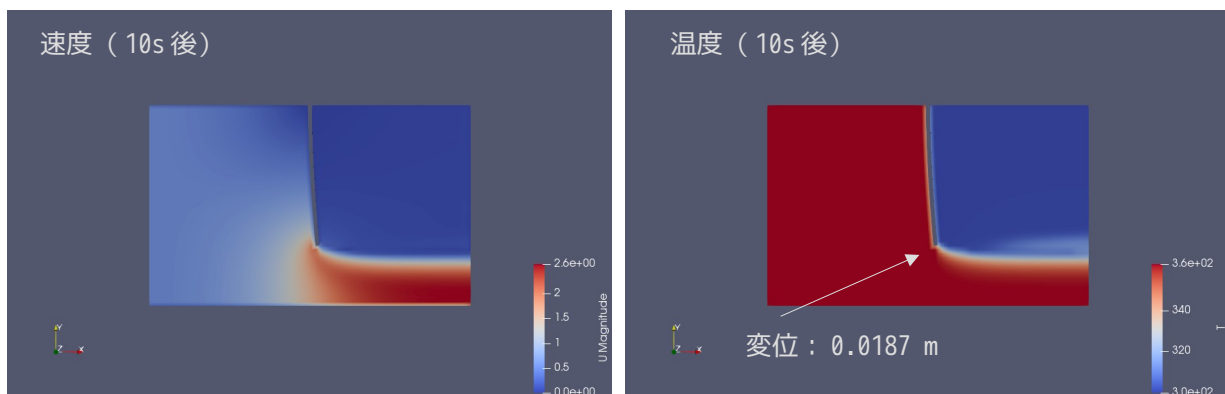
尚、EasyISTR 上で編集した場合は、必ずメニュー「ファイル」>「保存 (temp → dir)」を選択して、修正した設定内容を実フォルダに反映させる。

連成計算開始は、以下のボタンをクリックして計算開始する。

- |                  |  |
|------------------|--|
| 「結果クリア」ボタン       | OpenFOAM、FrontISTR 両方の計算結果をクリアする。  |
| 「設定取得・保存」ボタン     | dialog の設定内容を couplingData ファイルに保存し、連成計算に必要なスクリプトをコピーする。OpenFOAM の該当 patch の内容を codedMixed, codedFixedValue に置き換える。FrontISTR 側の step 数、時間増分などを置き換える。 |
| 「連成計算開始」ボタン      | 連成計算を開始する。   |
| 「Fistr の log」ボタン | FrontISTR の出力 log を表示する。   |

今回の計算は、バイメタルが発振しない様に粘性係数を設定していることから、deltaT を大きな値で計算させ、連成計算のタイミングも途中で skip させて、計算時間を早めている。

以下が連成計算結果になる。



バイメタルが、圧力と温度上昇により変形している。

#### 8-4-2-3. 熱連成のみの解析の具体例

流体固体の熱連成のみであれば、OpenFOAM の solver は、温度 T field が扱える solver で計算できる。以下は、reactingFoam (メタンの燃焼: CH<sub>4</sub> と O<sub>2</sub> の化学反応) を計算する solver と FrontISTR 間で熱連成を行った結果になる。

この例は、\$TreeFoamPath/frontIstr/tutorials/OF-9/reactingFoam フォルダ内に「fire\_OF9-FrontISTR.zip」として保存している。この圧縮ファイルを適当な場所で展開する。

展開後、TreeFoam を起動して、「fluid」フォルダにシ点マークを付け、解析 folder に設定する。この後、メニュー「計算」>「流体-構造連成解析」を選択して、連成解析用の dialog を起動する。

この case は、Tutorials 内の「combustion:燃焼」>「reactingFoam」>「laminar/counterFlowFlame2D」(OF-11, 12 では、「foamRun:一般的なCFD」>「multicomponentFluid」>「counterFlowFlame2D」) をコピーして、メッシュと境界条件を変更し、setFields で温度 T を設定して、流体側の case を作成している。

**流体-構造の連成解析**

指定したOpenFOAM側のpatch名とFrontISTR側のSGRP名を通じて連成する。  
計算は、startTime, latestTimeからも再開できる。

連成解析の内容

基本Dir: /home/caeuser/CAE/CAE-FOAM/OF-9/coupledFistr/fire\_OpenFOAM-FrontISTR/fluid

連成する項目

☐ fsi: 流体-構造の連成 (流体圧力->構造変位を計算)

☒ cht: 流体-固体の熱連成 (熱移動を計算) ☐ chtss: 固体の熱ひずみを計算

OpenFOAM(流体)とFrontISTR(個体)の計算方法

☐ 同時に計算 (高速) ☒ 交互に計算 (安定:maxCo制御で計算可能)

OpenFOAMの設定

連成するfield, patchを選択

p fieldの係数: 1.0 p fieldのshift量: 0.0

scalarField: T

pointField:

patch名

sideW

outlet

walls

frontBack

☐ 並列計算する

FrontISTRの設定

連成するsolidDir, SGRPを選択

command「easyistr」「fistr1」のpath設定

easyistr: /home/caeuser/easyIstr 参照...

fistr1: /usr/bin/fistr1 参照...

solidDir: ../solid 参照...

SGRP名

tempWall

frontBack

☒ 並列計算する 全core数: 4

連成計算するタイミング (deltaTの倍数:step数を設定)

流体-構造連成計算用のstep数: nStepsFsi: 1

熱移動、熱ひずみ計算用のstep数: nStepsHeat: 50

各Jobの待ち時間

最大待ち時間: 60 sec

結果クリア 設定取得・保存 連成計算開始 Fistrのlog

EasyISTR起動 流体folder開く 固体folder開く 閉じる

dialog中の赤枠内をまず確認する。

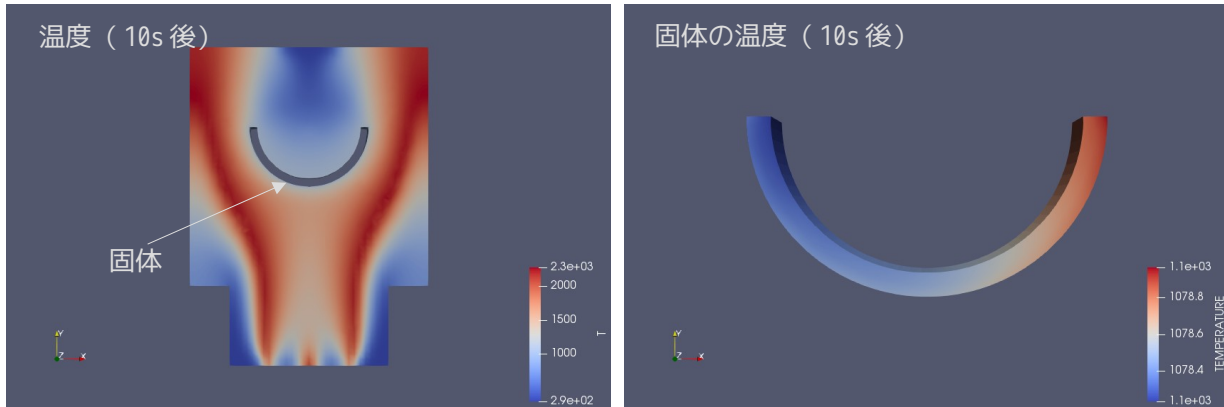
連成計算の為の設定は、既に設定済。計算開始させる為には、「結果クリア」「設定取得・保存」「連成計算開始」ボタンを順番にクリックすることで、連成計算が開始される。

この計算は、燃焼 (化学反応) を計算している為、deltaT が小さい値に設定されている。

しかし、固体の温度変化の速度は、それに比べると非常に遅い。この為、連成計算するタイミング (nStepsHeat) を 50 step に設定している。(1回/50回の頻度で連成計算させる設定)

FrontISTRの熱解析時間は、殆どかからない為、OpenFOAM側は、連成計算を感じさせないぐらい早く計算が進んでいく。

この条件で計算した結果が以下になる。

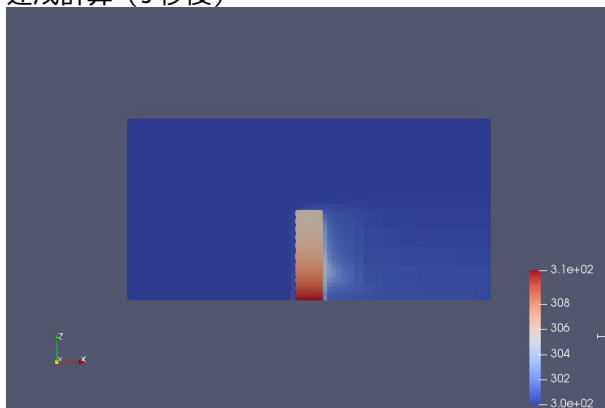


温度は、メタン  $\text{CH}_4$  の燃焼の為、 $\text{max}2000\text{K}$  を超えている。半円状の固体の温度は、 $300 \rightarrow 1100\text{K}$  に上昇する。  
(固体の比熱を小さくして、温度変化しやすいようにしている。)

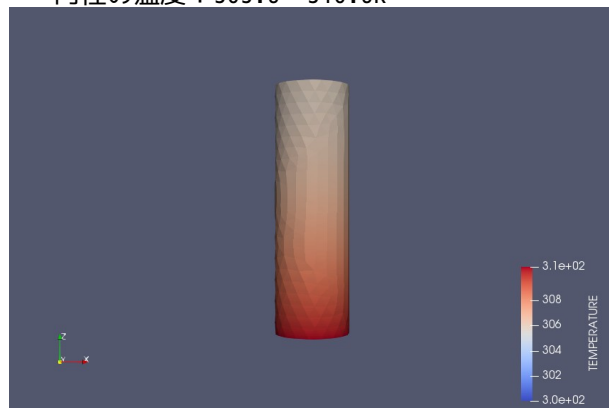
### 8-4-3. 熱連成計算の検証

流体固体間の熱連成計算の確からしさを確認する。  
確認方法は、OpenFOAM に元々準備されている `chtMultiRegionFoam` を使って結果を確認する。  
同じ形状、材質のモデルで、計算した結果を比較する。  
モデル形状、条件は、「9-5-1. multiRegion の case」で例題として使っているモデルになる。  
このモデルは、空気の流れの中に Cu の円柱を配置し、円柱の下面に  $186\text{e}3 \text{ W/m}^2$  の熱流束を与える。  
円柱 Cu の物性値は、密度： $8960 \text{ kg/m}^3$ 、熱伝導率： $372 \text{ W/m.K}$ 、比熱： $419 \text{ J/kg.K}$  で計算。  
3 秒後の計算結果が以下になる。

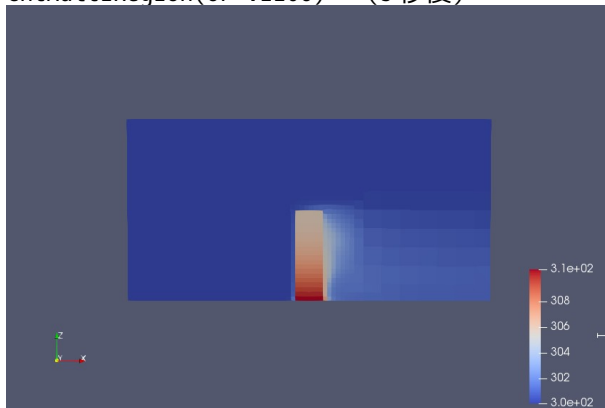
連成計算 (3 秒後)



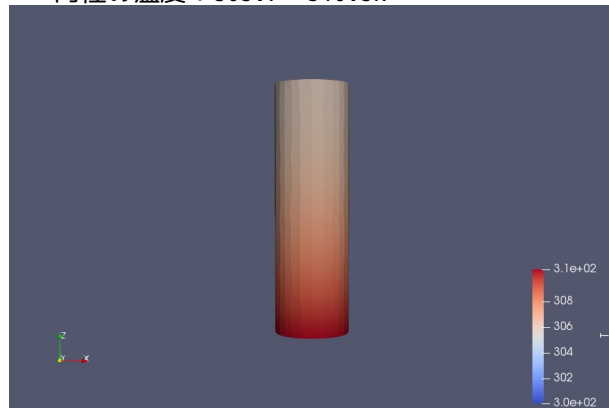
円柱の温度：305.6～310.6K



`chtMultiRegion(OF-v2206)` (3 秒後)

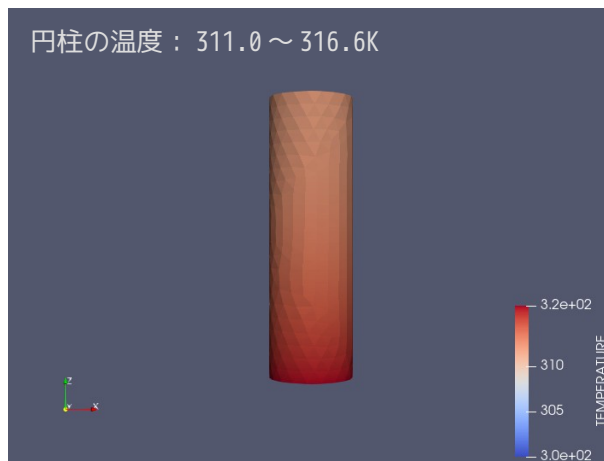


円柱の温度：305.7～310.5K



連成計算と `chtMultiRegionFoam` の計算結果は、ほぼ合致する結果になった。  
参考までに、円柱を断熱した状態で、円柱底面に  $186\text{e}3 \text{ W/m}^2$  の熱流速を 3 秒間与えた時の温度分布が以下

になる。(FrontISTR で計算)



円柱を断熱した状態では、max316.6K まで上昇している為、約 6K 分が空気側に移動した熱量になる。

#### 8-4-4. OpenFOAM、FrontISTR の他バージョンへの適用

今回の連成解析の Tutorials は、OpenFOAM-9 で作成している為、OpenFOAM-9 では、問題なく動く。  
他に OF-10, OF-11, OF-12, OF-13, OF-v2206, OF-v2306, OF-v2406, OF-v2506 でも作動を確認した。

また、OF-9、OF-v2206、OF-v2306 のバージョンでは、差は殆どないが、solver の仕様が微妙に変わっており、tutorials では、ここを反映させている。

OF-10 では、dynamicMeshProperties の書式が変わっている、buoyantPimpleFoam の solver が buoyantFoam に変わっている、熱伝導率  $\kappa$  の定義が変わり controlDict 内の code を修正している為、OF-10 用で tutorials を再作成している。

OF-11 では、連成計算用の code は、OF-10 と変えていないが、solver の構成が大幅に変わっているので、case を作り直している。

OF-11 では、sourceCode が変更されており、OF-11 用で作成している。

tutorials には、OpenFOAM の version 毎に folder を作成して、この中に case を圧縮して保存している。

連成解析の case の作成方法は、  
     OpenFOAM 単独で動く case を作成する。  
     FrontISTR 単独で動く case を作成する。  
     OpenFOAM、FrontISTR を連成させて計算させる。  
 の順で確認し、計算する。

FrontISTR のバージョンについては、FrontISTR-5.2 以上で連成計算する。  
 FrontISTR-5.1.1 では、restart ファイルを作成しない、圧力「0」では、計算結果を出力しない為、連成計算できない。




## 9. 応用例

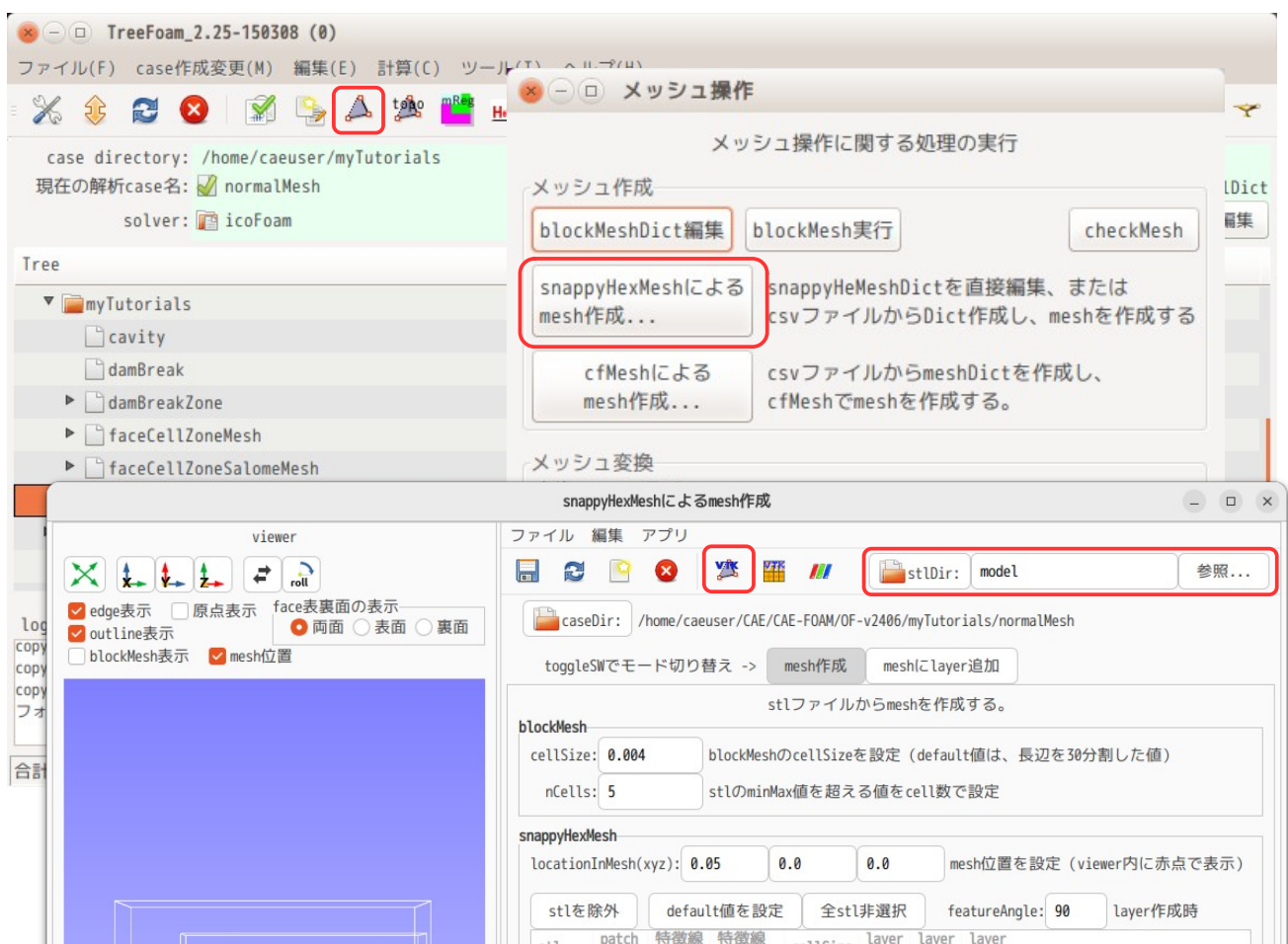
### 9-1. ファイルの操作・編集

#### 9-1-1. stl ファイルの編集

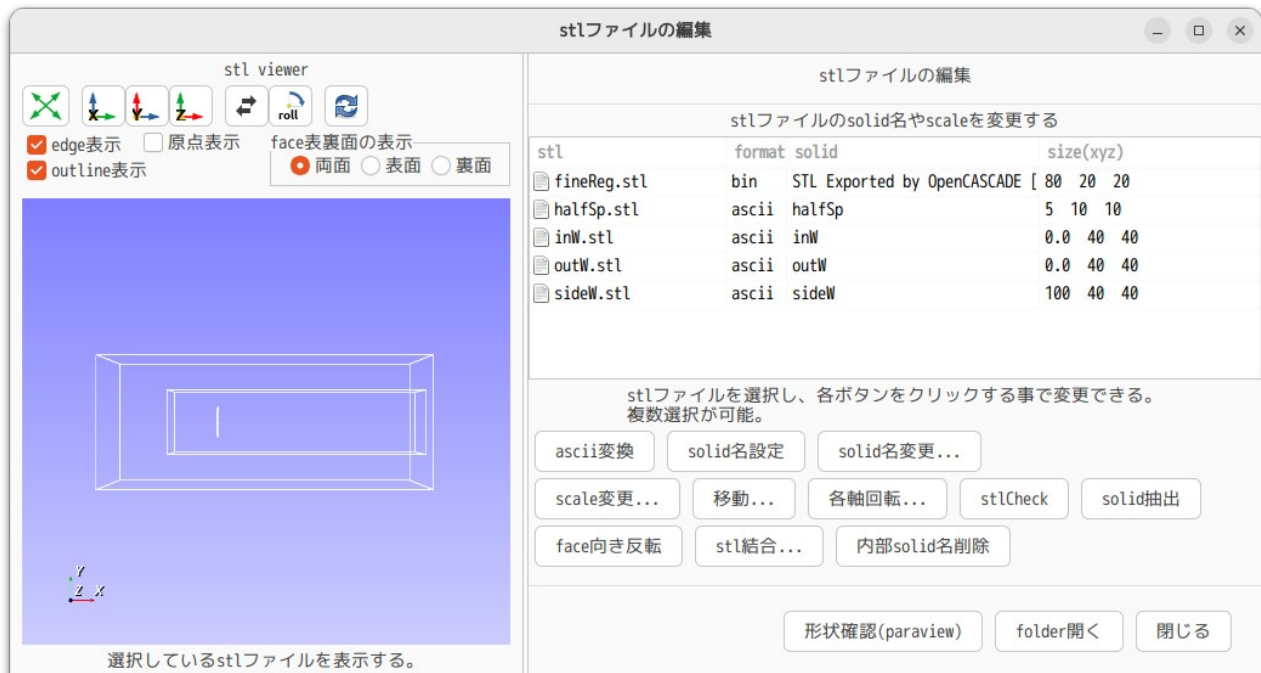
snappyHexMeshを使ってメッシュを作成する時、stl ファイルを元にメッシュを作成している。この stl ファイルを編集 (scale 変更、binary を ascii に変換する、stl ファイルの結合、face の向き反転等) を GUI で行なえるツールを作成している。次項以降にその操作方法について示す。

##### 9-1-1-1. 起動方法、起動画面

TreeFoam 上で本格的に stl ファイルを扱う場面は、現在のところ、snappyHexMesh と cfMesh を使う時しか無いので、起動は、「snappyHexMesh による mesh 作成」又は「cfMesh による mesh 作成」画面上の  ボタンをクリックして起動する。起動する前に、stl ファイルの保存場所 (stlDir) を確認した上で起動する。

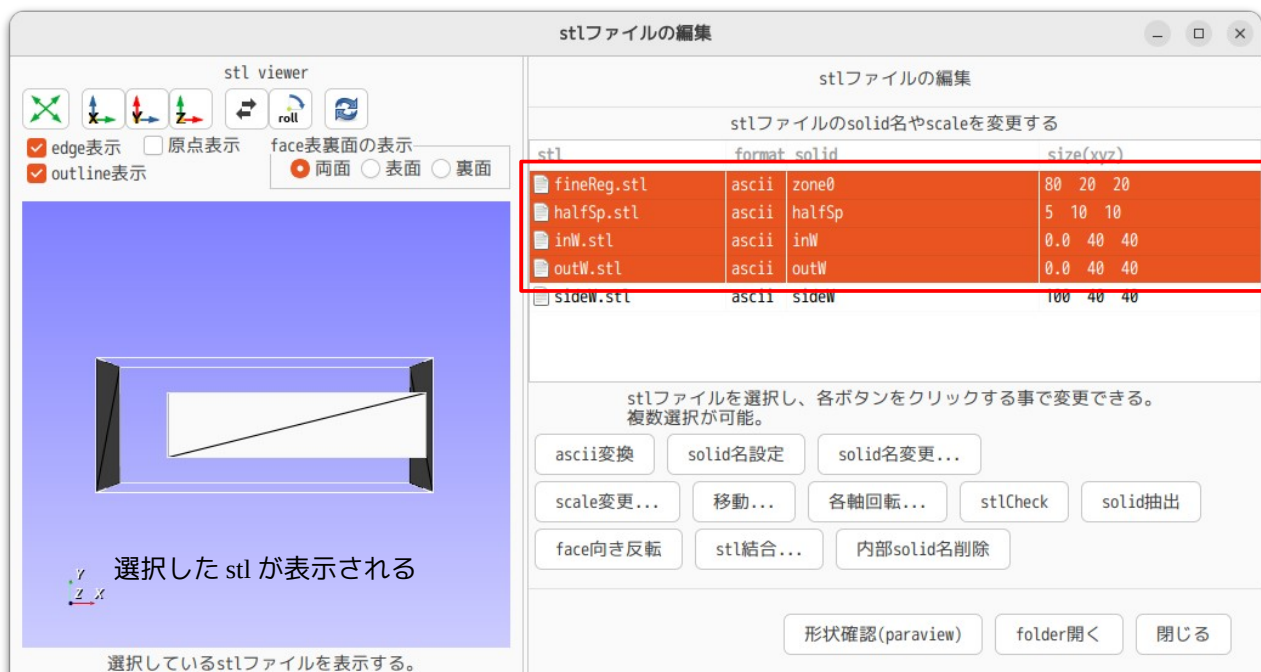


「stl チェック...」ボタンで「stl ファイルの編集」画面を起動すると、以下の画面が現れる。

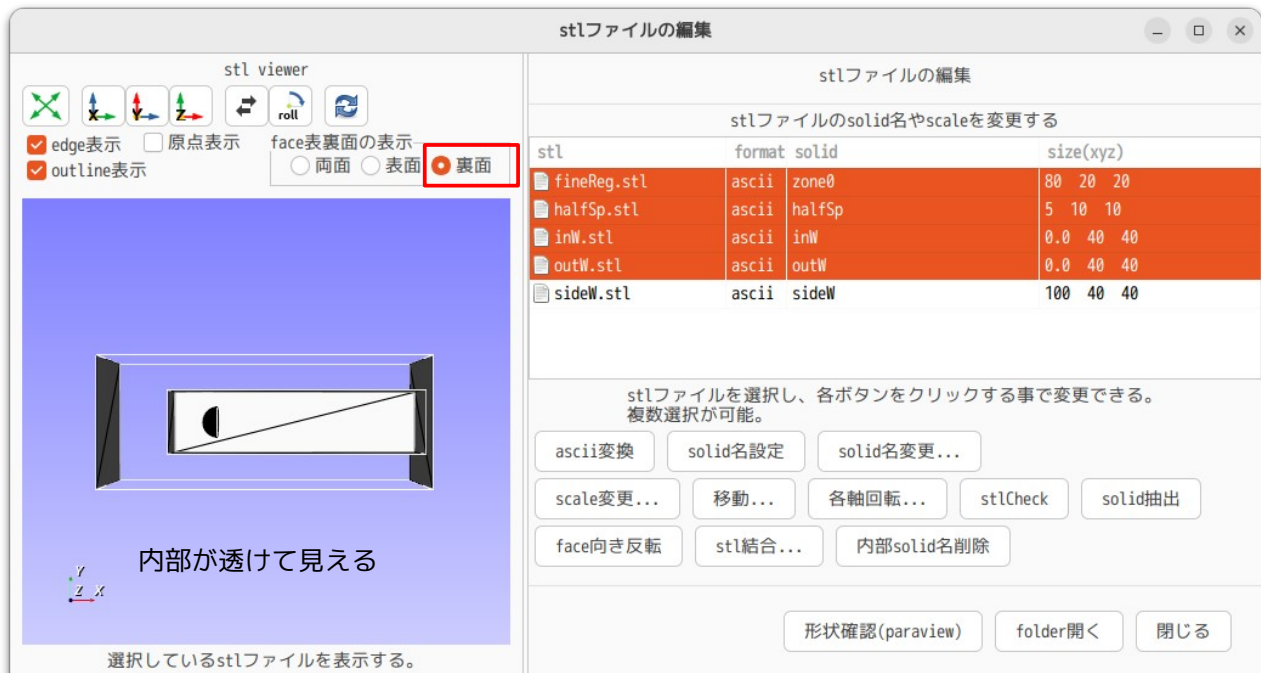


この画面左側の viewer には、「./model」フォルダ内に保存されている全ての stl ファイルの outline (feature edge) が表示されている。  
画面右側には、同じく全 stl ファイルの内容が list 表示されている。

尚、リスト内の stl ファイルを選択すると、選択した stl ファイルの形状が左画面上に表示される。stl ファイルの選択は、複数選択が可能であり、以下は複数選択した状態になる。



さらに、face 面の裏側のみ表示させる事ができ、この表示方法では、内部に隠れている形状を表示させる事ができる。その方法は、「裏面」のラジオボタンをチェックする事で確認できる。(下図参照)



この表示は、表面が隠れて、裏面のみ見えている状態のため、内部にある stl 形状が透けて見える状態になる。以上の様に、この画面上では、各 stl の形状を確認しながら、stl ファイルを編集する事ができる。

#### 9-1-1-2. ascii 変換

stl ファイルには ascii 形式と binary 形式の 2 種類の形式が存在しているが、OpenFOAM で扱う場合、ascii 形式の方が扱いやすい。ファイルの書式が binary 形式の場合は、ここで ascii 形式に変換できる。変換の前に、ascii 形式と binary 形式の保存形式を確認すると、これらは、1)、2)項に示す形式になっている。

##### 1) ascii 形式

ascii 形式の場合以下の様に、1 行目と最終行に solid 名「inW」が記述され、この間には三角形の数分の向きと座標のセットが記述されている。

```

1 solid inW
2 facet normal -1 -0 -0
3   outer loop
4     vertex 0 0.02 0.02
5     vertex 0 0.02 -0.02
6     vertex 0 -0.02 0.02
7   endloop
8 endfacet
9 facet normal -1 0 0
10  outer loop
11    vertex 0 0.02 -0.02
12    vertex 0 -0.02 -0.02
13    vertex 0 -0.02 0.02
14  endloop
15 endfacet
16 endsolid inW

```

solid 名「inW」を定義  
 三角形の向き (単位ベクトル)  
 三角形の座標

三角形の向き (単位ベクトル)  
 三角形の座標

最後に solid 名「inW」が記述

##### 2) binary 形式

binary 形式の場合、以下の様に、先頭から 80 byte が header、4 byte が三角形の数、この後、三角形の数分の向きと座標のセットが続く。

```

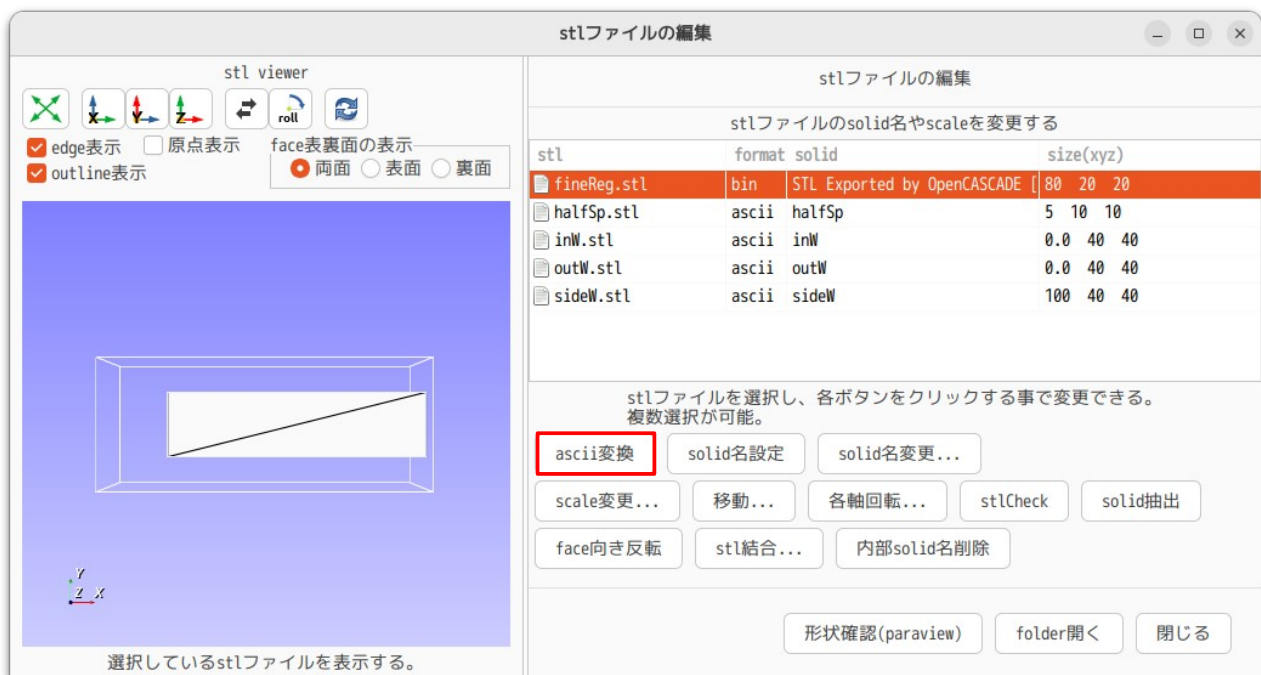
1 80 byte string header (この部分は規定がないが、solid 名やコメントが記述される場合がある)
2 4x1 byte int 三角形の数
3 4x3 byte float 三角形の向き (単位ベクトル)
4 4x3 byte float 三角形の座標
5 4x3 byte float ↑
6 4x3 byte float ↑
7 2 byte - 未使用
8 4x3 byte float 三角形の向き (単位ベクトル)
9 4x3 byte float 三角形の座標
10 4x3 byte float ↑
11 4x3 byte float ↑
12 2 byte - 未使用
13 :

```

これを踏まえた上で ascii 変換してみる。

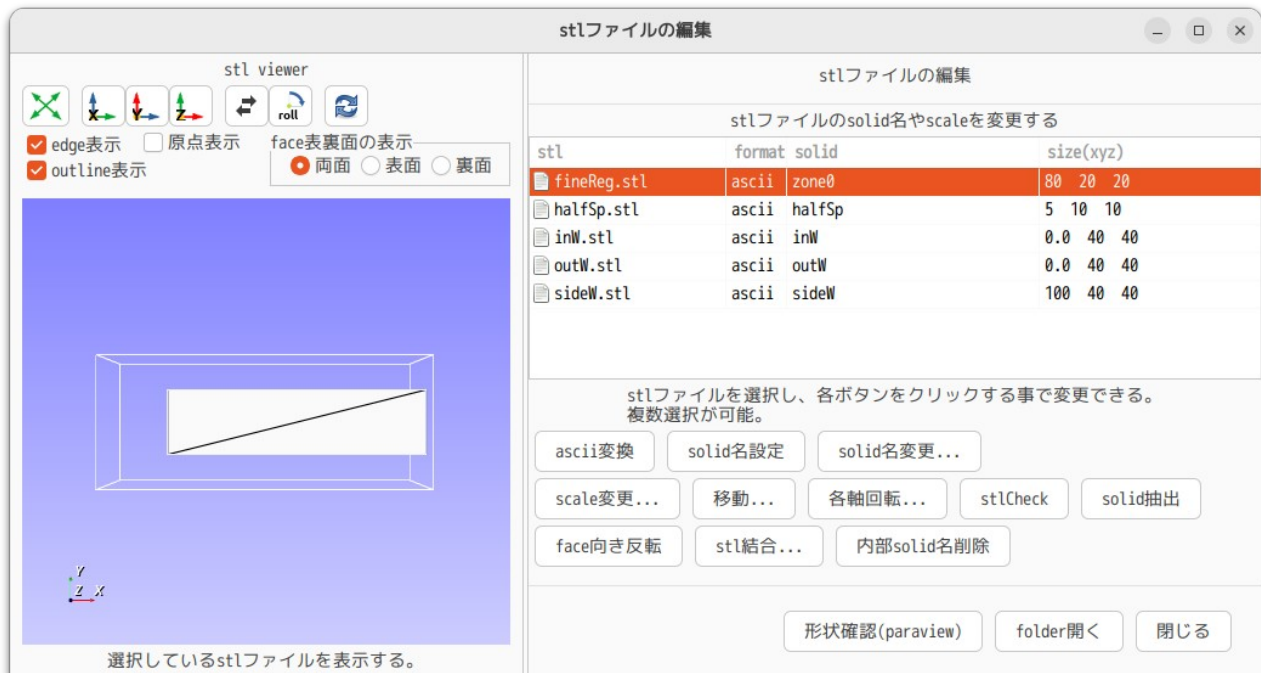
「stl ファイルの編集」画面では、stl ファイルが ascii でも binary でもその形式を判断して読み込む事ができ、そのファイル形式、solid 名 (binary の場合は、header の内容)、stl ファイルのモデルの大きさを表示している。

ファイルリスト中の「fineReg.stl」は、binary 形式である事が判る。  
この binary 形式のファイルを ascii 形式に変換してみる。その変換方法は、以下の様に ascii 変換したい stl ファイルを選択し、「ascii 変換」ボタンをクリックする。



変換が完了すると、以下の様に、ファイル形式が「ascii」に変わる。



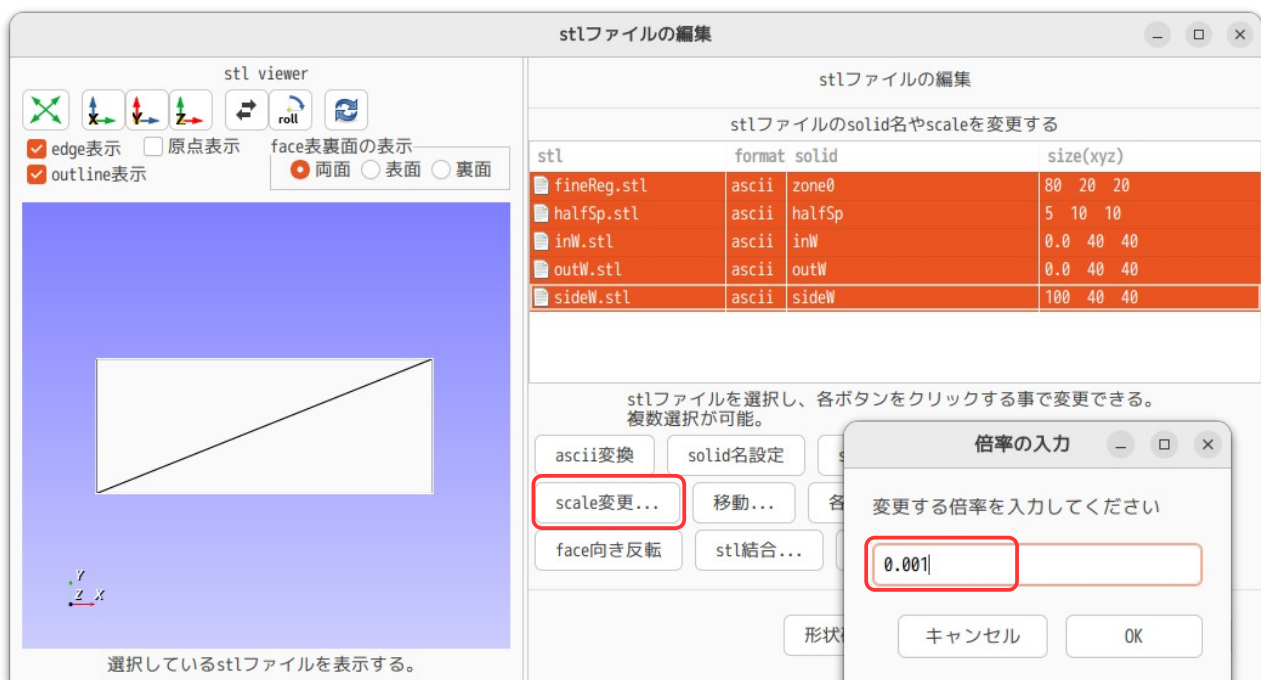


この処理は、logを見て判るように「surfaceTransformPoints -scale '(1.0 1.0 1.0)」」を実行しているのみ。このコマンドを実行すると、形式を ascii に変換してくれるので、このコマンドを使っている。

### 9-1-1-3. scale 変更

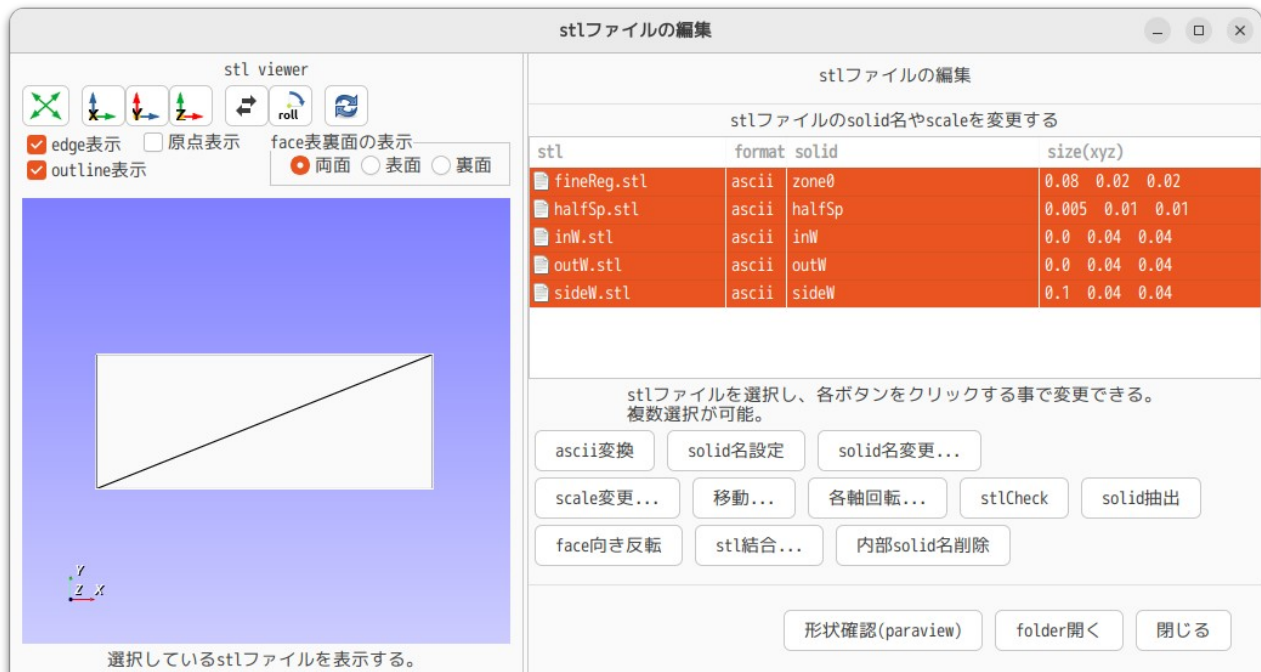
作成したモデル (stl ファイル) の単位は、リスト内の size(xyz)の値から、mm 単位である事が確認できる。この為、全ファイルの scale を変更して m 単位に設定する。

scale を変更するためには、以下の様に、変更したいファイルを選択し、「scale 変更」ボタンをクリックして、倍率「0.001」を入力する。





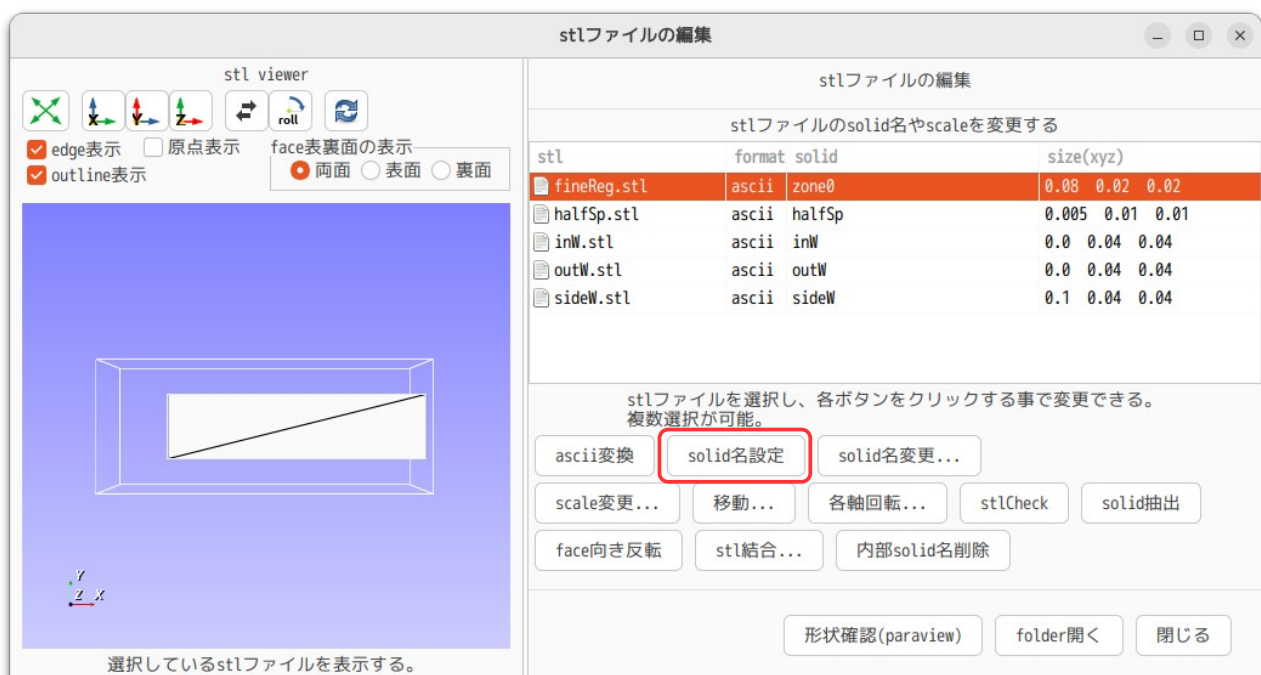
scale が変更されると、以下の様に「size(xyz)」の値が 1/1000 のサイズに変わる。



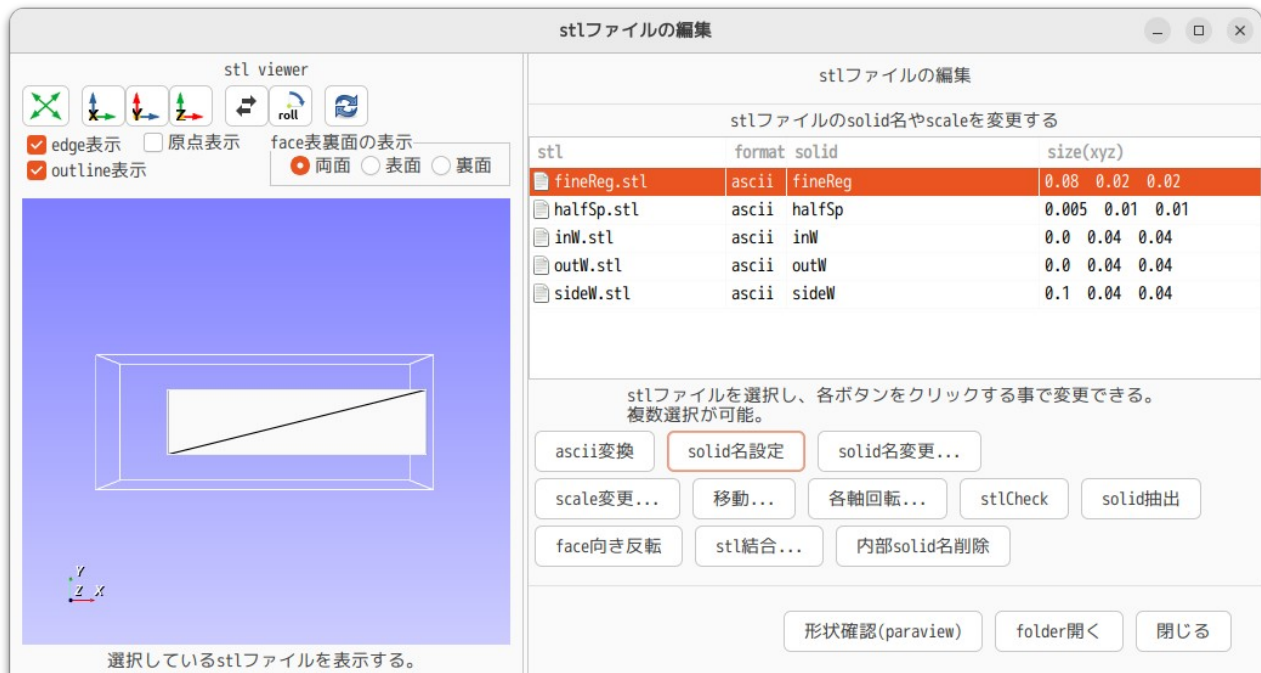
#### 9-1-1-4. solid 名設定

fineReg.stl の solid 名は、scale 変更時に surfaceTransformPoints コマンドが、勝手に設定した solid 名の為、この名称を stl ファイルの名称に変更する事で、solid 名を意味のある名称に変更する事ができる。

この方法は、以下の様に変更したいファイルを選択し、「solid 名設定」ボタンをクリックする。

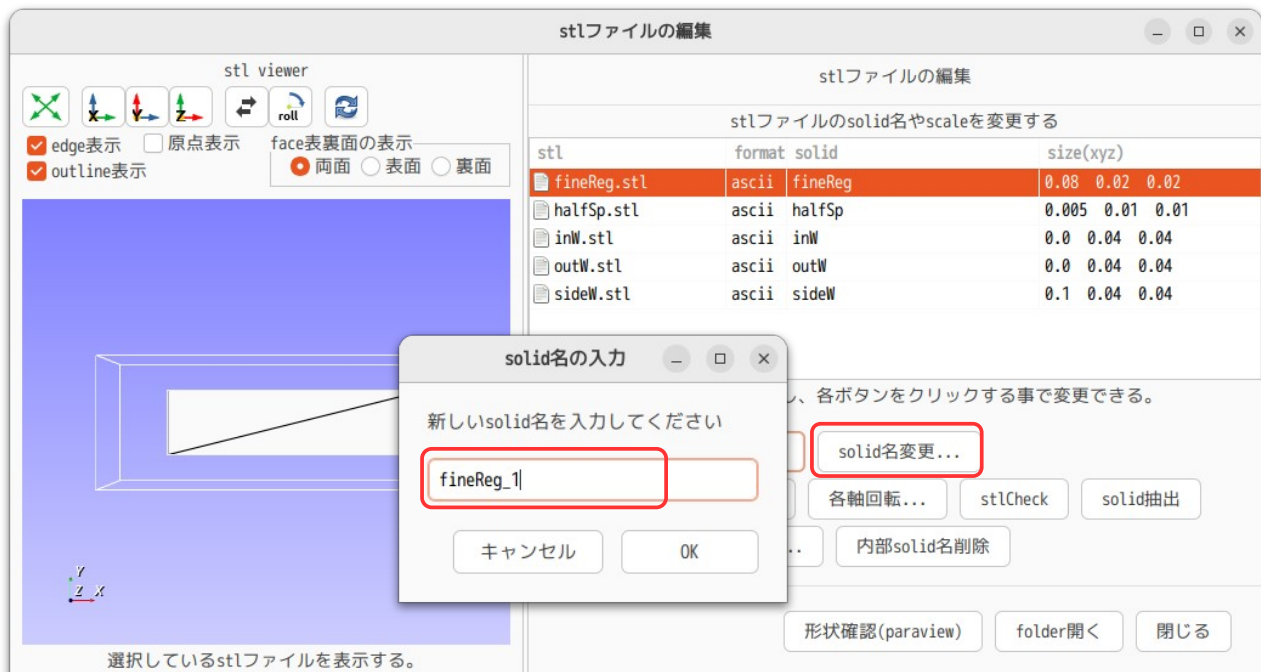


以下が、変更した結果になる。solid 名が、ファイル名に変更されている。

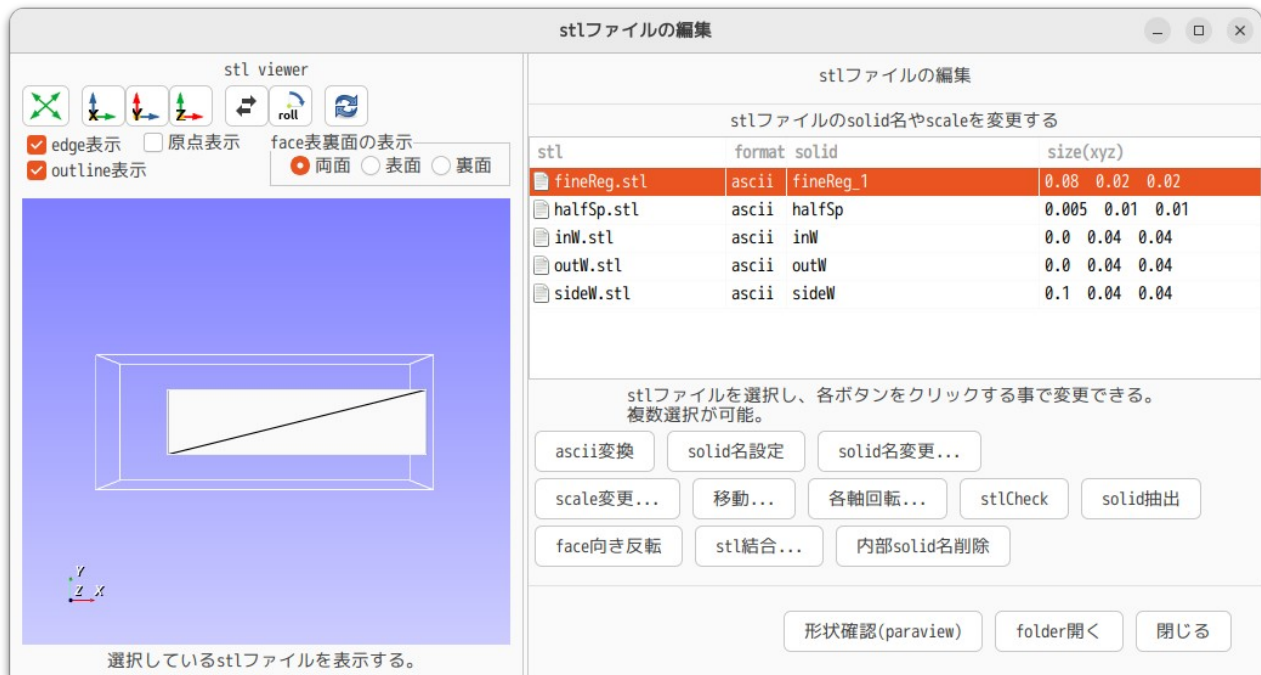


#### 9-1-1-5. solid 名変更

前項は、複数の stl ファイルを一括して solid 名を設定する事ができるが、これを個別に特別な名称を設定したい場合は、以下の様に対象のファイル 1 枚を選択し「solid 名変更...」ボタンをクリックして新しい solid 名を入力して変更できる。

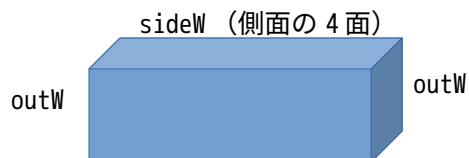


以下が変更後の画面になる。solid 名が変更されている。

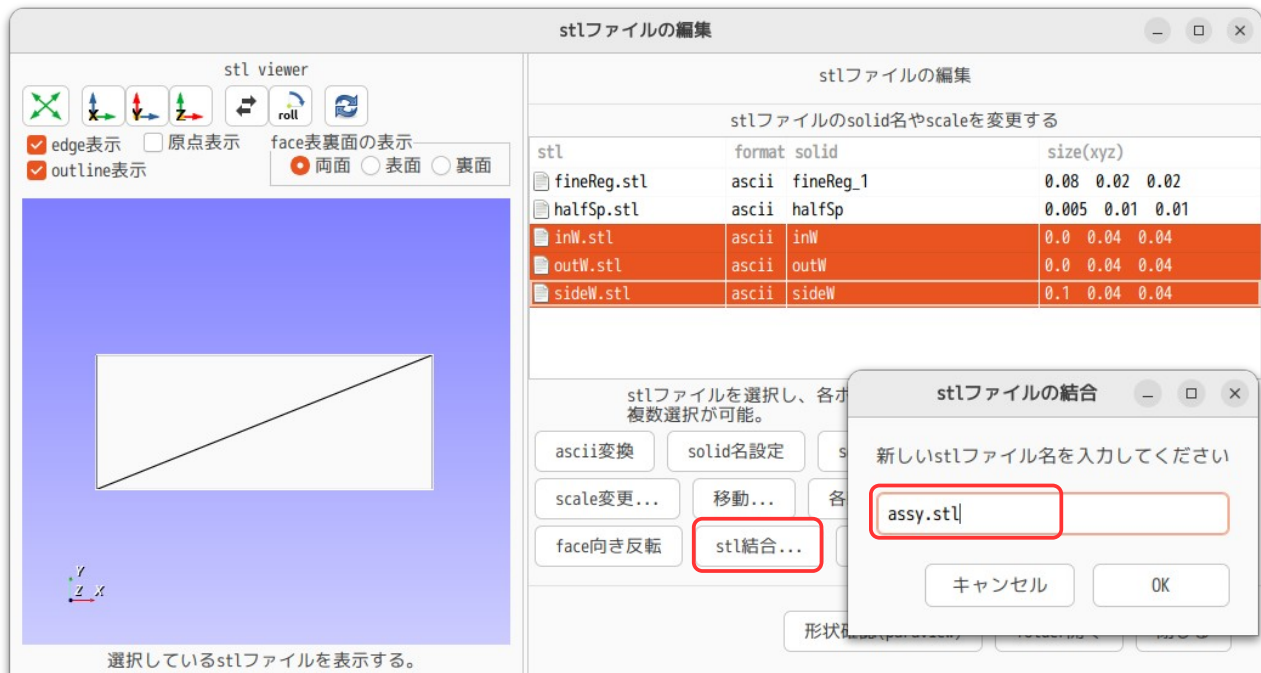


#### 9-1-1-6. stl 結合

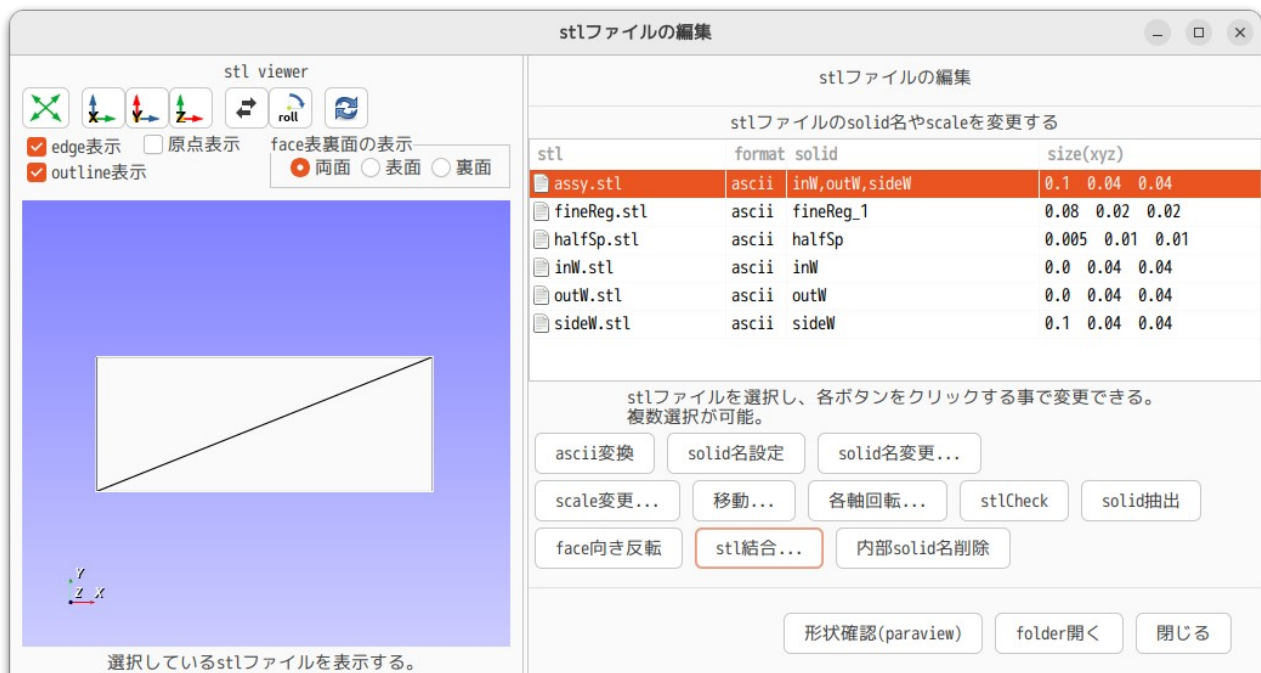
複数の stl ファイルを solid 名付きで結合したい場合には、この方法を用いる。  
例えば、以下の様なモデルの場合、「inW」「sideW」「outW」でモデル全体を表しているので、これらを solid 名付きで 1 枚の stl ファイルを作りたい場合に用いる。



その方法は、以下の様に、結合したい stl ファイル名を選択し、「stl 結合」ボタンをクリックして、ファイル名を入力する事で、結合された stl ファイルを作成する事ができる。



以下が結合した結果の画面になる。「assy.stl」ファイルが追加され、solid名「inW, outW, sideW」が確認できる。内部のsolid名が表示されるので、何を結合したものが確認できる。

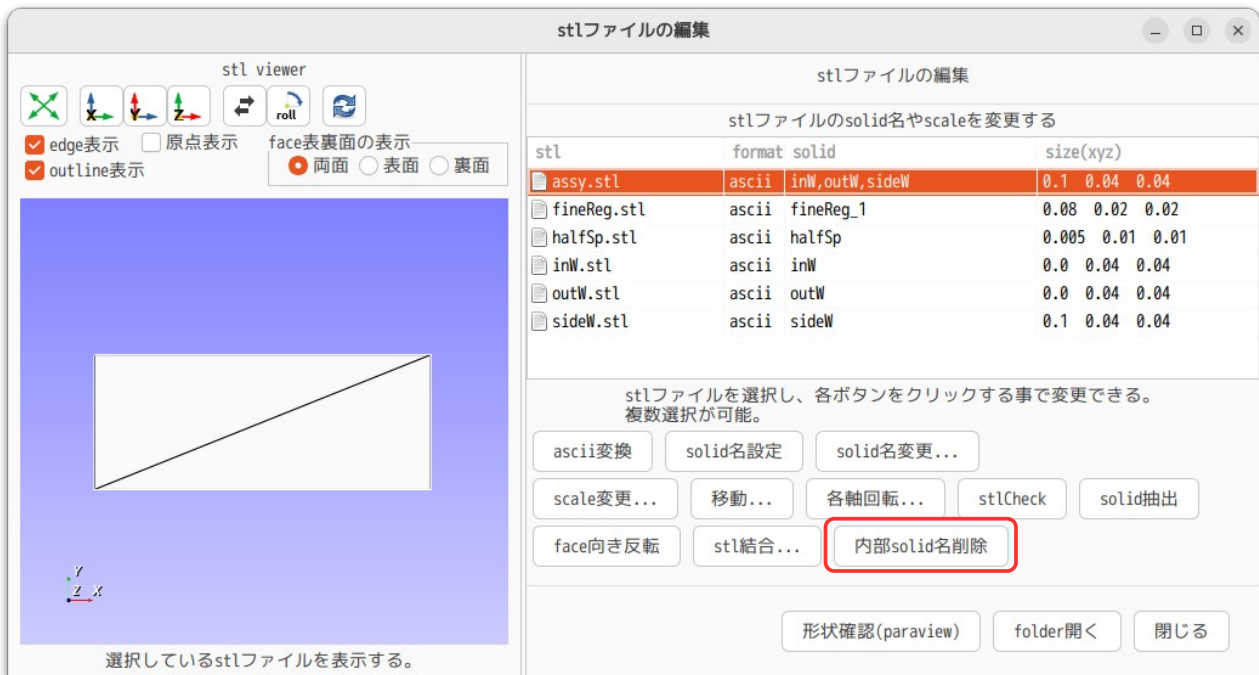


#### 9-1-1-7. 内部 solid 名削除

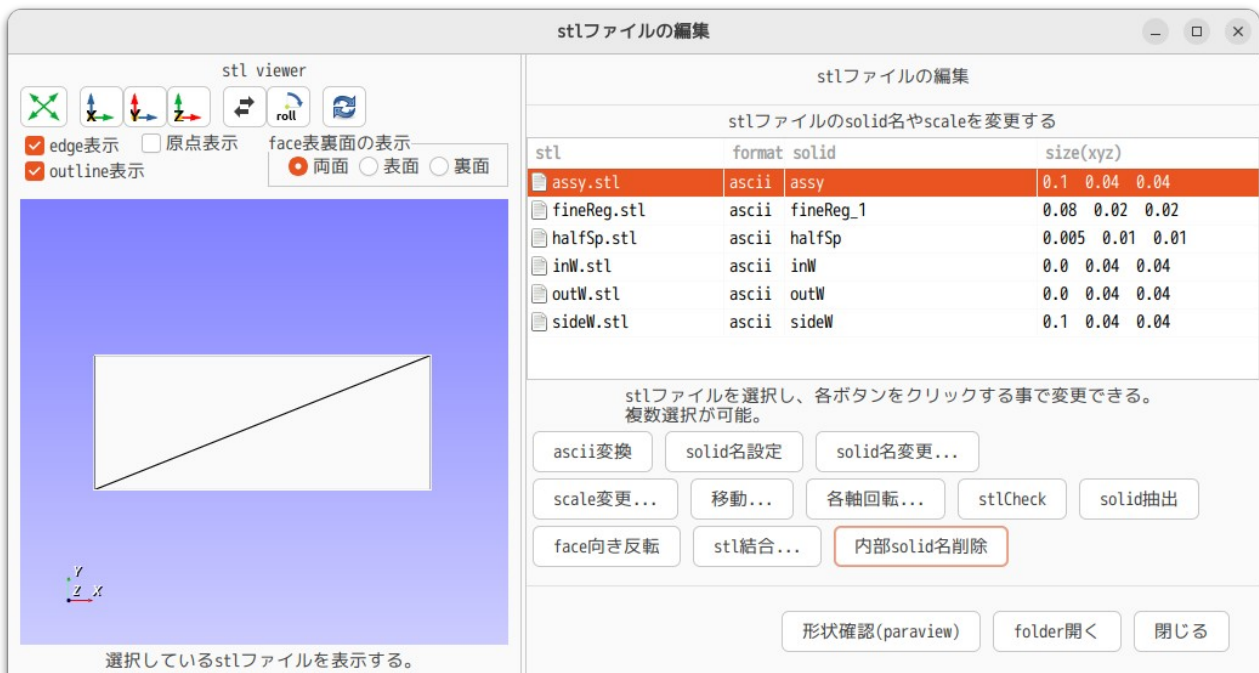
前項で stl ファイルの内部に solid 名を含んだ stl ファイルを作成したが、この solid 名を削除して、1 塊の塊とした stl ファイルを作成する場合は、この方法を用いる。

その方法は、以下の様に、修正したい stl ファイルを選択し、「内部 solid 名削除」ボタンをクリックすることで修正できる。





以下が、内部 solid 名を削除した結果になる。solid 名が「assy」に変わっている。



#### 9-1-1-8. faceの向き反転

stl ファイルの face の向きを反転させたい場合には、これを用いる。

face の向きを反転させる方法は、三角形のベクトルの向きと、三角形の座標の順番を入れ替えて実現している。以下の例では、2 行目のベクトルの向きを変えて、5 行目と 6 行目を入れ替えている。



&lt;変更前&gt;

```

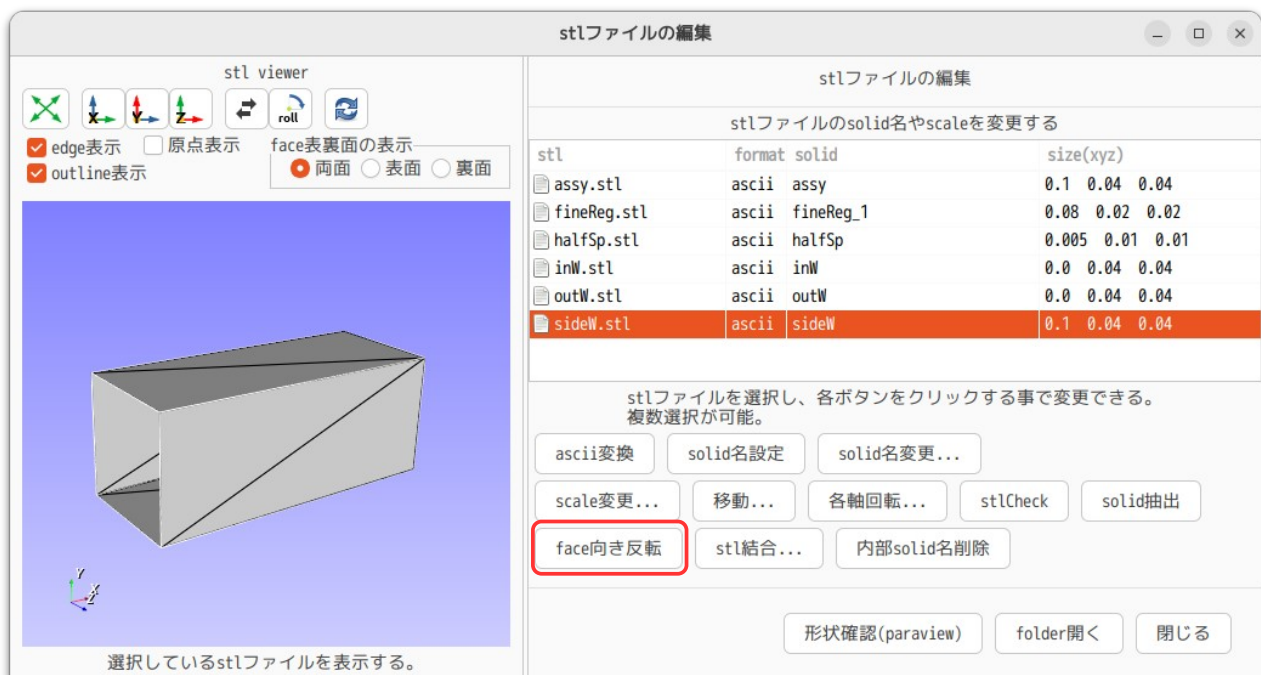
1 solid inW
2 facet normal -1 -0 -0
3   outer loop
4     vertex 0 0.02 0.02
5     vertex 0 0.02 -0.02
6     vertex 0 -0.02 0.02
7   endloop
8 endfacet
:
```

&lt; face 向き反転 &gt;

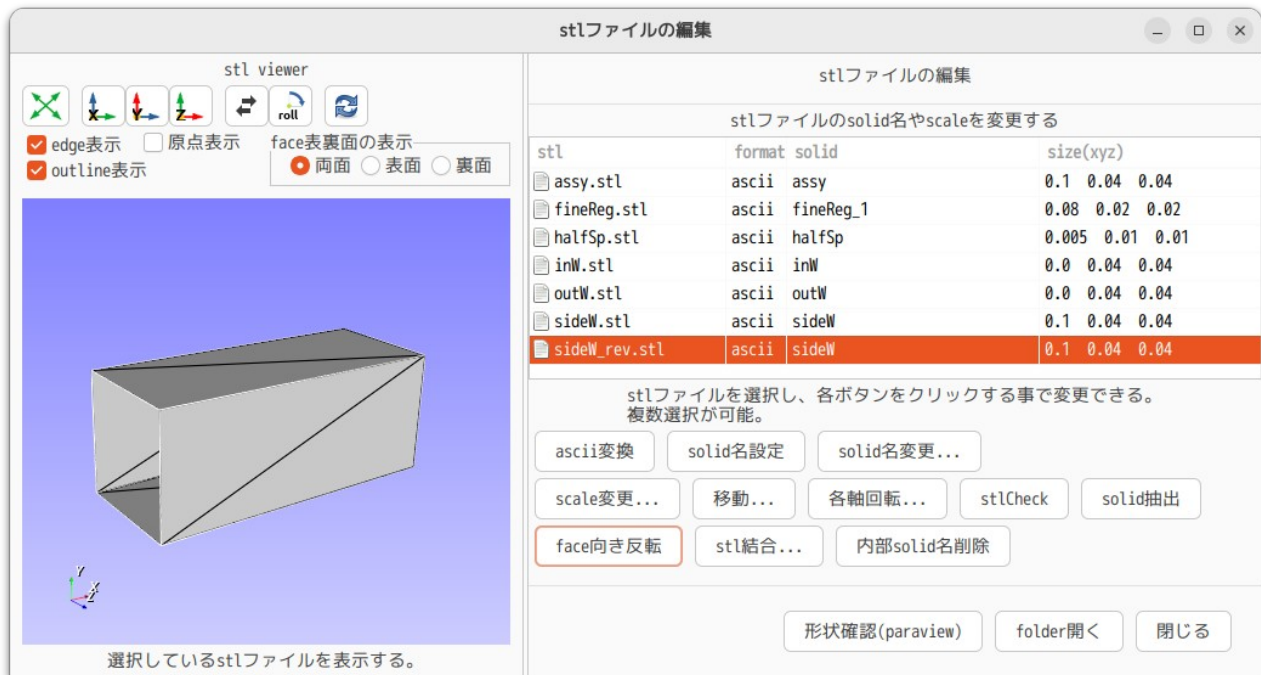
```

1 solid inW
2 facet normal 1 0 0
3   outer loop
4     vertex 0 0.02 0.02
5     vertex 0 -0.02 0.02
6     vertex 0 0.02 -0.02
7   endloop
8 endfacet
:
```

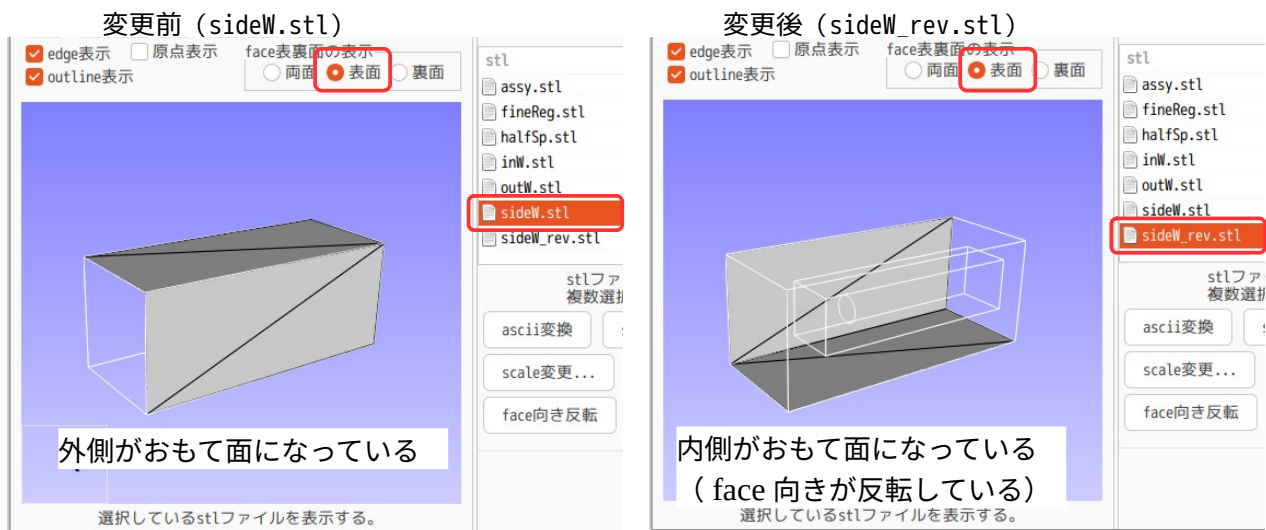
faceの向きを反転させたい場合は、以下の様に反転させたいstlファイルを選択し、「face 向き反転」ボタンをクリックする事により、反転させたstlファイルができて上がる。



faceの向きを反転させた「sideW\_rev.stl」ファイルが追加されている。



faceの向きは、faceの表面のみを表示させる事によって確認できる。（「表面」のラジオボタンをチェックする。）チェックした状態で、確認した結果が以下になる。  
faceの向きが反転している事が確認できる。



#### 9-1-1-10. 移動

stl を XYZ 各軸方向に平行移動させたい時には、これを用いる。

fineReg.stl, halfSp.stl, inW.stl, outW.stl, sideW.stl の5ヶのファイルをまとめてX軸方向に「0.02」移動させてみる。  
移動させたいstlを選択し、「移動...」ボタンをクリックして、X軸方向の移動量「0.02」を入力する。



これにより、選択した5ヶのファイルがX軸方向に0.02移動する。  
viewerには、原点を通る各軸を表示させているので、移動後は、stlが原点から離れており、X軸方向に移動した事が確認できる。



#### 9-1-1-11. 回転

stlを各軸回りに回転させたいときには、これを用いる。

前項と同じstlをZ軸周りに90°回転させてみる。  
回転させたいstlを選択後、「各軸回転...」をクリックして、Z軸周りに「90」を入力して、回転させる。



下図がZ軸周りに90°回転した結果になる。  
原点に対し、90°回転していることが確認できる。



#### 9-1-1-12. stl ファイル check

stl ファイルの内容をチェックする。  
チェックする内容は、「stlCheck」と「solid 抽出」があり、各々以下をチェックしている。

<stlCheck>  
 face の重複定義 (厚さ「0」の solid)  
 face 向きの不揃い



## &lt;solid 抽出&gt;

face の重複定義 (厚さ「0」の solid)

face 向きの不揃い

隣接 face を持っていない face を削除 (離れ小島になっている face を削除)

MeshMixer 等を使って、stl ファイルの編集を行っている時、stl の face 向きが不揃いになったり、solid に余分な face が追加 (隣接 face を持っていない face が追加) される事がある。  
 この様な stl を使ってメッシュを作成しようとするとエラーが発生してしまう。

この為、stl の内容をチェック修正する為に「stlCheck」「solid 抽出」を設定している。



## 9-1-1-13. ポップアップメニュー

stl ファイルのリスト上で右クリックして、以下の様にポップアップメニューを開く事ができ、ここでコピーや削除等のファイル操作を行う事ができる。



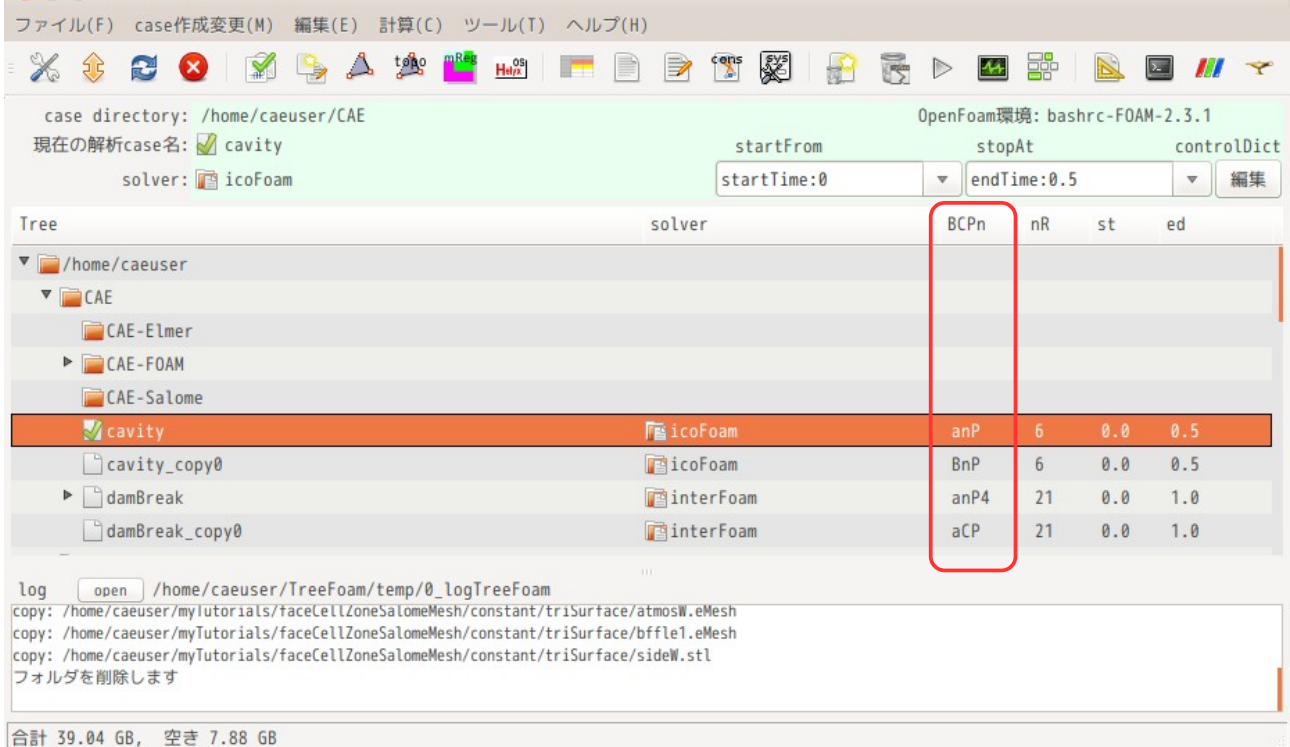
## 9-1-2. binary 形式ファイルの扱い方

TreeFoam 上で binary 形式の file を扱う事ができる。file の書式が ascii なのか binary なのかは、TreeFoam 上の「BCPn」欄から判断できる。下図参照。

BCPn の最初の文字が「B」の場合は binary で、「a」の場合は ascii になる。  
 従って、下図の「cavity」は ascii で、「cavity\_copy0」は binary になる。



TreeFoam\_2.25-150308 (0)



OpenFOAM の binary 形式は、8-1-5-1 項に示している様に、ファイル上部の FoamFile 部は、必ず ascii 形式

また、そのファイル内に binary データがどのように組み込まれているかは、foamFile 内の class によって

```
/*-----* C++ *-----*/
```

```

Field
Operation
And
Manipulation
OpenFOAM: The Open Source CFD Toolbox
Version: 2.3.0
Web: www.OpenFOAM.org
FoamFile
{
    version      2.0;
    format       binary; ← この内容で ascii か binary かを判断
    class        volVectorField;
    location     "0.1";
    object        U;
}

```

区分	class	読み込み形式	binary 部検出方法
----	-------	--------	--------------

区分	class	読み込み形式	binary 部検出方法
polyMesh	labellist	list 形式	例 : 9(*****)

	faceList		binary 部 ascii 文字の "(" を検索して、 その個数分の binary を取得。
	faceCompactList		
	refinementHistory		
	cellSet		
	faceSet		
	pointSet		
	regIOobject	field 形式	例 : List<scalar> 9(*****); binary 部  ascii 文字の "List<" を検索して、 その個数分の binary を取得
	cellZoneList		
	faceZoneList		
	pointZoneList		
	vectorField		
	polyBoundaryMesh		
field	volScalarField		
	volVectorField		
	volSymmTensorField		
	volTensorField		
	surfaceScalarField		
	surfaceVectorField		
	pointScalarField		
	pointVectorField		
その他	***List	list 形式	-
	以外	field 形式	-

binary データを検出した後は、そのデータの変数 (vector 等) とその組数を確認した上で、その個数分の binary データを読み込むことになる。変数と byte 数は、以下の関係にある。

変数	type	byte 数
scalar	double	1 x 8 byte
vector	double	3 x 8
symmTensor	double	6 x 8
tensor	double	9 x 8
label	int	1 x 4
faceList	int	1 x 4
bool	bool	1 x 1

binary データの変数型と個数が判断できれば、これを読み込み ascii 文字に変換できる。

そのファイルの読み込みに当たっては、ファイル内に ascii と binary が混在しているので、ascii 部と binary データ部を分けて読み込み、binary データ部は、予め定められた個数分の binary データを ascii 文字に変換して、ascii 部に挿入し、ファイル全体として ascii 文字ファイルを完成させる。完成した ascii 文字ファイルを「TreeFoam/temp」フォルダに一時保管し、これを editor で開く方法をとっている。

予め定められた ascii 変換する個数は、gridEditor 側で設定されている個数になる。8-1-5-2 項参照。

以下は、binary 形式の field を editor で開いた状態になる。  
この内容は、tutorials の「multuRegionHeater」の書式を binary に変更して「./Allrun」を実行し、その内容を editor で確認した結果になる。このファイルには、binary 部が 2 箇所ある。それぞれの箇所に binary データのインデックスを付加し、binary データが後で挿入できる様に設定している。

また、元々の ascii データ部は編集が可能だが、binary 部 (以下の例では、「List<vector>」から「...U.0...」のインデックスまで) は、編集できない。この部分を書き換えてしまうと、binary データを挿入し直す事ができなくなってしまう。

```
/*-----*- C++ -*-----*\
```

```
(
(0.055683056417 3.7381459595e-05 0.000307644111694)
(0.0911708959769 0.000217452513024 0.000556139881553)
(0.0906610051146 -0.000522416709669 0.00053342825514)
(0.0557214369188 -0.000168090767077 0.000199110902365)
(0.0793714677545 7.27172801494e-05 0.000780484945166)
(0.125751195706 0.000279342072753 0.00102485007502)

```

```

(0.125120231553 -0.000664690666303 0.000938273723701)
(0.0798125947138 -0.000261577718551 0.000325489467223)
(0.0841102455769 -1.76933859644e-05 0.000808970812252)
(0.132447694997 0.000216108863418 0.0005015637297741)
(0.131589725751 -0.000593147136692 0.00037334      binary データ部 (40 個の vector データ)
(0.0845429672062 -0.000202325527195 -0.000101
(0.0841930124845 0.000311827214104 0.00115406
(0.13218551605 0.000796856416352 -2.48674522348e-05)
(0.13252281893 -0.000192590341716 -0.000224296117474)
(0.0851048968908 -6.74618554409e-05 -0.000394807987696)
(0.0829037483249 0.00126062269493 0.000678843121059)
(0.128968343445 0.00242704533856 -0.000244173483344)
(0.132895934354 0.000908346978078 -0.000423933698924)
(0.0856451980135 0.000302336591309 -0.000375218256195)
...U.1...
);
    }
    minZ
    {
        type          fixedValue;
        value          uniform (0 0 0);
    }
    maxZ
    {
        type          fixedValue;
        value          uniform (0 0 0);
    }
    topAir_to_rightSolid
    {
        type          fixedValue;
        value          uniform (0 0 0);
    }
    topAir_to_heater
    {
        type          fixedValue;
        value          uniform (0 0 0);
    }
    topAir_to_leftSolid
    {
        type          fixedValue;
        value          uniform (0 0 0);
    }
}

// *****

```

上記の内容を修正し、保存する場合は、ascii 変換されている binary データ部に、元の binary データを挿入し直して、一旦「TreeFoam/temp」内に保存し、この後で元の場所に戻している。



この様に、編集後保存して editor を閉じた後、前記した後処理が必要になってくるので、configTreeFoam 内の editor の設定は、2-3 11) 項に示してある様に、editor を裏で起動しない様に、ubuntu の場合、editor「gedit」をオプション「--standalone」の設定で起動する設定にしている。editor を閉じた後、前記した後処理 (binary データを挿入し直して保存) を行っている。(editor を閉じないと、修正内容が反映されない。)

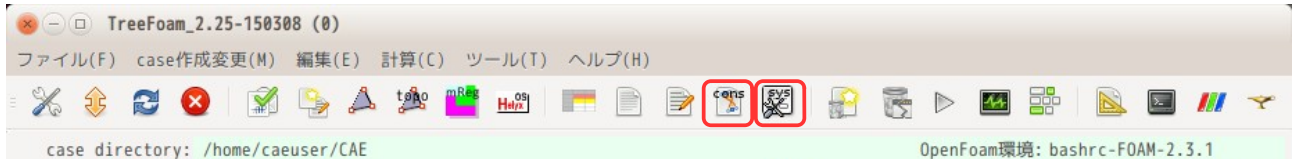
#### 9-1-2-2. binary ファイルを editor で確認する方法

binary ファイルを editor で開いて編集する方法は、以下の 3 種類の方法がある。

- 1) gridEditor から editor を開く  
8-1-5 項を参照。(T, U, p 等の field に限る)
- 2) properties または dictionary の編集画面で editor を開く  
解析 case 内の全ての binary ファイルを editor で開くことができる。
- 3) 端末から editor を開く  
解析 case 内の全ての binary ファイルを editor で開くことができる。

1)項は、gridEditorを使う為、扱える binary ファイルは、case 内の field ファイルに限られるが、gridEditor 上で field 名をダブルクリックするだけで開くので、容易そのファイルを開く事ができる。使い方は、8-1-5 項を参照。

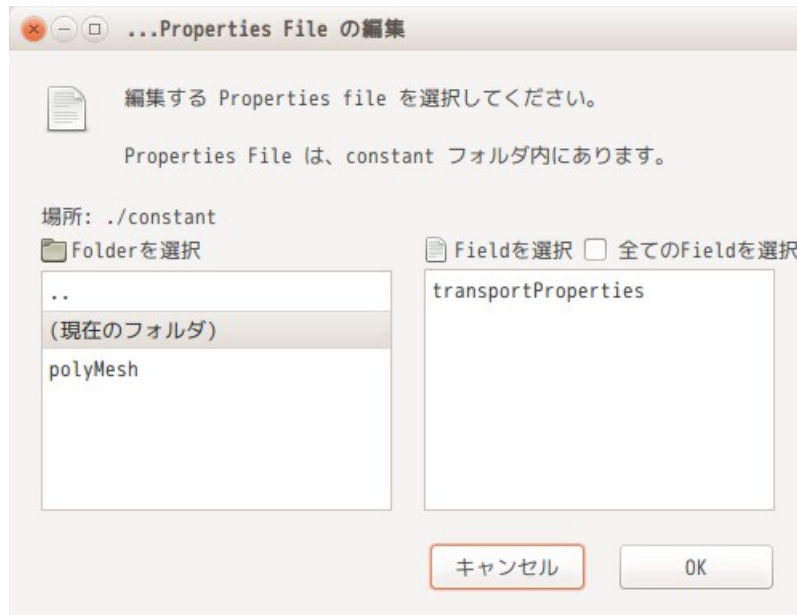
2)項は、その case 内の全ての binary ファイルが扱える。  
その使い方は、TreeFoam 上の  ボタン、または  ボタンをクリックする。



これにより以下の画面が表示されるので、ここからファイルを選択して、editor で開くことになる。

この画面上で、「folder を選択」のリストボックス中のフォルダ名をダブルクリックすると、そのフォルダに移動し、右側のリストボックス内にファイル名のリストが表示される。また、フォルダのリストボックス内の「..」をダブルクリックすると親フォルダに移動できる。この為、case 内の全てのフォルダに移動でき、全てのファイルを editor で確認できることになる。

ファイルを editor で開くときは、ファイル名をダブルクリックするか、ファイルを選択して「OK」ボタンをクリックする事で開くことができる。

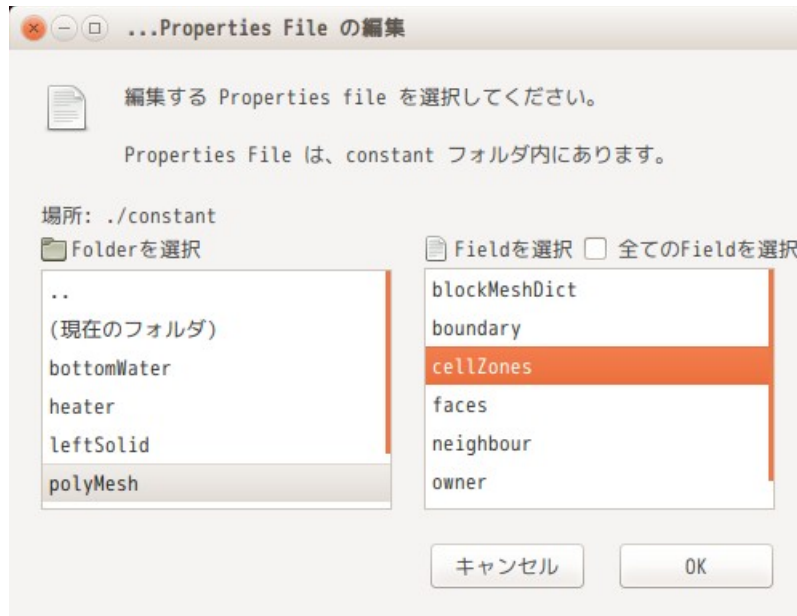


ここから実際に binary ファイルを開いてみる。

editor で開くファイルは、tutorials の「multiRegionHeater」case の controlDict 内の writeFormat を「binary」に書き換え、「./Allrun」を実行して、binary の case を完成させた後、case 内の「constant/polyMesh/cellZones」の binary ファイルを開いてみる。

下図の「properties file の編集」画面上で、「polyMesh」フォルダと「cellZones」ファイルを以下の様に選択し、「OK」ボタンをクリックするかダブルクリックすると、editor が起動して binary 形式のファイルを開く事ができる。





以下が、editor で開いた結果になる。本来であれば、binary データが存在しているので editor で開くことはできないが、binary データを ascii 文字に変換することで editor で内容が確認できる。

ascii 文字データ部は編集できるので、editor 上で cellZone 名を変更する事もできる。ただし、binary 部（以下の例では、「List<label>」から「...cellZones.0...」のインデックスまで）は、編集できない。

```

/*-----* 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
OpenFOAM: The Open Source CFD Toolbox
Version: 2.3.1
Web: www.OpenFOAM.org
/*-----*/

FoamFile
{
    version      2.0;
    format       binary;
    class        regIOobject;
    location     "constant/polyMesh";
    object       cellZones;
}
// *****

5
(
    heater
    {
        type cellZone;
        cellLabels List<label>
        80
        (
            1273
            1274
            763
            1275
            163
            164
            165
            166
            1603
            1604
            1605
            1606
            1963
            binary データ部（80 個の label データ）
            binary を ascii 文字に変換して表示。
            変換する行数は、予め定められた行数分
            を ascii 変換する。
            ascii 変換行数は、gridEditor 側で
            決定されている。（8-1-5-2 項参照）
        )
    }
)

```

```
1964
1573
1965
1574
1966
1063
1575
...cellZones.0...    binary データのインデックス
);
}

leftSolid
{
    type cellZone;
    cellLabels      List<label>
130
(
457
458
459
160
161
162
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
760
761
762
2850
...cellZones.1...
);
}

rightSolid
{
    type cellZone;
    cellLabels      List<label>
130
(
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
167
168
169
170
171
172
173
...cellZones.2...
);
```

```
}

topAir
{
    type cellZone;
    cellLabels    List<label>
1200
(
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
...cellZones.3...
);
}

bottomWater
{
    type cellZone;
    cellLabels    List<label>
1460
(
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
...cellZones.4...
);
}

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

端末を起動して、コマンド入力で binary 形式のファイルを開く事もできる。  
この場合は、TreeFoam 上から端末を起動して、以下のコマンドを入力する事で、binary 形式のファイルを開く事ができる。

```
$ editFoamFile.py constant/polyMesh/cellZones
```

上記入力で、前記した結果と同じ内容が editor で確認できる。  
また、以下の様にオプションを追加することで、binary データの表示個数を 5 行に変更する事もできるので、表示内容をシンプルにする事ができる。

```
$ editFoamFile.py -n 5 constant/polyMesh/cellZones
```

このコマンドのオプションの詳細は、-h オプションで確認する事ができる。以下はその help の内容になる。

```
----- editFoamFile.py の使い方-----
OpenFOAM のファイルを編集する。
ファイルが、gzip、binary でも editor で編集できる。
データ部の行数を省略して表示するので editor の動作が軽くなるが
データ部の編集はできない。

使い方
editFoamFile.py [option] <file0> <file1>...
[option]
  -t      temporary folder の指定
          デフォルトは、「~/TreeFoam/temp」
  -n      データ部の表示行数を設定
          デフォルトは、「~/TreeFoam/data/griEditor_data」内の
          nMaxLinesBinToAscii の設定による。
          全て表示させる場合は、表示行数を「-1」に設定。
  -h      help の表示
```

## 9-2. gridEditor の表示

### 9-2-1. 列 (field) の表示

gridEditor は、各 field の patch 内容が表形式で確認できる為、理解しやすいが、field が多数あると横長の表ができあがり、表の内容が一望できなくなる。

この為、不要な field を非表示したり、表示順を変更したりできる様にしている。

#### 9-2-1-1. field の非表示

tutorials の simpleFoam : pitzDaily を例にとって確認する。  
まず、tutorials の「incompressible/simpleFoam/pitzDaily」の blockMesh を作成し、gridEditor で境界条件を確認した結果が、以下になる。

## TreeFoam 操作マニュアル (TreeFoam-3.31-250809)

gridEditor: pitzDaily\_simpleFoam/0/. (0:1)  
ファイル(F) 編集(E) 表示(V)

	define patch at constant/. (boundary)	U	epsilon	k	nuTilda	nut	p
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -3 0 0 0 0];	[0 2 -2 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 14.855;	uniform 0.375;	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type fixedValue; value uniform 14.855;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 0;	type calculated; value uniform 0;	type zeroGradient;
outlet	type patch;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;	type empty;	type empty;

上図は、全ての field を表示しているが、この内、k, epsilon を非表示にしてみる。  
その方法は、下図のように非表示させたい field (epsilon, k) を選択して、選択列ラベル部、またはセル部を右クリックしてポップアップメニューを表示させ、「選択した field を非表示」を選択する。(下図は、列名部を右クリックした状態。)

gridEditor: pitzDaily\_simpleFoam/0/. (0:1)  
ファイル(F) 編集(E) 表示(V)

	define patch at constant/. (boundary)	U	epsilon	k	nuTilda	nut	p
field type		volVectorField;	volScalarField;	Field;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -3 0 0 0 0];	[0 2 -2 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 14.855;	0.375;	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type fixedValue; value uniform 14.855;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 0;	type calculated; value uniform 0;	type zeroGradient;
outlet	type patch;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;	type empty;	type empty;

ポップアップメニューの「選択した field を非表示」を選択すると、epsilon と k field が下図のように非表示状態となる。非表示設定の場合は、ラベル名の色が濃い青に変わるので、現在の状態が非表示設定なのかそうでないかが判断できる。



gridEditor: pitzDaily\_simpleFoam/0/. (0:3)

ファイル(F) 編集(E) 表示(V)

	define patch (boundary)	U	nuTilda	nut	p
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type fixedValue; value uniform 0;	type calculated; value uniform 0;	type zeroGradient;
outlet	type patch;	type zeroGradient;	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;

尚、一度非表示設定を行うと、その状態が firstTime フォルダ内に隠しファイル「.displayField」が作成されるので、次回起動時には、これを読み込み起動するので非表示設定が反映された状態で起動する。

また、非表示設定された状態を元に戻す（全表示）為には、ポップアップメニューから、「全表示/非表示 field の切替え」を選択する。これにより、元の状態にもどる。

gridEditor: pitzDaily\_simpleFoam/0/. (0:2)

ファイル(F) 編集(E) 表示(V)

	define patch at constant/. (boundary)	U	nuTilda	nut	p
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -2 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type fixedValue; value uniform 0;	type calculated; value uniform 0;	type zeroGradient;
outlet	type patch;	type zeroGradient;	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;

全表示/非表示fieldの切替え  
選択したfieldを非表示  
field表示順変更  
fieldコピー  
field貼付(挿入)  
field名変更  
field削除

### 9-2-1-2. field の表示順を変更

前項では、field の非表示設定を行ったが、ここでは、field の表示順を変更してみる。

下図が、field を全表示させている状態だが、この表示順を「U, p, k, epsilon」とし、これ以外は非表示設定してみる。

	define patch at constant/. (boundary)	U	epsilon	k	nuTilda	nut	p
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -3 0 0 0 0];	[0 2 -2 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 14.855;	uniform 0.375;	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type fixedValue; value uniform 14.855;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 0;	type calculated; value uniform 0;	type zeroGradient;
outlet	type patch;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqRWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqRWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;	type empty;	type empty;

その方法は、下図の様に、列ラベル部を右クリックしてポップアップメニューを表示させ、「fieldの表示順変更」を選択する。

	define patch at constant/. (boundary)	U	epsilon	k	nuTilda	nut	p
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -3 0 0 0 0];	[0 2 -2 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 14.855;	uniform 0.375;	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type fixedValue; value uniform 14.855;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 0;	type calculated; value uniform 0;	type zeroGradient;
outlet	type patch;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqRWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type epsilonWallFunction; value uniform 14.855;	type kqRWallFunction; value uniform 0.375;	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;	type empty;	type empty;

この後、以下の画面が表示される。この画面上で、隠す field (nuTilda, nut) と、表示する field の表示順を設定する。

Fieldの非表示、表示の設定

非表示、表示fieldの設定と、Fieldの表示順を設定します  
Fieldを選択してボタンをクリックして設定する

隠すField

表示設定  
表示>>  
<<隠す

表示するField

表示順  
↑先  
↓後

キャンセル

OK

この画面を、最終的に以下の様に設定して、「OK」ボタンをクリックする事で、表示する field とその表示順が決定される。



以下がこの設定で、gridEditor を表示させた結果になる。field が「U, p, k, epsilon」の順番で表示されている。

	define patch at constant/. (boundary)	U	p	k	epsilon
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];	[0 2 -2 0 0 0];	[0 2 -3 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;	uniform 0.375;	uniform 14.855;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type zeroGradient;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 14.855;
outlet	type patch;	type zeroGradient;	type fixedValue; value uniform 0;	type zeroGradient;	type zeroGradient;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type kqRWallFunction; value uniform 0.375;	type epsilonWallFunction; value uniform 14.855;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type kqRWallFunction; value uniform 0.375;	type epsilonWallFunction; value uniform 14.855;
frontAndBa	type empty;	type empty;	type empty;	type empty;	type empty;

尚、この設定は、前項と同様に firstTime フォルダ内に「.displayField」の隠しファイルができあがっている。この内容は、以下であり、表示する field 名が表示順に記載されている。次回起動時にもこの設定が反映される。

尚設定を元に戻す（全 field を表示する）には、前項と同様に、ポップアップメニューから、「全表示/非表示 field の切替え」を選択する事で、元の状態にもどる。

## 9-2-2. 行 (patch 名など) の表示

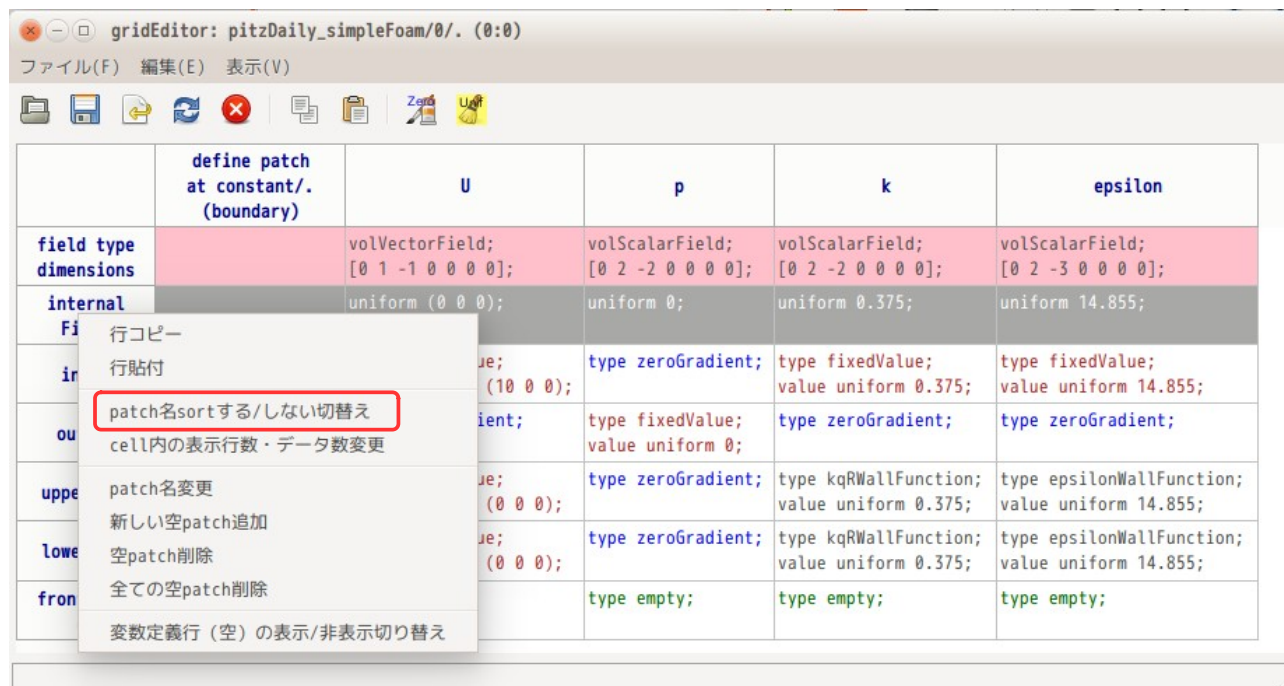
gridEditor の行ラベルについては、patch 名が入るが、この patch 名の表示順は、boundary に記述されている順番で表示される。この表示順を patch 名で sort させて表示させる事ができる。

また、通常は表示されていない変数定義行を表示させる事ができる。field 内で「\$iniTemp」等の変数を定義したい時に、この行を表示させて、この中で変数を定義する。

### 9-2-2-1. patch 名を sort して表示

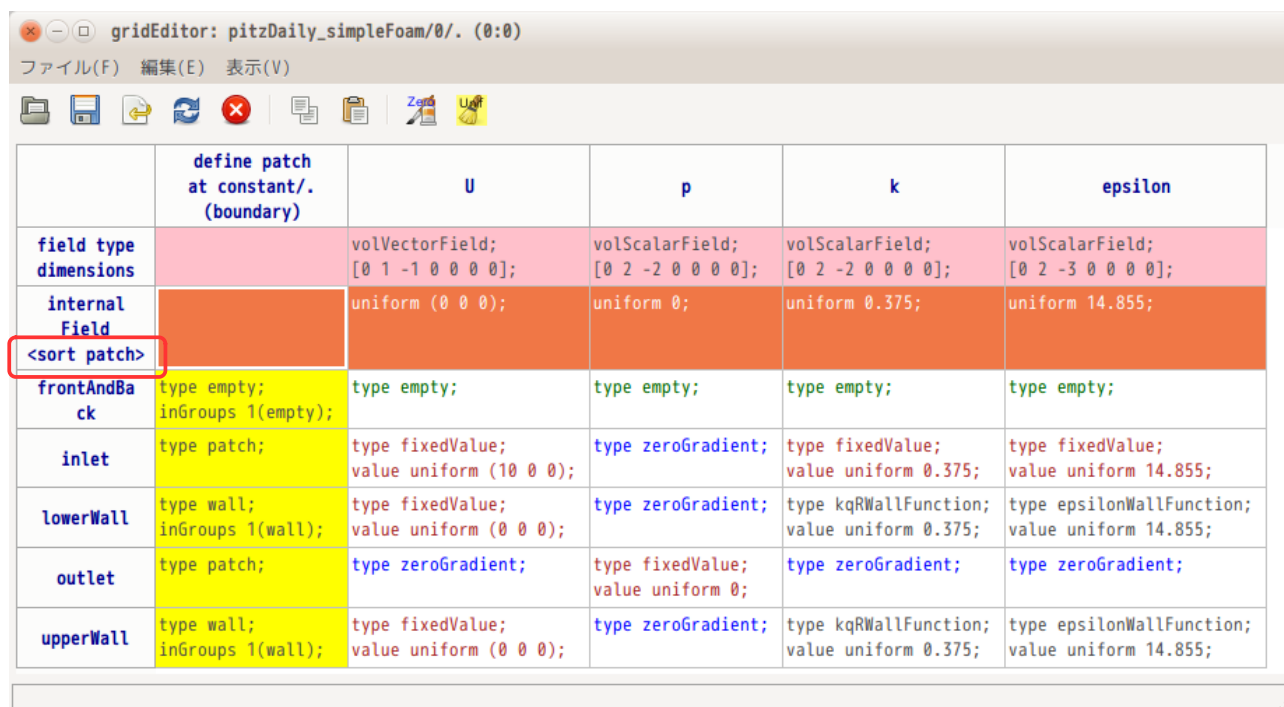
patch 名が多くある場合は、patch 名を sort させて表示させると判りやすくなる。

その方法は、以下の様に、行ラベル名部で右クリックしてポップアップメニューを表示させ、このメニューの「patch 名 sort する/しない切替え」を選択する。この操作により patch 名を sort して表示させる事ができる。



以下が patch 名を sort して表示させた状態になる。尚、patch 名を sort して表示させた状態は、以下の様に、「<sort patch>」が表示される。

また、この内容は、gridEditor 上の表示方法を変更しているのみで、実際の file の内容 (patch の記述順) は、変わらない。



尚、patch 名を sort するかしないかの設定は、「\$TreeFoamUser/data/gridEditor\_data」ファイルに記録される。この為、次回起動時この設定が反映される。



以下は、現在の「gridEditor\_data」の設定内容になる。

```
----- gridEditor_data の内容-----
#
#  gridEditor の設定
#

#patch 名を sort させて表示させる
sortPatchName yes

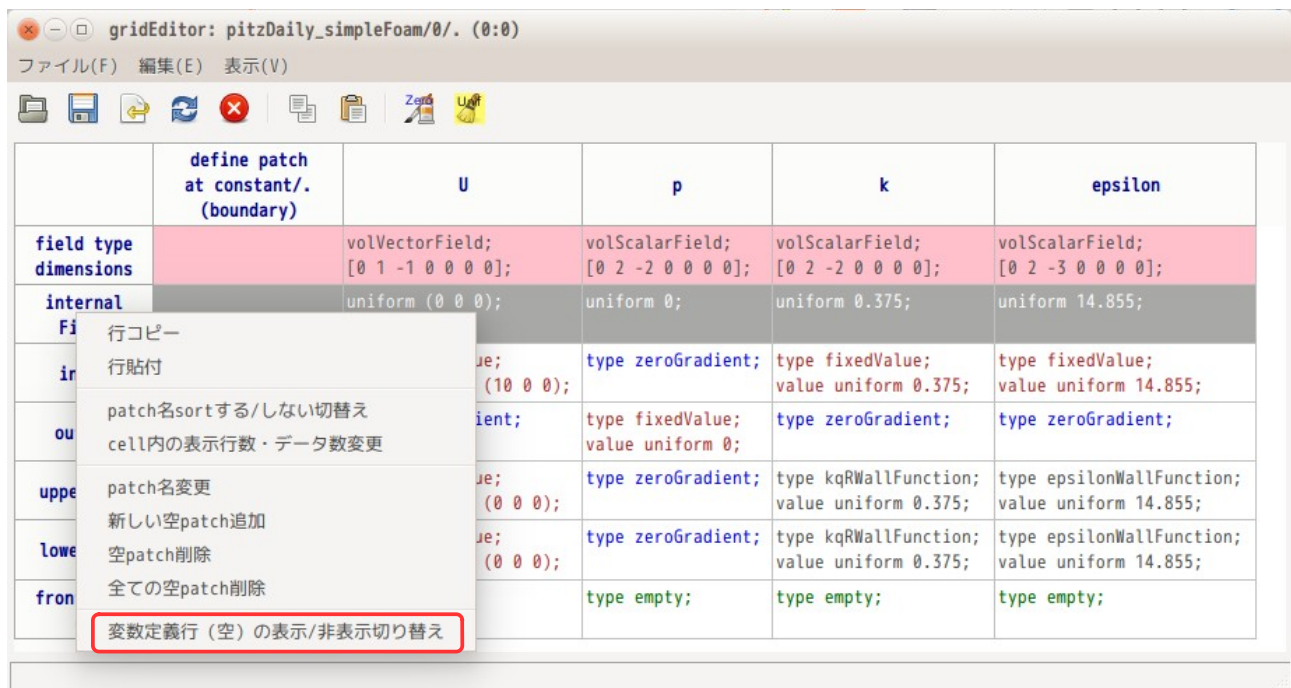
#cell 内の表示行数
maxLinesCellInternal 5
maxLinesCellPatch 10

#binnary→ascii 変換する行数
nMaxLinesBinToAscii 20
-----
```

### 9-2-2-2. 変数行の表示

gridEditor 上では、「\$iniTemp」等の変数を使うことができるが、使うためには変数を定義する必要がある。gridEditor では、変数を定義する場所として変数定義行を準備しているが、default の状態は、これが非表示になっているので、変数を定義することができない。

変数定義行を表示させるためには、以下の様に行ラベル上で右クリックしてポップアップメニューを表示させ、「変数定義行（空）の表示/非表示の切替え」を選択する事で、変数定義行を表示する事ができる。



以下が、変数定義行を表示させた状態になる。変数定義行（水色）が2行が表示されている。

otherNames : boundaryField の外側で定義  
 otherNames (boundary) : boundaryField の中で定義



gridEditor: pitzDaily\_simpleFoam/0/. (0:0)

ファイル(F) 編集(E) 表示(V)

	define patch at constant/ (boundary)	U	p	k	epsilon
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];	[0 2 -2 0 0 0];	[0 2 -3 0 0 0];
otherNames					
internal Field <sort patch>		uniform (0 0 0);	uniform 0;	uniform 0.375;	uniform 14.855;
otherNames (boundary)					
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type zeroGradient;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 14.855;
lowerWall	type wall;	type fixedValue;	type zeroGradient;	type kqRWallFunction;	type epsilonWallFunction;

尚、変数が定義されている field があると、以下のような表示となり、変数定義行を消す事ができなくなる。  
(「iniPress 0;」を定義したため、変数定義行が表示された状態になる。)

gridEditor: pitzDaily\_simpleFoam/0/. (0:0)

ファイル(F) 編集(E) 表示(V)

	define patch at constant/ (boundary)	U	p	k	epsilon
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];	[0 2 -2 0 0 0];	[0 2 -3 0 0 0];
internal Field <sort patch>		uniform (0 0 0);	uniform 0;	uniform 0.375;	uniform 14.855;
otherNames (boundary)			iniPress 0;		
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type zeroGradient;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 14.855;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type kqRWallFunction; value uniform 0.375;	type epsilonWallFunction; value uniform 14.855;
outlet	type patch;	type zeroGradient;	type fixedValue; value uniform 0;	type zeroGradient;	type zeroGradient;
upperWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type kqRWallFunction; value uniform 0.375;	type epsilonWallFunction; value uniform 14.855;

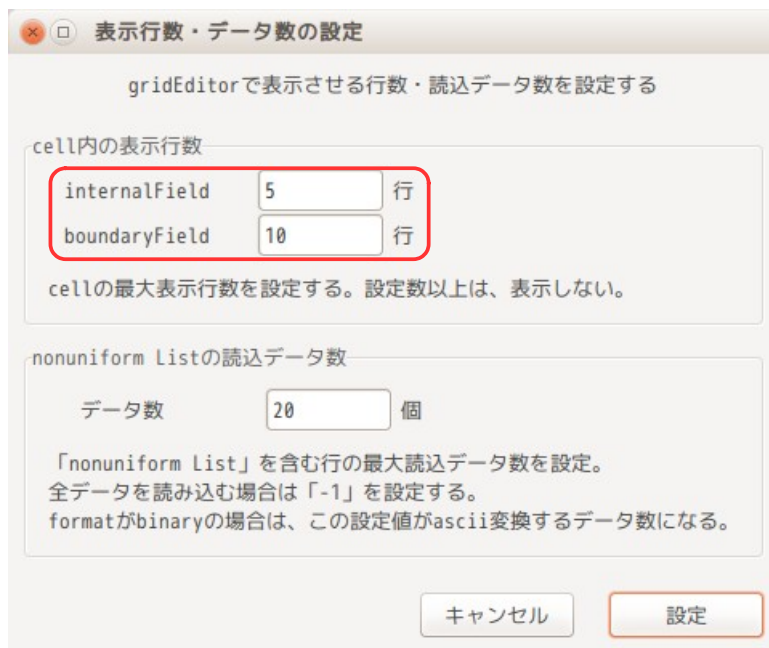
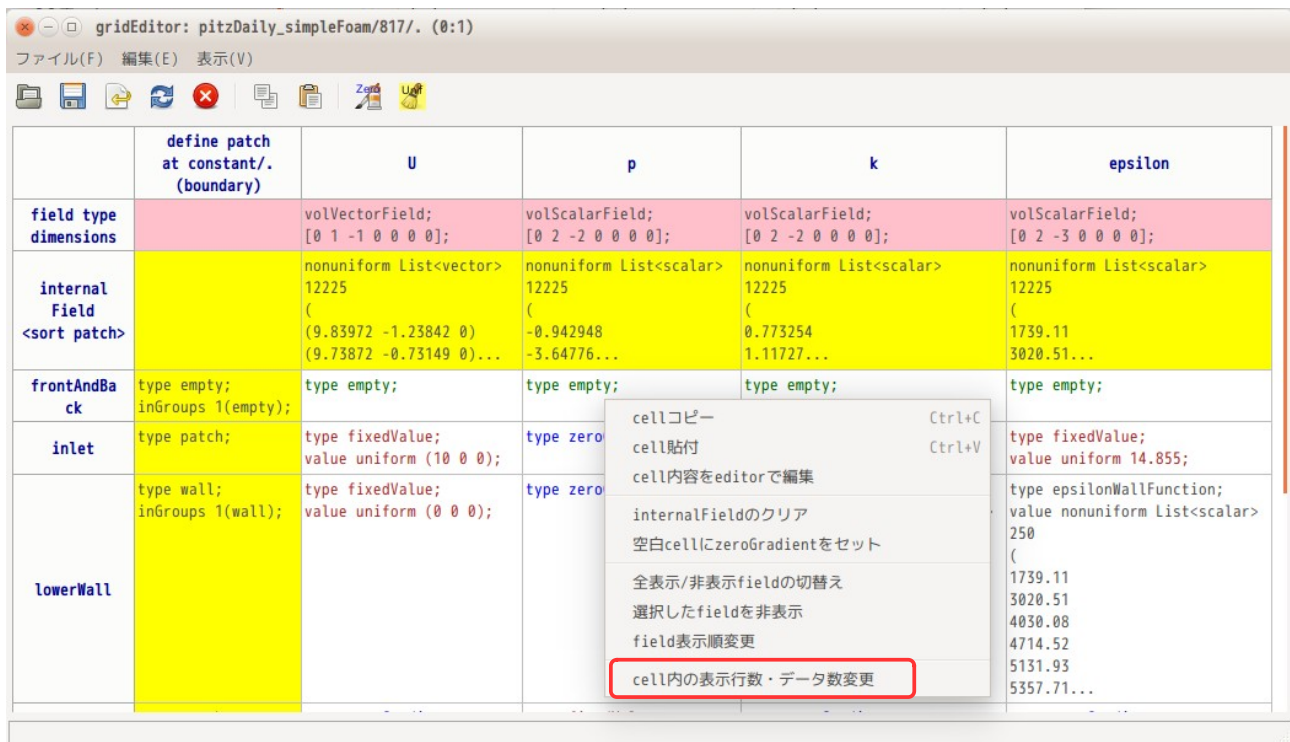
### 9-2-3. セル (patch 内容など) の表示

gridEditor は、表形式で internalField や patch データを表示している。field に計算結果が入ると、internalField や patch 内には、膨大なデータが入っており、これを表形式で表示させる事はできない。

この為、表のセルに表示させる行数の制限を設けており、設定された行数以上は表示させない設定になっている。

この行数の設定は、gridEditor のセル部分を右クリックしてポップアップメニューを表示させ、「cell 内

の表示行数・データ数変更」を選択して、現れた「表示行数・データ数の設定」画面上で、設定する。



上記画面上で、internalFieldは5行、boundaryFieldは10行に設定されている。この行数を両方共6行に設定して、gridEditorを表示させた結果が以下になる。6行に変更されて表示されている。

gridEditor: pitzDaily\_simpleFoam/817/. (0:1)

ファイル(F) 編集(E) 表示(V)

	define patch at constant/. (boundary)	U	p	k	epsilon
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 2 -2 0 0 0];	[0 2 -2 0 0 0];	[0 2 -3 0 0 0];
internal Field <sort patch>		nonuniform List<vector> 12225 ( (9.83972 -1.23842 0) (9.73872 -0.73149 0) (9.53698 -0.467335 0)...	nonuniform List<scalar> 12225 ( -0.942948 -3.64776 -4.49568...	nonuniform List<scalar> 12225 ( 0.773254 1.11727 1.35409...	nonuniform List<scalar> 12225 ( 1739.11 3020.51 4030.08...
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty;
inlet	type patch;	type fixedValue; value uniform (10 0 0);	type zeroGradient;	type fixedValue; value uniform 0.375;	type fixedValue; value uniform 14.855;
lowerWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type kqRWallFunction; value nonuniform List<scalar> 250 ( 0.773254 1.11727...	type epsilonWallFunction; value nonuniform List<scalar> 250 ( 1739.11 3020.51...
outlet	type patch;	type zeroGradient;	type fixedValue; value uniform 0;	type zeroGradient;	type zeroGradient;
	type wall;	type fixedValue;	type zeroGradient;	type kqRWallFunction;	type epsilonWallFunction;

この設定は、「TreeFoam/data/grideditor\_data」ファイルに保存されるので、次回起動時にもこの設定が反映される。

以下が「gridEditor\_data」の内容になる

----- gridEditor\_data の内容 -----

```
#
#   gridEditor の設定
#

#patch 名を sort させて表示させる
sortPatchName yes

#cell 内の表示行数
maxLinesCellInternal 6
maxLinesCellPatch 6    }   cell 内の表示行数設定内容

#binary→ascii 変換する行数
nMaxLinesBinToAscii 20
-----
```

#### 9-2-4. 空 patch (face 数が「0」の patch) の作成、削除

モデル内部に patch を作る場合には、face の数が「0」の空 patch を予め作成しておく必要があった。(OPENFOAM-2.2 以降は空 patch を作成しなくても内部 patch が作成できるようになった。)

また、snappyHexMesh でメッシュを作成した時などは、空 patch が多数発生してしまうことがある。このような場合に、次項以下にあるように gridEditor を使うことで、容易に空 patch を作成したり、削除する事ができる。

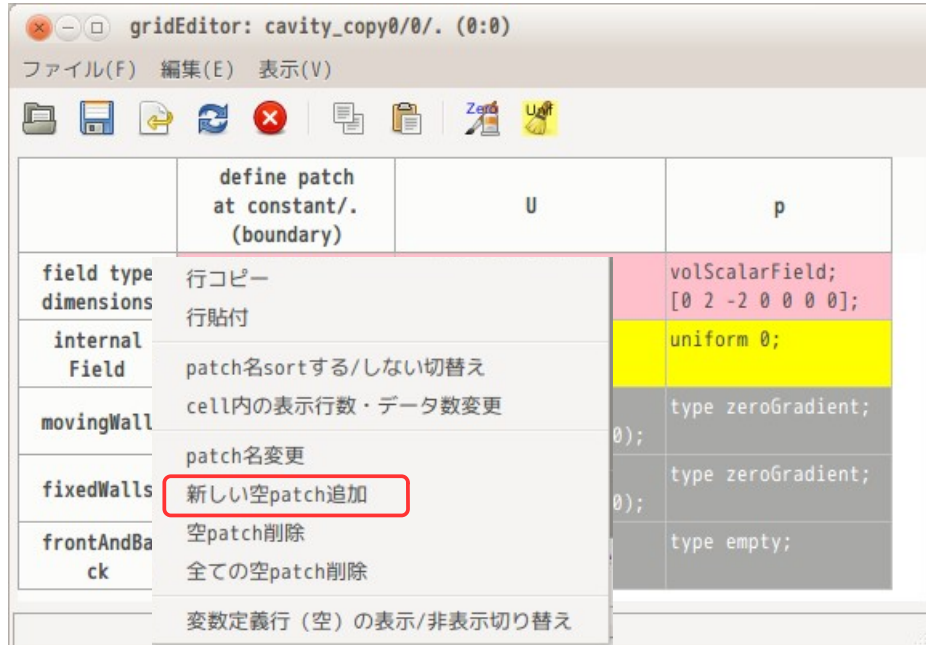
##### 9-2-4-1. 空 patch の作成方法

tutorials の cavity を使って、空 patch を追加してみる。空 patch を追加する為には、gridEditor 上で patch 名部 (行ラベル部) を選択した後、右クリックしてポップアップメニューを表示させ、「新しい空 patch 追加」を選択する事によって、1 枚の空 patch を追加する事ができる。

複数の空 patch を追加する場合は、複数行を選択して、ポップアップメニューを表示させる事によって、複数行（選択した行数分）の空 patch を追加する事ができる。

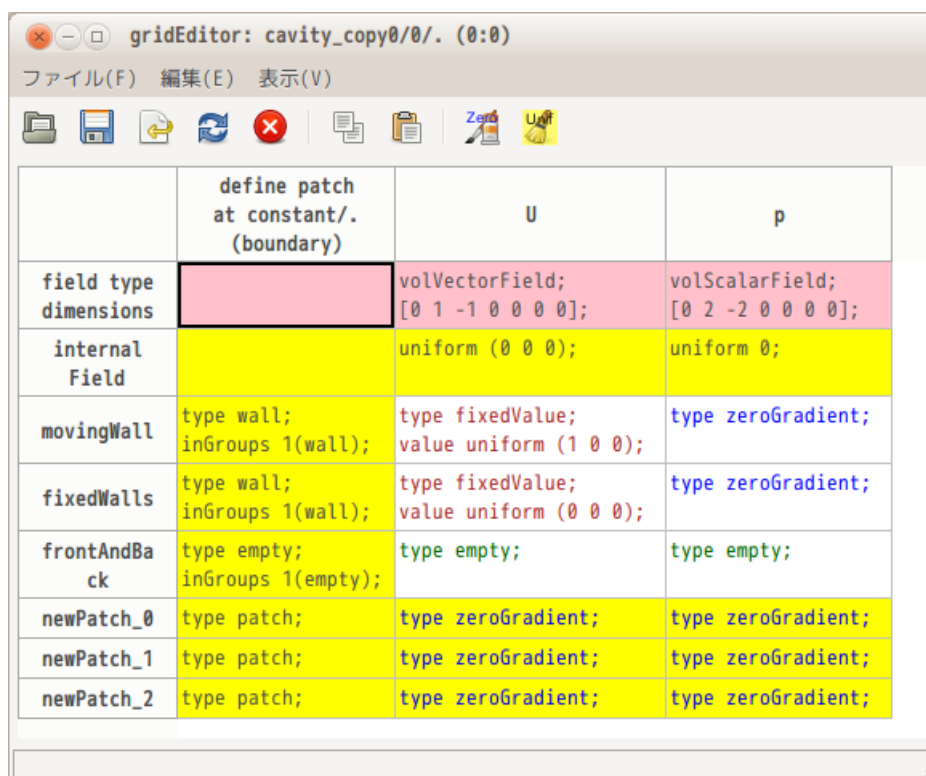
以下の例は、3 行の空 patch を追加する例になる。

下図の様に 3 行選択した上で、patch 名部（行ラベル部）を右クリックしてポップアップメニューを表示させ、「新しい空 patch 追加」を選択すると、3 行の空 patch が追加される。



以下は、3 行の空 patch を追加した状態になる。空 patch（face の数が「0」のパッチ）は、黄色で表示される。また、追加と同時に各 field の boundaryField の整合性も取るので、空 patch の boundaryField には、「zeroGradient」が設定される。

さらに追加する場合は、引き続き、追加する行数分の行を選択して同様な操作を行う事により、追加できる。



また、patch 名は「newPatch\_0」の様な patch 名が付加されている。この patch 名を変更する場合は、変更したい patch 名部をダブルクリックする事によって、patch 名が修正できる。

#### 9-2-4-2. 空 patch の削除


空 patch を削除する場合は、削除したい空 patch を選択して、前項と同様にポップアップメニューを表示させ、「空 patch 削除」を選択すると、選択行の空 patch が削除される。

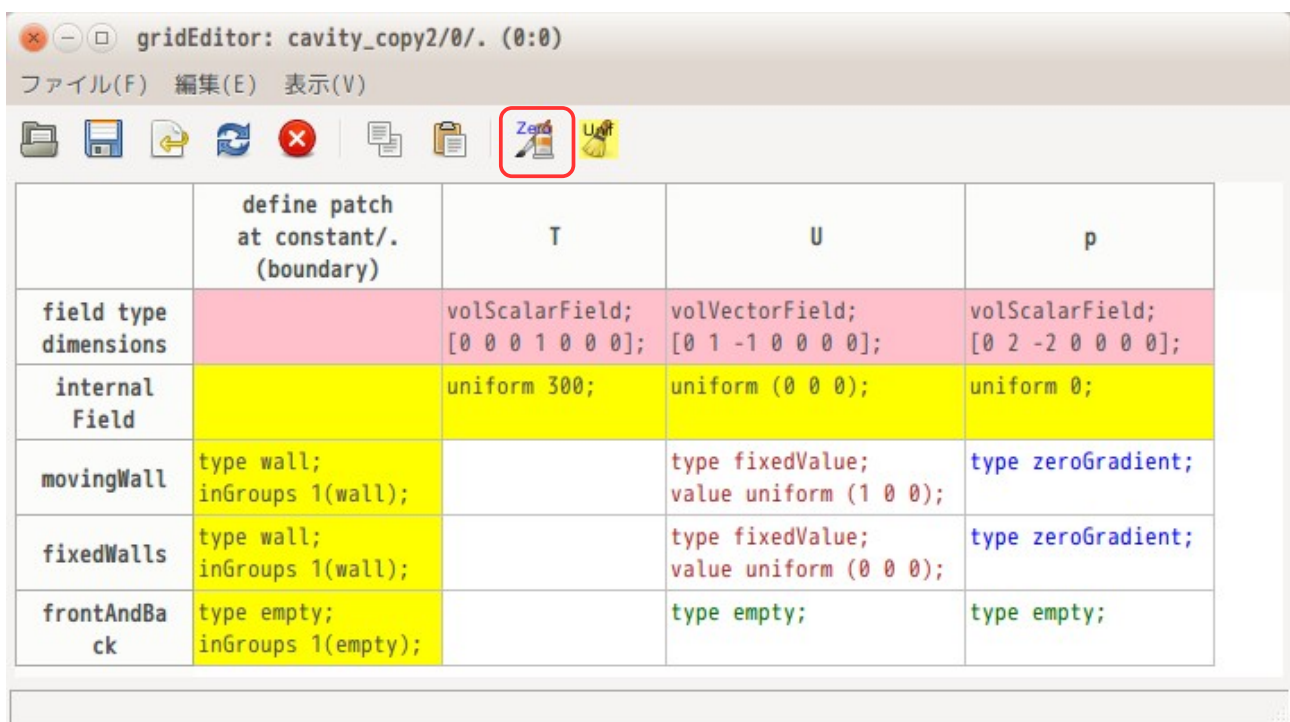
全ての空 patch を削除する場合は、ポップアップメニューから「全ての空 patch 削除」を選択すると、全ての空 patch を削除してくれる。

#### 9-2-5. 空白セルを zeroGradient で埋める

モデルが異なる他の case から field をコピーすると、boundary の整合が取れなくなる。このような field を gridEditor で読み込むと、boundary の整合が取れていない patch の内容が空白で表示される。

以下は、boundary の整合が取れていない field を読み込んだ状態になる。T field が boundary の整合が取れていない。

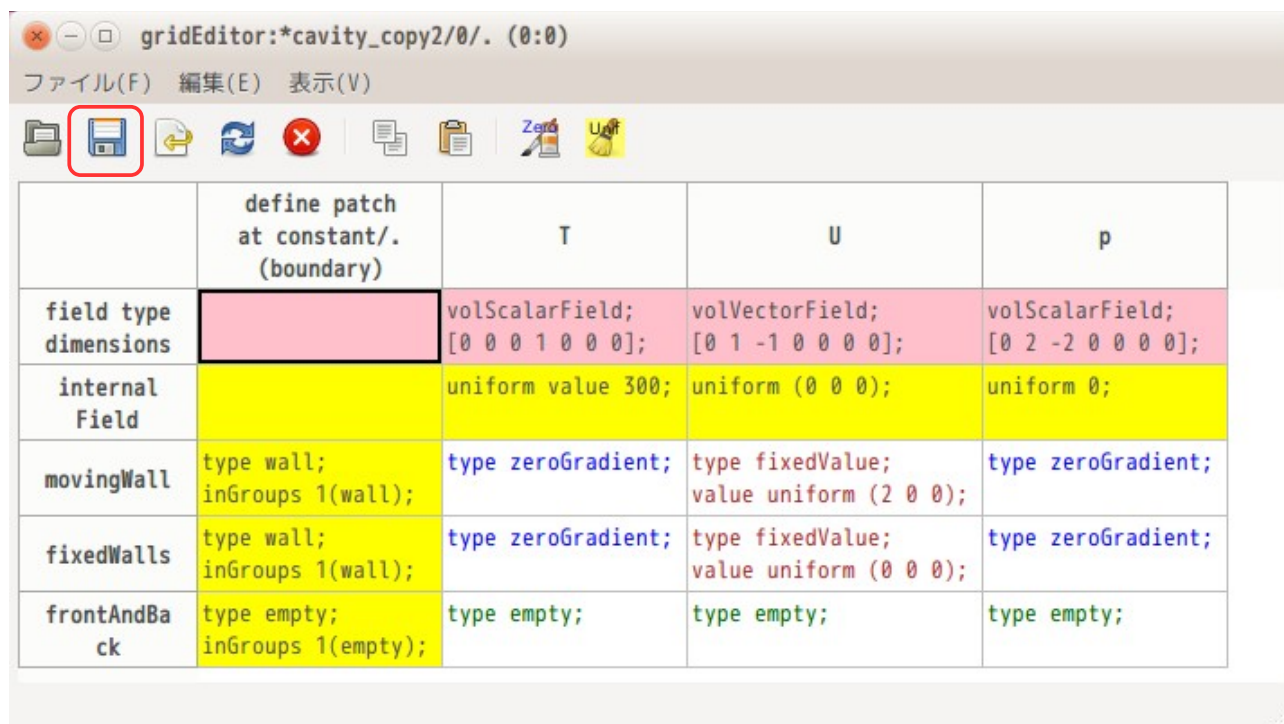
この状態で、 ボタンをクリックすると、空白セルを「zeroGradient」で埋める事ができる。（正確には、boundary の patchType に応じた内容で埋める。）これにより、boundary の整合が取れる。




	define patch at constant/. (boundary)	T	U	p
field type		volScalarField;	volVectorField;	volScalarField;
dimensions		[0 0 0 1 0 0 0];	[0 1 -1 0 0 0 0];	[0 2 -2 0 0 0 0];
internal Field		uniform 300;	uniform (0 0 0);	uniform 0;
movingWall	type wall; inGroups 1(wall);		type fixedValue; value uniform (1 0 0);	type zeroGradient;
fixedWalls	type wall; inGroups 1(wall);		type fixedValue; value uniform (0 0 0);	type zeroGradient;
frontAndBack	type empty; inGroups 1(empty);		type empty;	type empty;

以下は、 ボタンをクリックした状態になる。空白セルが boundary の patchType に応じた cell 内容で埋められている。（今回の場合は、「zeroGradient」と「empty」で埋められている。）





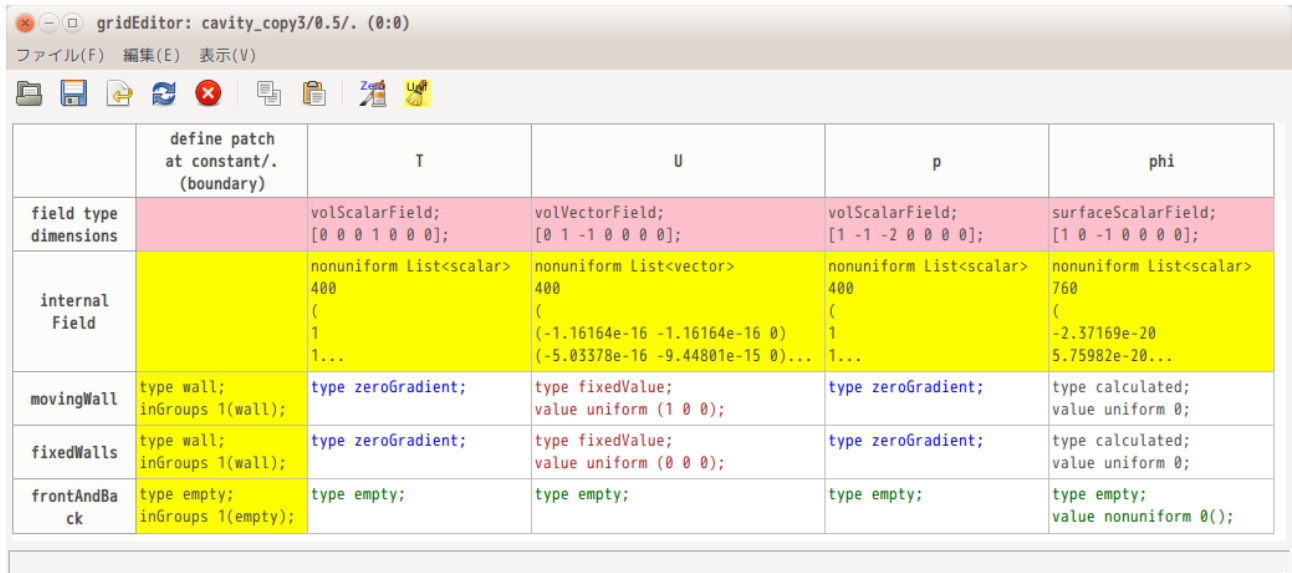
今の状態は、gridEditor 上のみで整合が取れている状態なので、 ボタンをクリックして、保存する必要がある。（保存して最終的に boundary の整合が取れた状態になる。）

### 9-2-6. internalFieldをクリア

計算結果が入っている timeFolder を gridEditor で開くと、internalField 内には、nonuniform 形式 (List 形式) で膨大なデータが入っている。

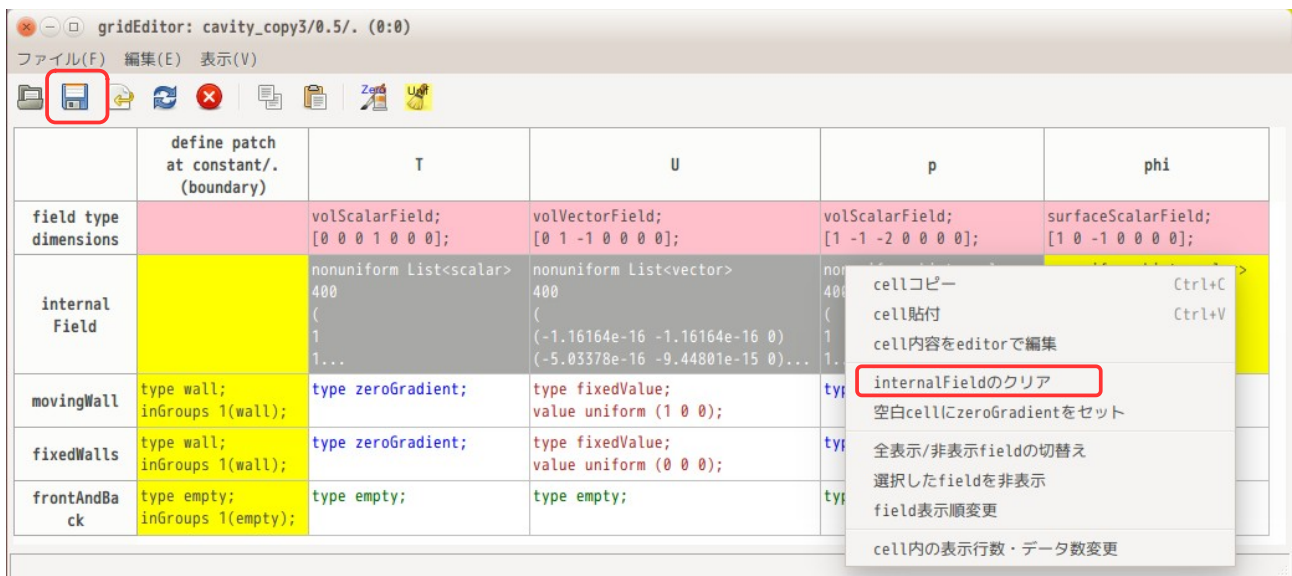
この nonuniform 形式のデータを uniform 形式に変更して、internalField をクリアする事ができる。

以下は、計算結果が入っている timeFolder を gridEditor で表示した状態になる。この中で T, U, p field の internalField をクリアしてみる。




	define patch at constant/ (boundary)	T	U	p	phi
field type		volScalarField;	volVectorField;	volScalarField;	surfaceScalarField;
dimensions		[0 0 0 1 0 0 0];	[0 1 -1 0 0 0 0];	[1 -1 -2 0 0 0 0];	[1 0 -1 0 0 0 0];
internal Field		nonuniform List<scalar> 400 ( 1 1...	nonuniform List<vector> 400 ( (-1.16164e-16 -1.16164e-16 0) (-5.03378e-16 -9.44801e-15 0)...	nonuniform List<scalar> 400 ( 1 1...	nonuniform List<scalar> 760 ( -2.37169e-20 5.75982e-20...
movingWall	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform (1 0 0);	type zeroGradient;	type calculated; value uniform 0;
fixedWalls	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty; value nonuniform 0();

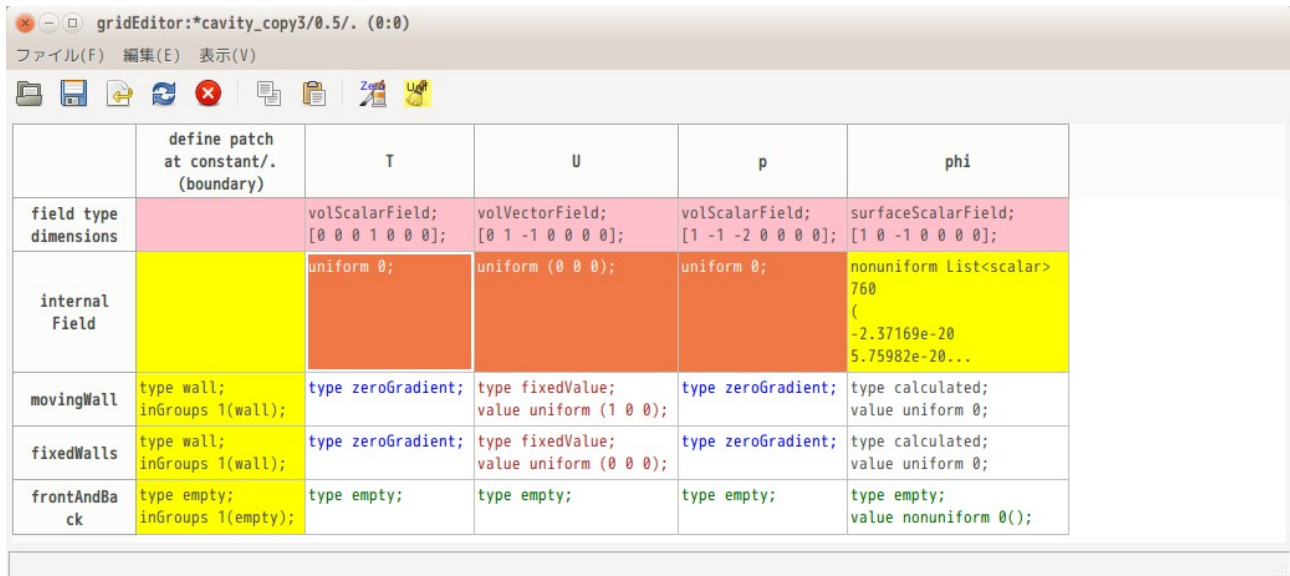
まず、以下の様に、クリアしたい internalField を選択し、右クリックしてポップアップメニューを表示させ、「internalFieldのクリア」を選択する。



	define patch at constant/ (boundary)	T	U	p	phi
field type		volScalarField;	volVectorField;	volScalarField;	surfaceScalarField;
dimensions		[0 0 0 1 0 0 0];	[0 1 -1 0 0 0 0];	[1 -1 -2 0 0 0 0];	[1 0 -1 0 0 0 0];
internal Field		nonuniform List<scalar> 400 ( 1 1...	nonuniform List<vector> 400 ( (-1.16164e-16 -1.16164e-16 0) (-5.03378e-16 -9.44801e-15 0)...	nonuniform List<scalar> 400 ( 1 1...	nonuniform List<scalar> 760 ( -2.37169e-20 5.75982e-20...
movingWall	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform (1 0 0);	type zeroGradient;	type calculated; value uniform 0;
fixedWalls	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty; value nonuniform 0();

cellコピー (Ctrl+C)  
 cell貼付 (Ctrl+V)  
 cell内容をeditorで編集  
**internalFieldのクリア**  
 空白cellにzeroGradientをセット  
 全表示/非表示fieldの切替え  
 選択したfieldを非表示  
 field表示順変更  
 cell内の表示行数・データ数変更

以上の操作で、以下の様に internalField がクリアされた状態になる。この後、 ボタンをクリックしてクリアした状態を保存して、field を書き換える。(保存しないと、クリアした結果が反映されない。)



The screenshot shows the gridEditor application window titled "gridEditor:\*cavity\_copy3/0.5/. (0:0)". The menu bar includes "ファイル(F)", "編集(E)", and "表示(V)". Below the menu is a toolbar with icons for file operations and editing. The main content area displays a table with the following data:

	define patch at constant/. (boundary)	T	U	p	phi
field type		volScalarField;	volVectorField;	volScalarField;	surfaceScalarField;
dimensions		[0 0 0 1 0 0 0];	[0 1 -1 0 0 0 0];	[1 -1 -2 0 0 0 0];	[1 0 -1 0 0 0 0];
internal Field		uniform 0;	uniform (0 0 0);	uniform 0;	nonuniform List<scalar> 760 ( -2.37169e-20 5.75982e-20...
movingWall	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform (1 0 0);	type zeroGradient;	type calculated; value uniform 0;
fixedWalls	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;
frontAndBack	type empty; inGroups 1(empty);	type empty;	type empty;	type empty;	type empty; value nonuniform 0();

尚、選択した internalField の内容が uniform 形式の場合は、変更せずそのまま。その内容が nonuniform 形式の場合にクリア (nonuniform 形式を uniform 形式に変更) する。

また、uniform 形式への変更は、そのデータタイプ (scalar、vector、symmTensor、tensor) に応じて、値を「0」クリアする。

#### 9-2-7. cell データを editor で編集 (「...」付きデータの編集)

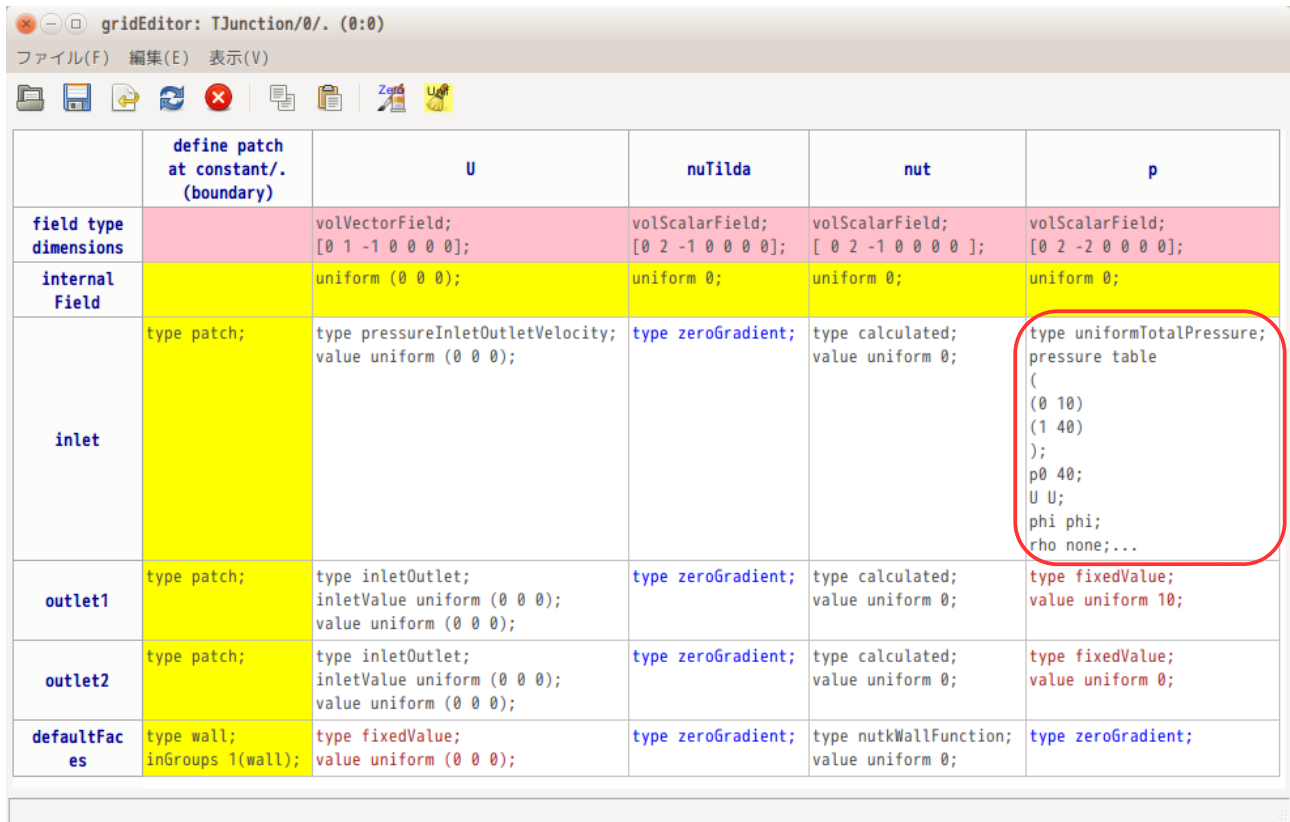
gridEditor では、cell に表示する最大行数を設定して、その行数以上は、表示させない様にしている。

この理由は、計算結果が入った field を gridEditor で開いた場合、cell 内のデータ量が膨大になり、表示しきれなくなる為。(特に internalField 内のデータは、1 ケの cell 内では表示しきれない。この為、cell に表示する最大行数を設定して、それ以上は表示させない様に設定し、データの最後には「...」を追加して、まだデータが続く事を表示させている。これにより、gridEditor の扱うデータが減るので、gridEditor を軽快に作動させる事ができる。

cell 内に表示させる最大行数の設定は、9-2-3 項に示す方法で設定している。

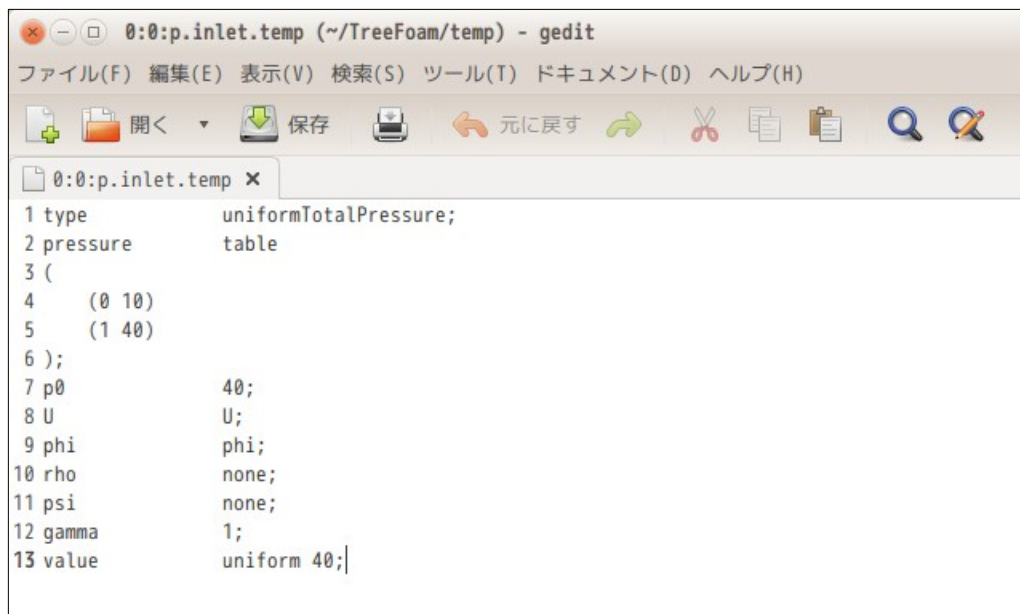
これにより、cell 内に表示できる行数 (デフォルトの設定は 10 行) に制限があるので、この行数以上の境界条件を設定・編集する事ができなくなってしまうので、これが編集できる様に工夫している。

例えば、tutorials の incompressible/pimpleFoam/TJunction の境界条件を下図に示しているが、field 「p」の patch 「inlet」部の境界条件が、cell 内の最大表示行数を超えている。(cell 内の最後が「...」で終わっている。)



	define patch at constant/ (boundary)	U	nuTilda	nut	p
field type dimensions		volVectorField; [0 1 -1 0 0 0 0];	volScalarField; [0 2 -1 0 0 0 0];	volScalarField; [0 2 -1 0 0 0 0];	volScalarField; [0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;	uniform 0;	uniform 0;
inlet	type patch;	type pressureInletOutletVelocity; value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;	type uniformTotalPressure; pressure table ( (0 10) (1 40) ); p0 40; U U; phi phi; rho none;...
outlet1	type patch;	type inletOutlet; inletValue uniform (0 0 0); value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 10;
outlet2	type patch;	type inletOutlet; inletValue uniform (0 0 0); value uniform (0 0 0);	type zeroGradient;	type calculated; value uniform 0;	type fixedValue; value uniform 0;
defaultFac es	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type nutkWallFunction; value uniform 0;	type zeroGradient;

このような場合、これを編集する為には、その cell をダブルクリックすると、editor でその cell が編集できるようになる。内容を編集後、editor を閉じる事によって、その内容が field に反映される。

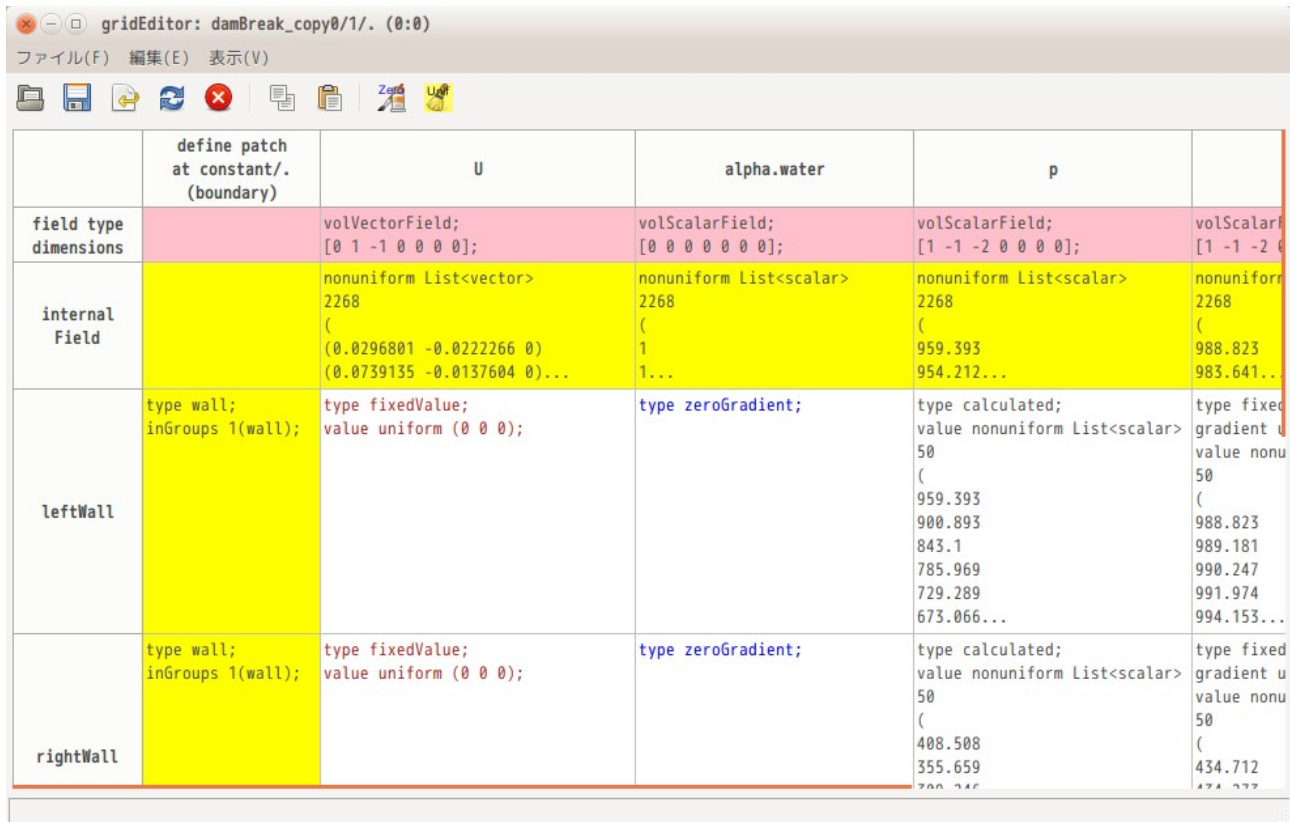


```

0:0:p.inlet.temp (~/.TreeFoam/temp) - gedit
ファイル(F) 編集(E) 表示(V) 検索(S) ツール(T) ドキュメント(D) ヘルプ(H)
開く 保存 元に戻す
0:0:p.inlet.temp x
1 type          uniformTotalPressure;
2 pressure      table
3 (
4   (0 10)
5   (1 40)
6 );
7 p0            40;
8 U             U;
9 phi           phi;
10 rho          none;
11 psi          none;
12 gamma       1;
13 value        uniform 40;

```

また、計算結果が入った field を gridEditor で開いた場合も同様な状態になる。以下は、damBreak の timeFolder 「1」 を gridEditor で開いた結果になる。



	define patch at constant/. (boundary)	U	alpha.water	p	
field type		volVectorField;	volScalarField;	volScalarField;	volScalarField;
dimensions		[0 1 -1 0 0 0];	[0 0 0 0 0 0];	[1 -1 -2 0 0 0];	[1 -1 -2 0 0 0];
internal Field		nonuniform List<vector> 2268 ( (0.0296801 -0.0222266 0) (0.0739135 -0.0137604 0)...	nonuniform List<scalar> 2268 ( 1 1...	nonuniform List<scalar> 2268 ( 959.393 954.212...	nonuniform List<scalar> 2268 ( 988.823 983.641...
leftWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type calculated; value nonuniform List<scalar> 50 ( 959.393 900.893 843.1 785.969 729.289 673.066...	type fixedValue; value nonuniform List<scalar> 50 ( 988.823 989.181 990.247 991.974 994.153...
rightWall	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type calculated; value nonuniform List<scalar> 50 ( 408.508 355.659 300.346...	type fixedValue; value nonuniform List<scalar> 50 ( 434.712 474.377...

全ての internalField と p field の patch 内容が全て表示されていない。(nonuniform 形式のデータの為、データ量が膨大であり、cell 内に表示しきれない。)

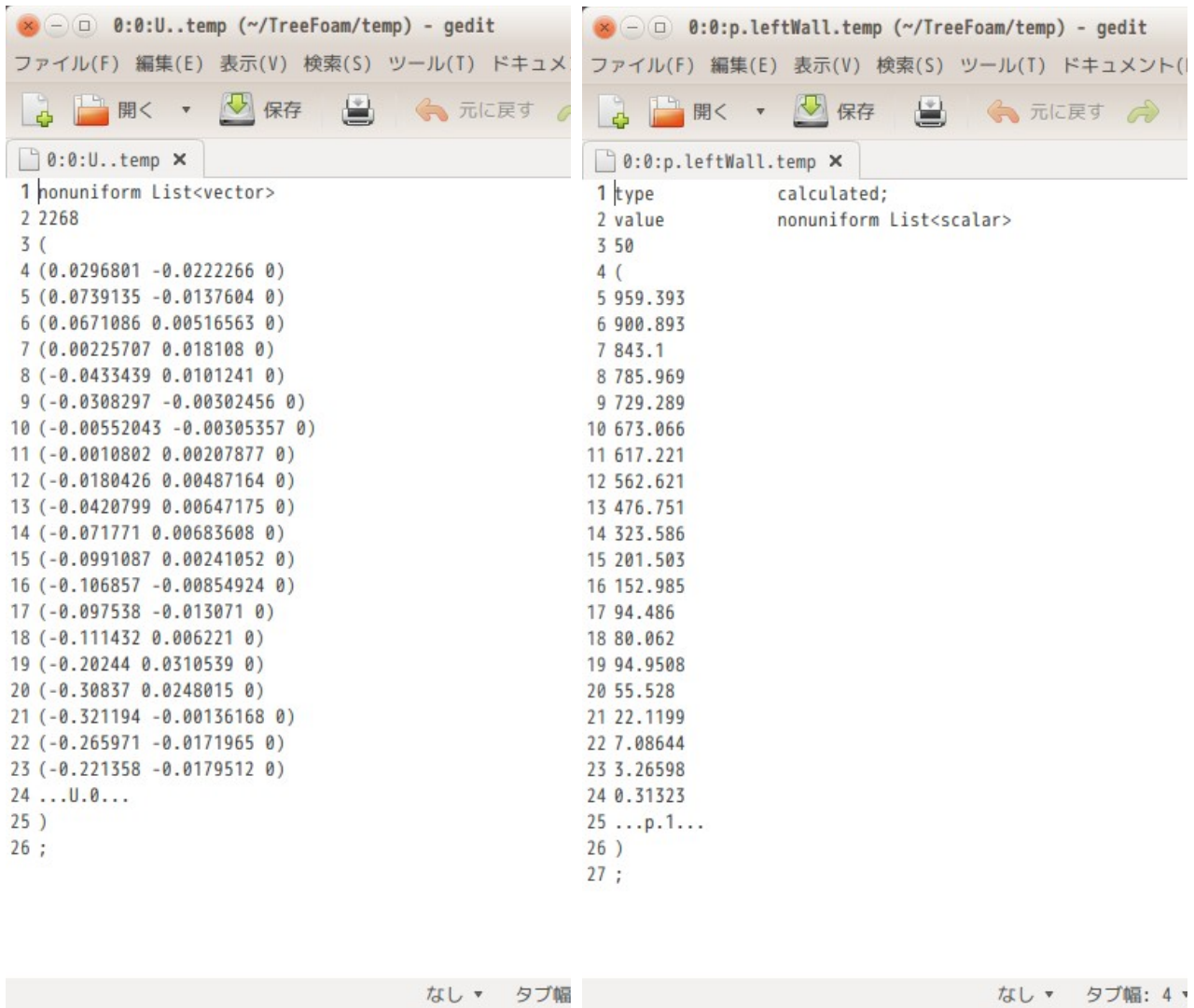
U field の internalField 部と、p field の leftWall patch 内容をダブルクリックして、editor で開いた状態を確認すると、以下の様に確認できる。(同時に 2 枚の editor を開くことができないので、下図は 1 枚ずつ editor で開いている。)

同時に複数の editor を開いて確認する場合は、該当する列ラベル部 (field 名部) をダブルクリックして、field 内容全体を editor で開く様にする事で、複数の editor を開き、同時に確認できる。

データ部の表示は、データ数の制限を設けているので、データ部の最後は「...U.0...」の様なインデックスが付加されている。このデータ数制限の設定は、8-1-5-2 項を参照。  
この為、「nonuniform List<vector>...」から「...U.0...」のインデックスまでは、編集できない事になる。

この為、この例では、U の internalField の内容は、編集できない。p field の leftWall の patch 内容については、「type calculated;」の行のみ編集でき、後は編集できない事になる。






### 9-3. field へのデータセット

#### 9-3-1. setFields によるデータセット

tutorials の damBreak の様に、field 内の特定部分に値をセットする場合、setFieldsDict を作成し、setFields コマンド実行して、値をセットする。値をセットする field や特定領域が多数ある場合は、単純作業が続くことになるので、ここで説明する方法を使った方が楽に setFieldsDict を作成する事ができる。

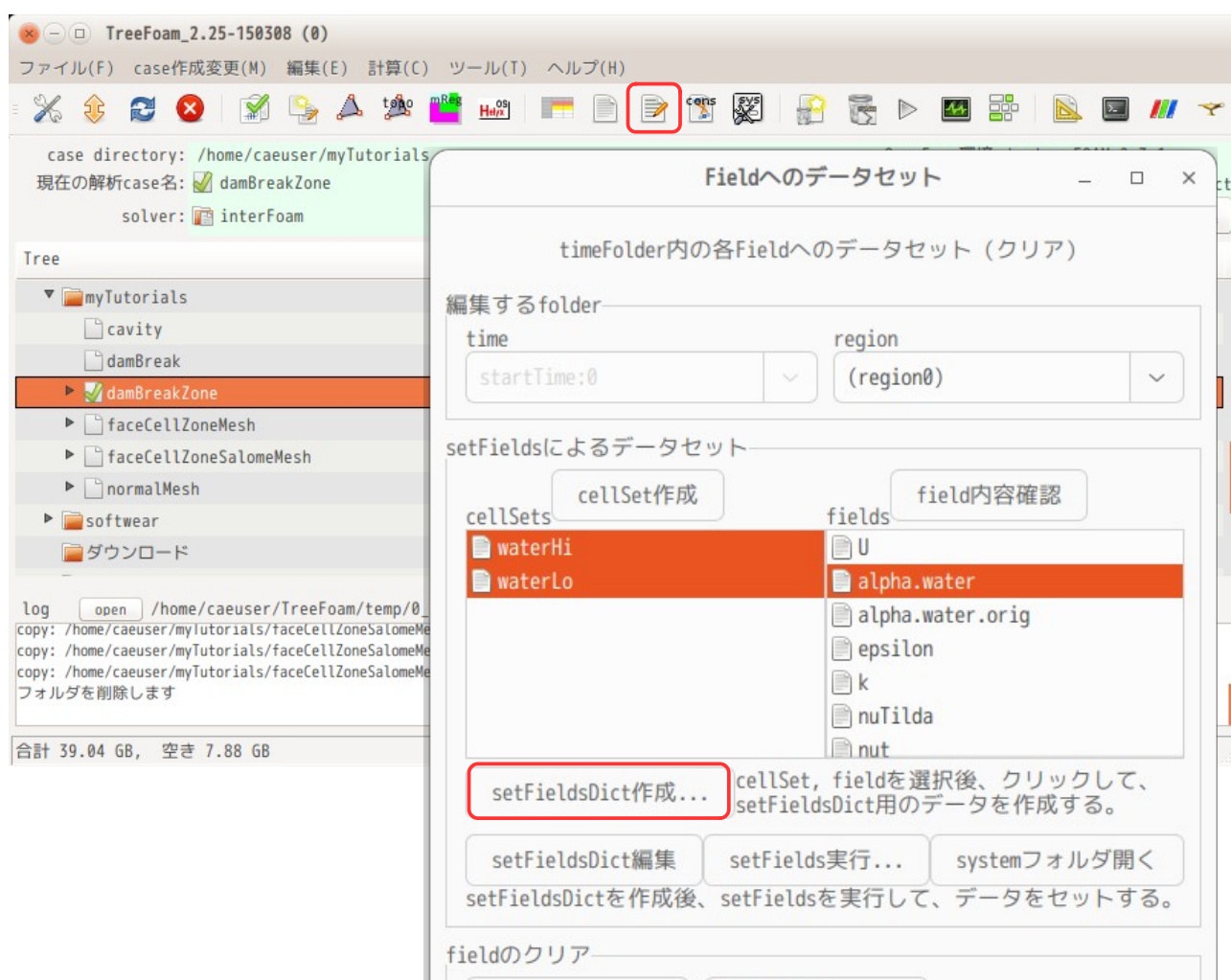
OF-13 から setFields コマンドが変更されている。従来の setFields では、cellSet の領域に値をセットしていたが、OF-13 では、cellZone の領域に値をセットすることができる。この為、cellSet を作成する必要がない。また topoSet コマンドも無くなっており、topoSetEditor が使えない。

##### 9-3-1-1. 起動方法と起動画面

起動は、TreeFoam 上の  ボタンをクリックして現れた「fields へのデータセット」画面上の「setFields」タグ上の「setFieldsDict 作成...」ボタンをクリックして起動する。

尚、「field へのデータセット」画面内には、この case 内に setFieldsDict が存在する場合、これを読み込んで、Dict 内で使用している cellSet や field を選択した状態に設定する。setFieldsDict が存在しない場合は、リストのみ表示されて選択された状態にはならない。

今回の例では、解析 case を 7-2 項で作成した case 「damBreakZone」にしているのので、setFieldsDict 内に記述してある cellSet (waterHi、waterLo) と field (alpha.water) が選択された状態になっている。(OF-13 の場合、cellZone (waterHi、waterLo) が表示、選択された状態になる。)



上記状態から「setFieldsDict 作成...」ボタンをクリックすると、以下の画面が表示される。この画面上にデータを入力して、setFieldsDict を作成する事になる。



### 9-3-1-2. cellSet (cellZone)、field 追加

前項で、選択した cellSet (cellZone) と field で、表を作成しているの、これらの洩れがあった場合、この画面上でも追加できる。

cellSet (cellZone) を追加する場合は、「cellSet 追加...」ボタン（「cellZone 追加...」ボタン）をクリックして、追加したい cellSet (cellZone) 名を選択して追加することになる。新たに cellSet (cellZone) を追加するためには、追加したい cellSet (cellZone) が存在している必要がある。



fieldを追加する場合は、「field 追加...」ボタンをクリックして、追加したい field を選択する。



以上の操作を行って、cellSet (cellZone) と field を追加した結果が以下になる。新たに cellSet (cellZone) 「waterMiddle」と field 「U」が追加されている。



### 9-3-1-3. box、cylinder、shpere 追加

box を追加する場合は、「box 追加」ボタンをクリックする事で、以下の様に box を追加する事ができる。cylinder (円柱)、sphere (球) の場合も同様に、各ボタンをクリックする事で追加できる。これらの場合は、最終的に box の座標を入力し直す必要がある。

また、これら box、cylinder、sphere の領域指定方法は、OpenFOAM のバージョンによって異なっている。特に、本家と v1906 とでは、cylinder、sphere の指定方法が異なっている。これに対応するために、この指定内容は、topoSetDict 内から抽出して設定している。



#### 9-3-1-4. データ入力

該当する cell にデータを直接入力する。今回の場合、以下の様に入力した。

waterHi、waterMiddle、waterLo、box には alpha.water を「1」にセット  
 waterMiddle には、Y 方向の初速 0.01 をセット

表形式にしていることによって、何をどうしたいのかが、直感的に理解できる様になっている。





#### 9-3-1-5. setFieldsDict 作成

入力した状態で setFieldsDict を作成する場合は、「Dict 保存」ボタンをクリックする事で、setFieldsDict が作成できる。

以下が前項の設定で、作成した setFieldsDict の内容になる。

```
// * * * * * //
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
    volVectorFieldValue U (0 0 0)
);
regions
(
    cellToCell
    {
        set waterHi;
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
    cellToCell
    {
        set waterLo;
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
    cellToCell
    {
        set waterMiddle;
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
}
```

```

        volVectorFieldValue U (0 0.01 0)
    );
}
boxToCell
{
    box (0 0.19 -1) (0.34 0.29 1);
    fieldValues
    (
        volScalarFieldValue alpha.water 1
    );
}
};
// *****

```

#### 9-3-1-6. csv 保存、読み込み

作成した表形式のデータを表形式のまま csv 形式で保存し、読み込む事ができる。  
今回のデータを csv 形式で保存して、office で読み込んだ結果が以下になる。

	A	B	C	D	E	F
1	<setFieldsDict>					
2	items	geometry	data	alpha.water	U	
3	defaultFieldValues			0 (0 0 0)		
4	waterHi			1		
5	waterLo			1		
6	waterMiddle			1 (0 0.01 0)		
7	(box)	box (0 0.19 -1) (0.34 0.28 1);		1		
8						
9						
10						
11						
12						

表形式のイメージをそのまま保存する。また、この csv 形式を読み込む事ができるので、cellSet が多数ある場合は、office などでもデータを入力して、読み込むこともできる。

#### 9-3-1-7. その他編集

前記した以外の編集方法として、行・列の削除、copy & paste、cell のクリアを表示しているボタンやポップアップメニューを表示させて、編集を行う事ができる。

これらの操作は、いずれも予め、対象の行・列・cell を選択した後、ボタンやポップアップメニューを選択して操作する事になる。

### 9-3-2. mapFields によるデータセット

pimpleDyMFoam の様な移動メッシュを扱う場合、メッシュ移動と共にメッシュが潰れてしまい、メッシュエラーで停止する事がある。このような場合、メッシュエラーの直前で停止させ、改めてメッシュを切り直して、新しいメッシュに今までの計算結果をマッピングする事によって、計算を継続させる事ができる。

#### 9-3-2-1. 移動メッシュの例

pimpleDyMFoam を使って、移動メッシュの計算を行ってみる。

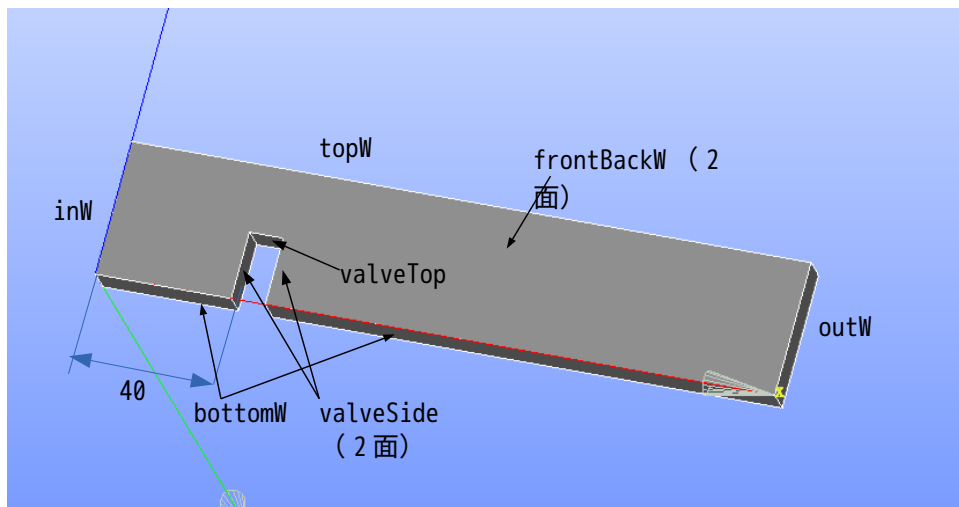
##### 9-3-2-1-1. case の作成

solver は、pimpleDyMFoam を使うので、tutorials から「incompressible:非圧縮性流れ」>「pimpleFoam」>「laminar/movingCone」(0F-11, 12 の場合は、「foamRun:一般的なCFD」>「incompressibleFluid」>「movingCone」) をコピーし、case 名を「movingValve」に変更しておく。この後、blockMesh を実行し、case を完成させておく。

さらに、メッシュ作成用の case が必要なので、cavity の case をコピーして、case 名を「movingValveMesh」に変更し、stl ファイル保存用の movingValveMesh/model フォルダを作成しておく。

##### 9-3-2-1-2. モデルの作成 (メッシュ作成)

解析モデルは、以下のモデルを考える。  
 大きさは、 $200 \times 10 \times 40$  mm のダクトにバルブ (スリット部:  $10 \times 10 \times 20$  mm) が存在する。  
 バルブ (スリット部) は、inW 側から 40 mm の場所に存在する。



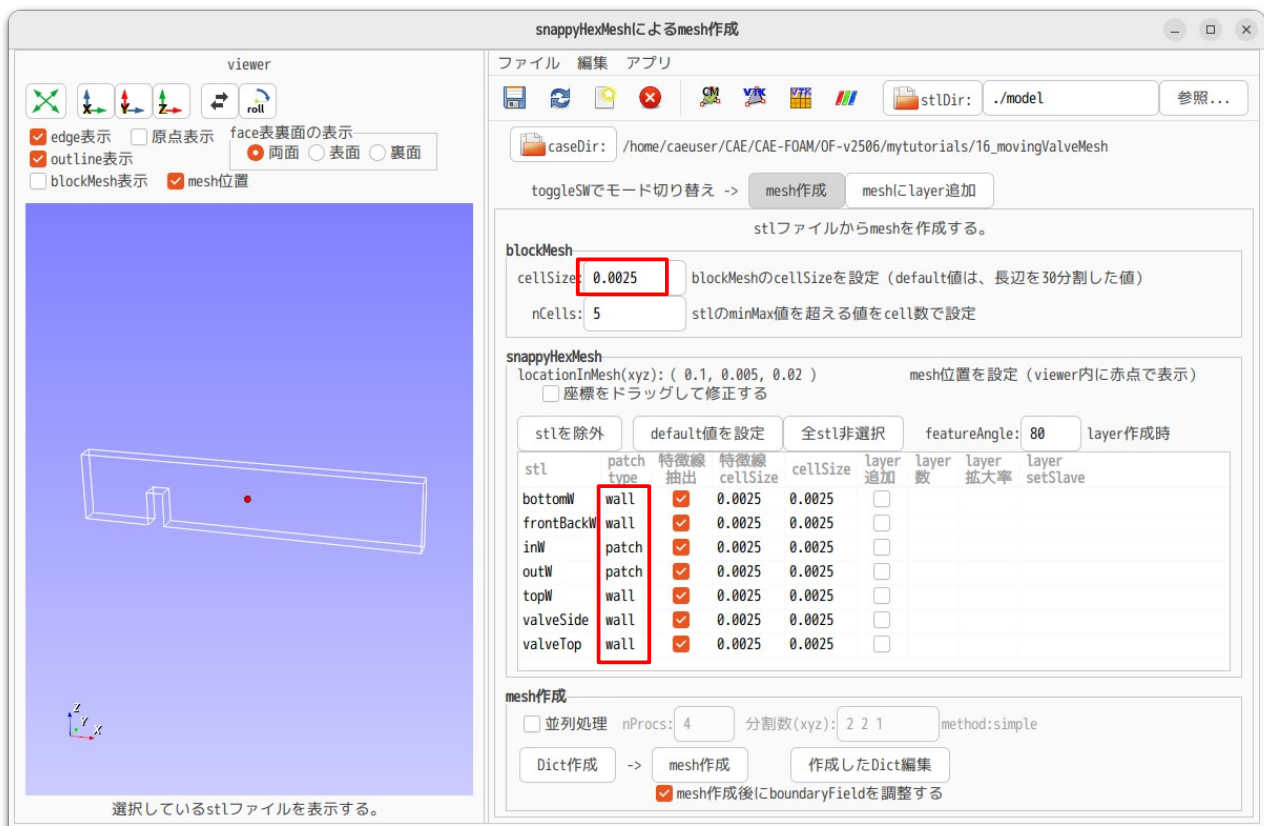
解析は、valveTop 部を上方へ移動させ、ダクトを狭める様に移動 (流路が次第に狭くなっていく) させた場合の流速や圧力分布を確認する。

stl ファイルは、以下のものを作成する。以下の 5 ケの stl ファイルで閉じた形状になっている。

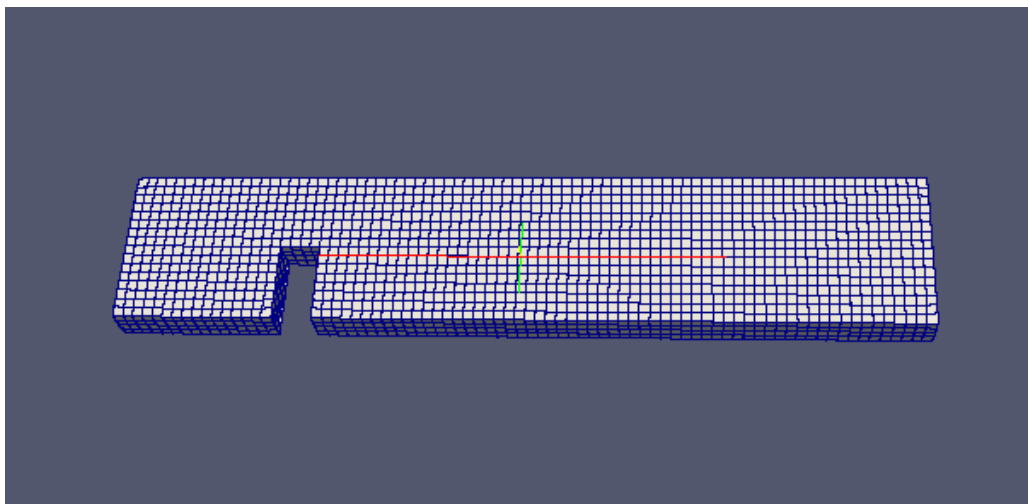
inW.stl	流入側
outW.stl	流出側
topW.stl	上面 (壁)
bottomW	底面 (壁)
frontBackW	表裏 2 面 (slip)
valveTop.stl	移動させる面 (壁)
valveSide.stl	移動とともに伸びる面 (壁)

これらの stl ファイルは、\$TreeFoamPath/data/stlFiles/movingValve フォルダ内に保管してあるので、ここから stl ファイルが取得できる。  
 これらの stl ファイルを「movingValveMesh/model」フォルダ内に保存しておく。

以下の画面内容でメッシュを作成している。(メッシュ作成方法は、7-1 項を参照。)  
  内が追記した内容。



以下ができあがった mesh になる。



この mesh をコピーして、解析用 case として作成した「movingValve」の case に mesh 貼り付けする。コピー & mesh 貼り付けする方法は、「7-2-6. 解析用 case の作成」を参照。

### 9-3-2-1-3. メッシュ移動の確認

作成したメッシュで、メッシュのみを移動させ、どこまで移動が可能なのか(メッシュエラーが発生しないか)確認してみる。

メッシュ移動用の case を作成する為、今の「movingValve」case をコピーして、新しく「moveMesh」の case を作成する。  
この後、controlDict の内容を以下の様に修正する。(0.1s 間隔で 1s まで計算する設定。)

```

/*-----*- C++ -*-----*\
=====
\\      F ield      | OpenFOAM: The Open Source CFD Toolbox
\\      O peration  | Website:  https://openfoam.org
\\      A nd        | Version:   9
\\      M anipulation
\*-----*/
FoamFile
{
    version      2.0;
    format       ascii;
    class        dictionary;
    location     "system";
    object       controlDict;
}
// *****

//application    pimpleFoam;
application      moveMesh;

startFrom        startTime;
startTime        0;
stopAt           endTime;
endTime          1.0;
deltaT           0.1;

//writeControl    timeStep;
writeControl      adjustableRunTime;

writeInterval     0.1;

purgeWrite        0;

writeFormat       binary;

writePrecision    6;

writeCompression off;

timeFormat        general;

timePrecision     6;

runTimeModifiable true;

adjustTimeStep    no;

maxCo             0.2;
/*
functions
{
    #include "cuttingPlane"
}
*/
// *****

```

また、constant/dynamicMeshDict」を以下の様に修正する。

```

/*-----*- C++ -*-----*\

```



さらに、system/fvSsolution 内に solver に「cellMotionUz」を追加しておく。

境界条件は、gridEditor を起動して設定する。  
移動方向がZ方向になる為、field名を「pointMotuinUx」を「pointMotuinUz」に修正する。  
この後、境界条件を設定する。

---

183

	define patch at constant/. (boundary)	U	p	pointMotionUz
field type dimensions		volVectorField; [0 1 -1 0 0 0];	volScalarField; [0 2 -2 0 0 0];	pointScalarField; [0 1 -1 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;	uniform 0;
bottomW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type fixedValue; value uniform 0;
frontBack	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zeroGradient;
inW	type patch; inGroups 1(patches);	type zeroGradient;	type zeroGradient;	type zeroGradient;
outW	type patch; inGroups 1(patches);	type zeroGradient;	type zeroGradient;	type zeroGradient;
topW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type fixedValue; value uniform 0;
valveSide	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zeroGradient;
valveTop	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type fixedValue; value uniform 0.01;

以上の条件で solver を 1 s 間走らせる。(valveTop 面が 10 mm 移動する。)

この後、FOAM 端末を起動して、「checkMesh」を実行して、メッシュエラーを確認する。確認した結果、以下の状態になる。

0.6 s (6 mm 移動) までは、Mesh OK だが、0.7 s (7 mm 移動) では、メッシュエラーが発生している。この為、このメッシュでは、0.6mm まで移動させる事が限界になる。

```

:
Time = 0.6
Checking geometry...
  Overall domain bounding box (0 0 0) (0.2 0.01 0.04)
  Mesh (non-empty, non-wedge) directions (1 1 1)
  Mesh (non-empty) directions (1 1 1)
  Boundary openness (8.96406e-18 -3.77545e-16 5.56993e-17) OK.
  Max cell openness = 2.25524e-16 OK.
  Max aspect ratio = 4.67778 OK.
  Minimum face area = 3.33818e-07. Maximum face area = 1.75026e-05. Face area magnitudes OK.
  Min volume = 3.32802e-09. Max volume = 3.26405e-08. Total volume = 7.73917e-05. Cell
volumes OK.
  Mesh non-orthogonality Max: 44.2808 average: 3.36092
  Non-orthogonality check OK.
  Face pyramids OK.
  Max skewness = 0.982128 OK.
  Coupled point location match (average 0) OK.

```

Mesh OK.

Time = 0.7

```

Checking geometry...
  Overall domain bounding box (0 0 0) (0.2 0.01 0.04)
  Mesh (non-empty, non-wedge) directions (1 1 1)
  Mesh (non-empty) directions (1 1 1)
  Boundary openness (-2.44473e-18 6.81512e-17 5.56996e-17) OK.
  Max cell openness = 2.3732e-16 OK.
  Max aspect ratio = 8.76424 OK.
  Minimum face area = 7.80121e-07. Maximum face area = 1.94884e-05. Face area magnitudes OK.
  Min volume = 1.69806e-09. Max volume = 3.54536e-08. Total volume = 7.72917e-05. Cell
volumes OK.

```

```

Mesh non-orthogonality Max: 125.061 average: 4.36297
***Number of non-orthogonality errors: 8.
<<Writing 8 non-orthogonal faces to set nonOrthoFaces
***Error in face pyramids: 12 faces are incorrectly oriented.
<<Writing 8 faces with incorrect orientation to set wrongOrientedFaces
Max skewness = 3.88886 OK.
Coupled point location match (average 0) OK.

```

Failed 2 mesh checks.

```

Time = 0.8
:

```

#### 9-3-2-1-4. pimpleFoamによる計算

inW側から 1 m/s の流速を与え、valveTop 面を 0.01 m/s の速度で上昇させる設定で、計算する。  
「movingValve」を解析 case に設定し、まず、constant/dynamicMeshDict を前項と同じ内容に修正しておく。  
この後、境界条件を以下の様に設定した。

	define patch at constant (boundary)	U (1)	p (2)	pointMotionUz (3)
field type		volVectorField;	volScalarField;	pointScalarField;
dimensions		[0 1 -1 0 0 0 0];	[0 2 -2 0 0 0 0];	[0 1 -1 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;	uniform 0;
bottomW	type wall; inGroups List<word> 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type fixedValue; value uniform 0;
frontBackW	type wall; inGroups List<word> 1(wall);	type slip;	type slip;	type slip;
inW	type patch; inGroups List<word> 1(patch);	type fixedValue; value uniform (1 0 0);	type zeroGradient;	type zeroGradient;
outW	type patch; inGroups List<word> 1(patch);	type inletOutlet; inletValue uniform (0 0 0); value uniform (0 0 0);	type fixedValue; value uniform 0;	type zeroGradient;
topW	type wall; inGroups List<word> 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type fixedValue; value uniform 0;
valveSide	type wall; inGroups List<word> 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type zeroGradient;
valveTop	type wall; inGroups List<word> 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	type fixedValue; value uniform 0.01;

system/controlDict は、以下の様に修正している。

```

/*-----*- C++ -*/
=====
\\      / F ield      OpenFOAM: The Open Source CFD Toolbox
\\    /  O peration  Website:  https://openfoam.org
\\  /    A nd        Version:  9
\\ /     M anipulation
/*-----*/

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

application    pimpleFoam;
startFrom      latestTime;

startTime      0;

stopAt         endTime;

```

```

endTime      0.6;
deltaT        5e-06;
//writeControl    timeStep;
writeControl  adjustableRunTime;
writeInterval  0.1;
purgeWrite     0;
writeFormat    binary;
writePrecision 6;
writeCompression off;
timeFormat     general;
timePrecision  6;
runTimeModifiable true;
adjustTimeStep yes;
maxCo          0.5;
/*
functions
{
    #include "cuttingPlane"
}
*/
// *****

```

以上の設定で、0.6 s まで計算させる。

0.6 s 以上は、メッシュエラーが発生するので、計算は 0.6 s まで行い、これ以降は新たに作り直したメッシュで計算を継続する。

#### 9-3-2-1-5. メッシュの作り直し (リメッシュ) とデータのマッピング

0.6 s までの計算が終了した段階で、この時と同じ状態のメッシュを新たに作り直す。このメッシュに 0.6 s 後の計算結果を各 field にマッピングする事になる。

この操作 (リメッシュとマッピング) をする為に、「remeshAndMapping.py」コマンドを準備している。このコマンドは、最終の計算結果 (latestTime) フォルダ内の patch 形状を stl 形式で取得し、その stl を使ってメッシュを作り直す。さらに、latestTime 内の field データを新しいメッシュにマッピングする。この為、コマンド実行後は、直ぐに計算開始できる状態になっている。

具体的には、0.6s までの計算結果がある case 「movingValve」を解析 case として設定して、FOAM 端末を起動する。この後、端末上で、以下を入力する。

```

$ remeshAndMapping.py --tempCase ../movingValveMesh
-----
snappyHexMesh でメッシュを作成した case

```

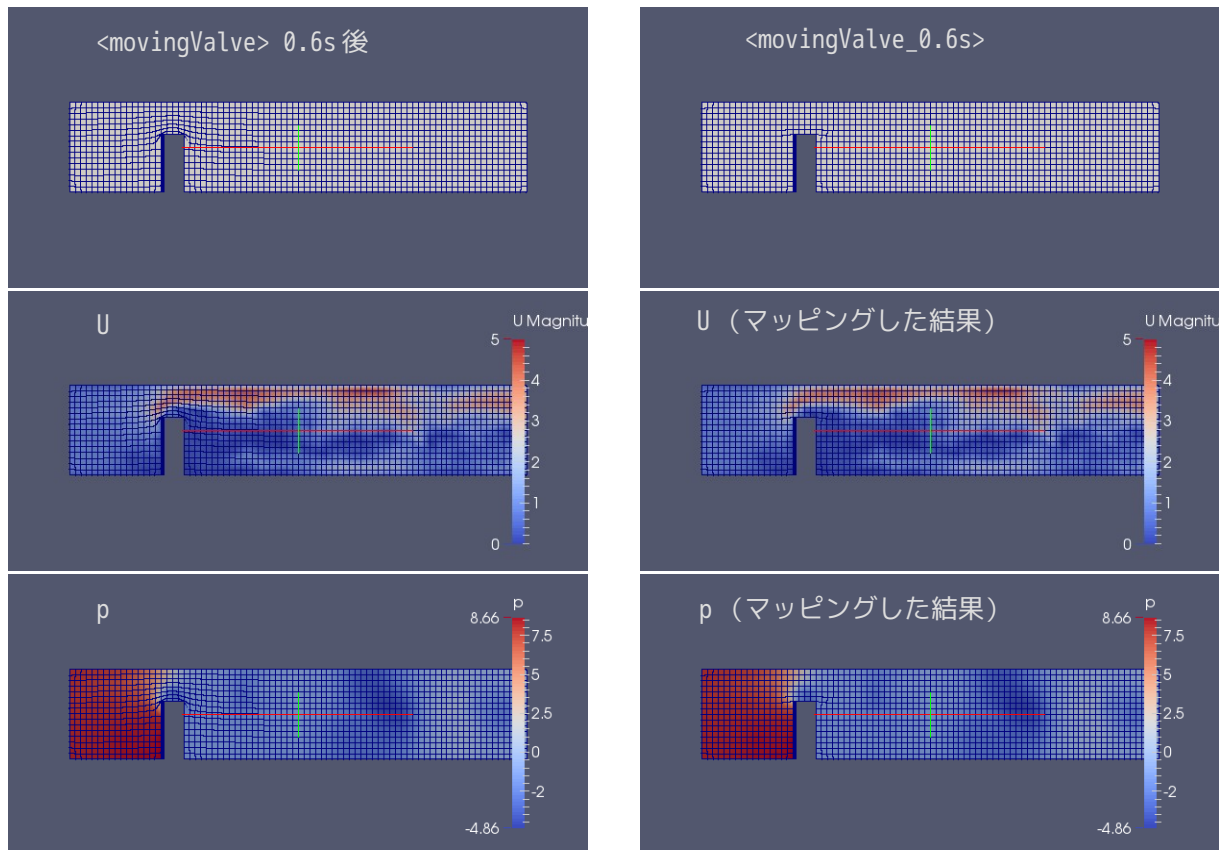
「movingValveMesh」case は、currCase のメッシュを作成した case の為、メッシュ作成の為の設定が既に成されている case であり、この case 内の stl ファイルを今回取得した stl に置き換えれば、新しいメッシュが出来上がる。

このコマンドは、以下の操作を行っている。

- 1) currCaseDir の latestTime 内の全 patch 形状を stl 形式で取得する。
- 2) 取得した stl を使って、「movingValveMesh」case フォルダでメッシュを作成する。
- 3) currCaseDir の「0」フォルダ内の全 field を「movingValveMesh」case にコピーする。
- 4) currCaseDir の latestTime の値を「movingValveMesh」の field に mapping する。
- 5) mapping した field と新しい mesh を currCaseDir に戻す。

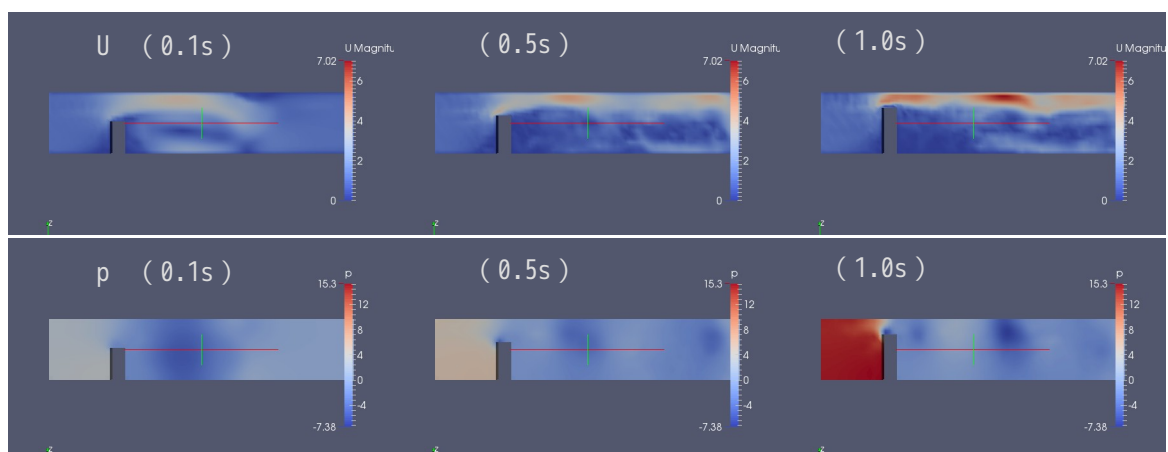
この為、currCase「movingValve」では、リメッシュとマッピングを行った事になり、直ぐに計算を開始することができる。

以下がリメッシュ、マッピングした結果になる。



データがマッピングできているので、さらに計算を継続してみる。  
境界条件は、変わっていないので、このまま計算を継続する。1s まで計算させた。

下図が、最初(0s)から最後(1s) まで計算させた結果になる。

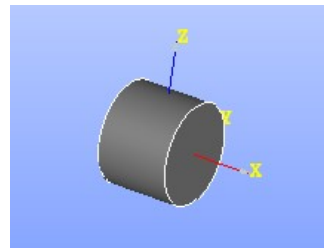
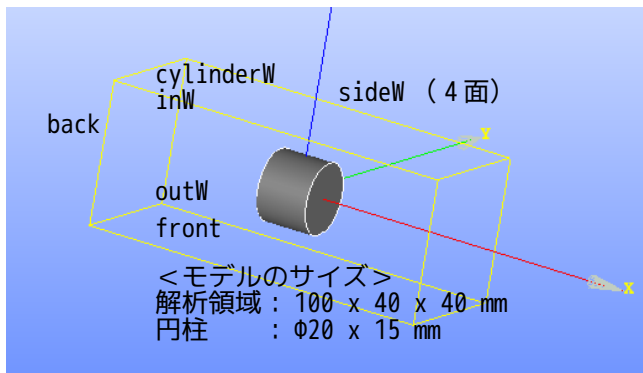


## 9-4. 内部 patch の作成

### 9-4-1. cyclic、mapped、baffle の patch 作成方法



モデル内部に cyclic, mapped, baffle の内部 patch を作成してみる。モデルは、以下の様にモデル内部の中心に内部 patch 用の円柱形状を定義する。この円柱形状の面に cyclic, mapped, baffle の内部 patch を定義してみる。



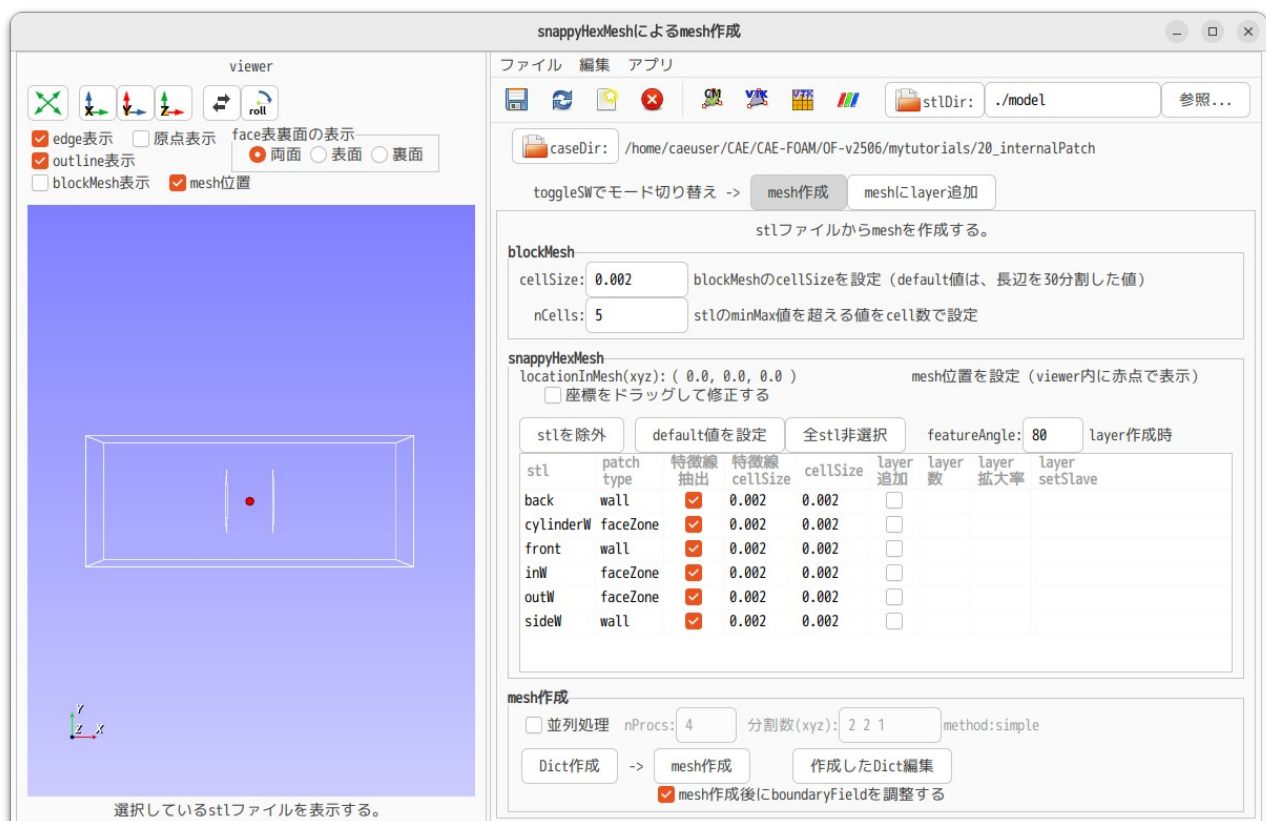
このモデルで以下の stl ファイルを作成する。  
 front.stl, back.stl, sideW.stl      patch 用  
 inW.stl, outW.stl, cylinderW.stl    内部 patch 用

内部 patch と stl ファイルを以下の様に設定し作成する。

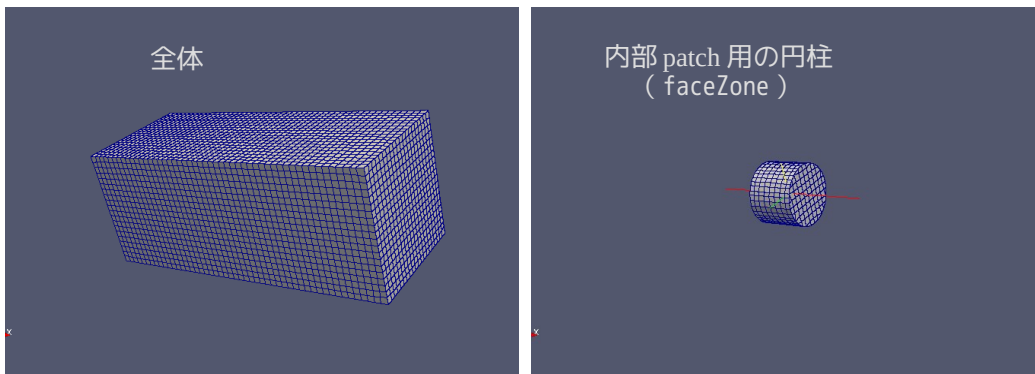
cylinderW      baffle  
 inW.stl       mappedPatch  
 outW.stl      cyclic


上記 stl は、\$TreeFoamPath/data/stlFiles/internalPatch フォルダ内に保存しているので、ここから取得できる。

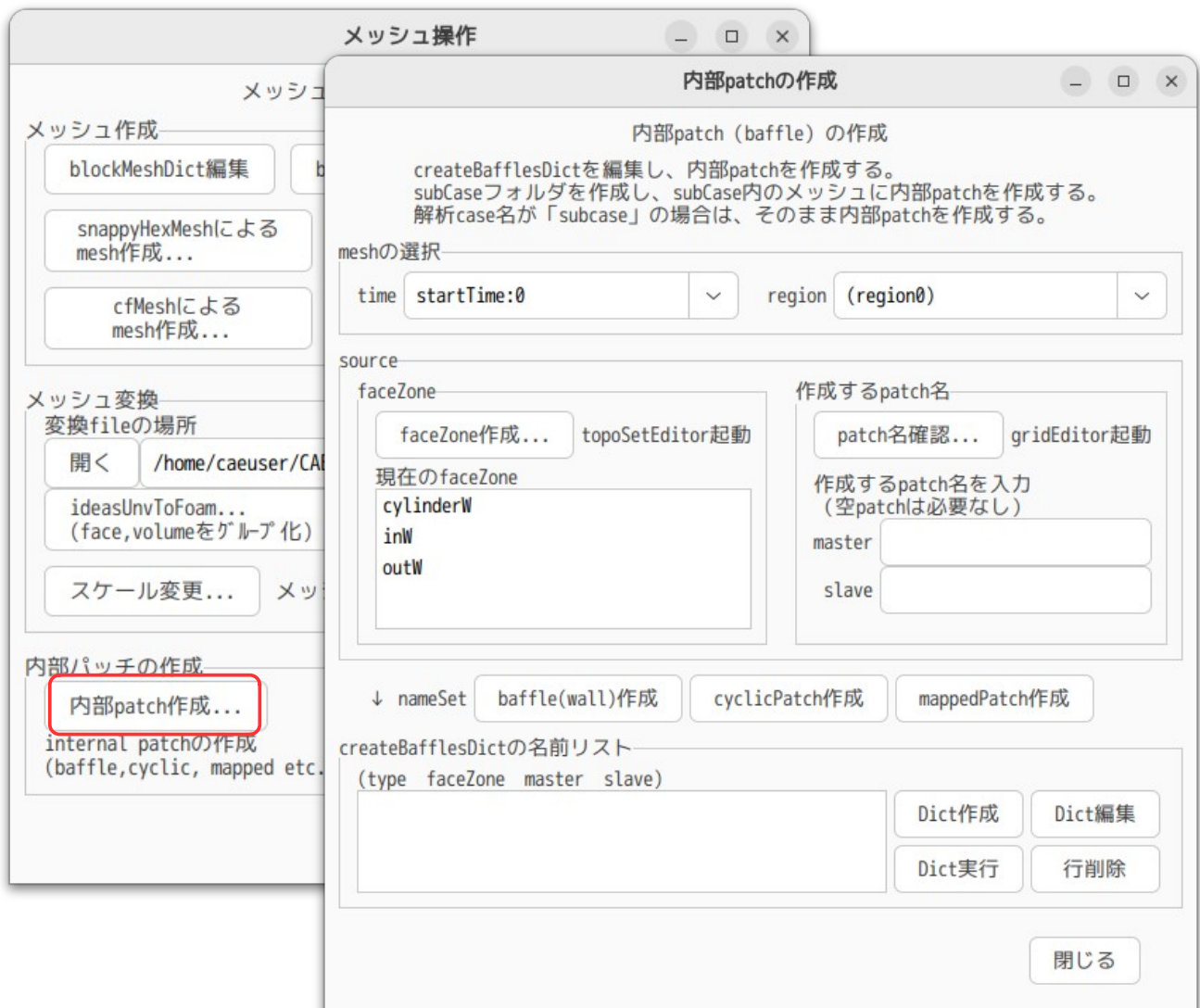
case は、tutorials の cavity をコピーして「internalPatch」case を作成する。  
 この case 内にて、上記 stl ファイルを使って、7-2 項の方法 (snappyHexMesh) で、メッシュを作成する。  
 mesh 作成の為に csv ファイルは、以下の様に作成した。内部 patch を作成する為に定義した cylinderW, inW, outW は、faceZone として設定している。



できあがったメッシュが以下になる。予定通りに faceZone ができあがっている。



できあがったメッシュに、内部 patch を作成する為に、TreeFoam 上の  ボタンをクリックして、現れた画面上で、「内部 patch 作成...」ボタンをクリックして、「内部 patch の作成」画面を表示させ、ここで内部 patch を作成する。



内部 patch は、faceZone を元に作成するので、前記「内部 patch の作成」画面が表示された時点で、いま存

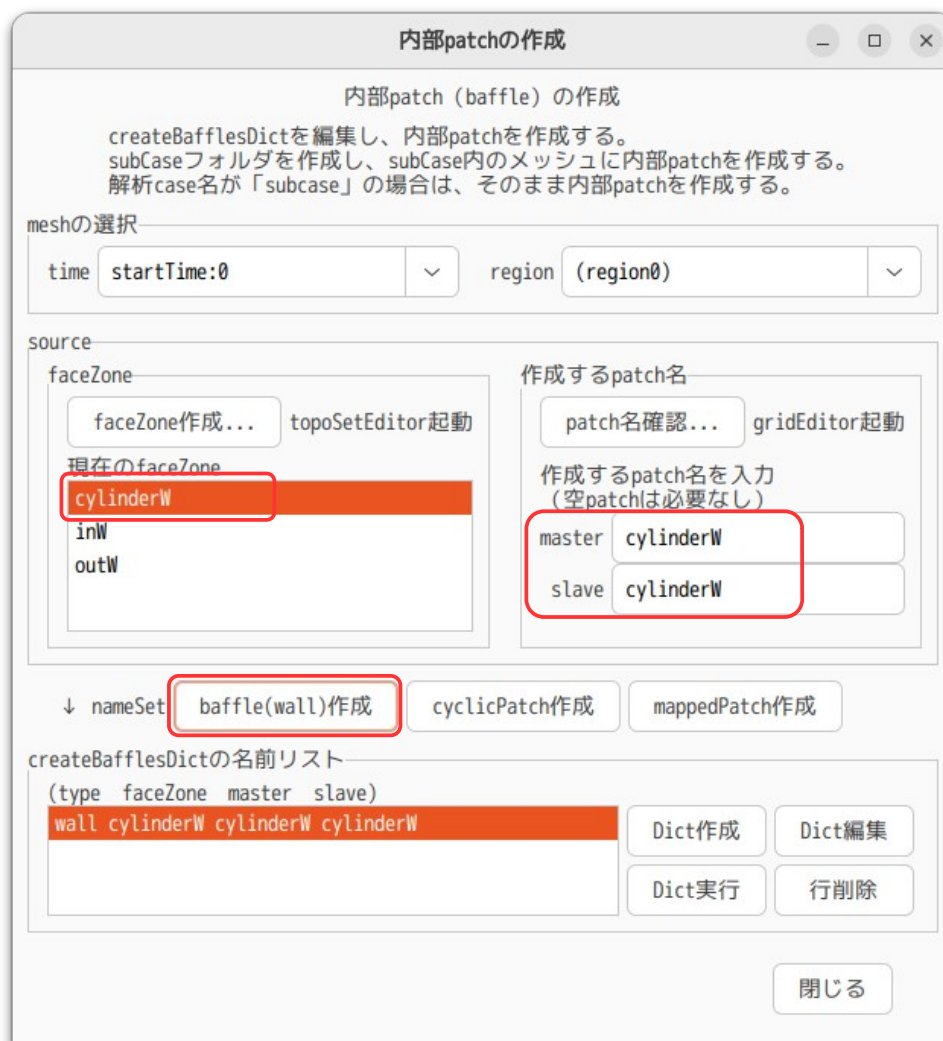
在している faceZone のリストが表示されている。

今回は、各々の faceZone から以下の内部 patch を作成するので、以下順番に説明する。

FaceZone	内部 patch
cylinderW	baffle
inW	mappedPatch
outW	cyclic

#### 1) cylinderW (baffle) 名前リスト作成

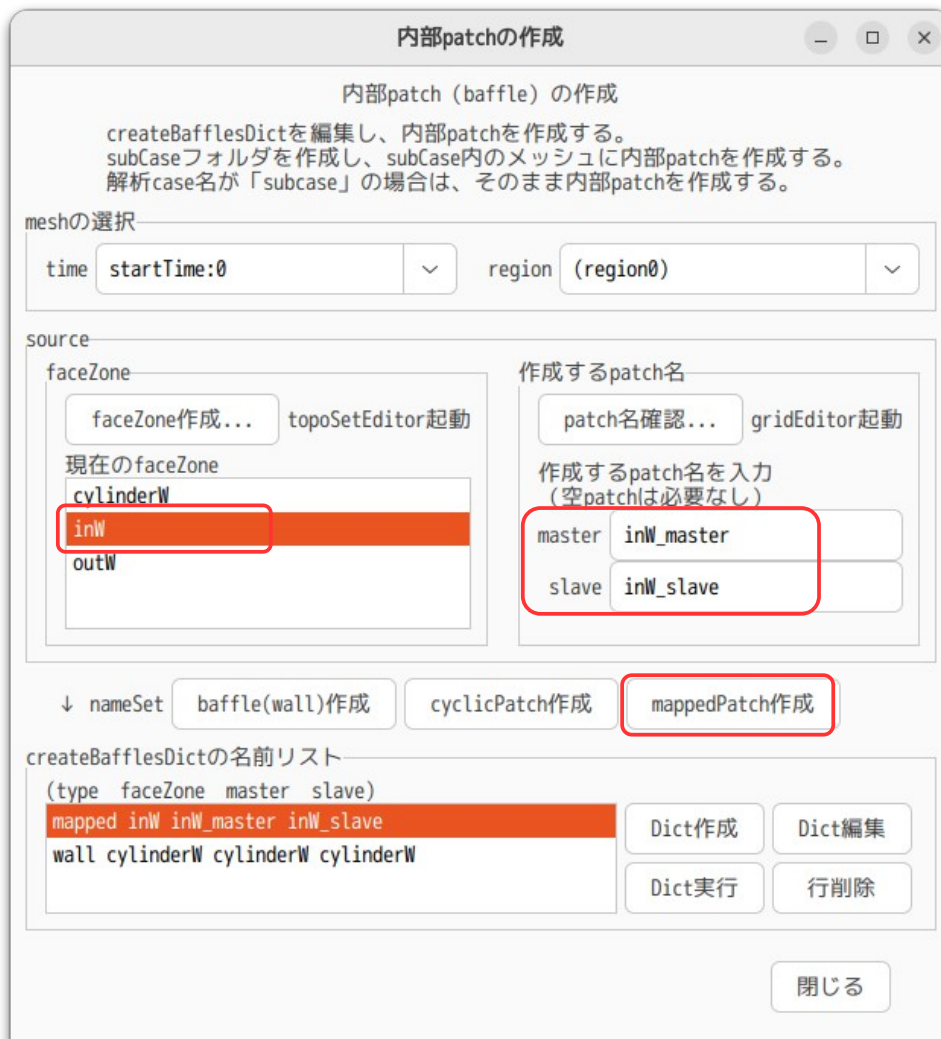
以下の様に faceZone 「cylinderW」を選択する。この選択により、master、slave に patch 名が表示されるが、ここでは baffle を作成するので、master、slave とともに、同じ名称に変更する。  
このあと、「patch 作成」ボタンをクリックして、名前リストに登録しておく。



尚、誤って名前リストに登録してしまった場合は、該当する名前リストを選択して、「行削除」ボタンをクリックして、その行（名前リスト）を削除して、再度登録し直す。

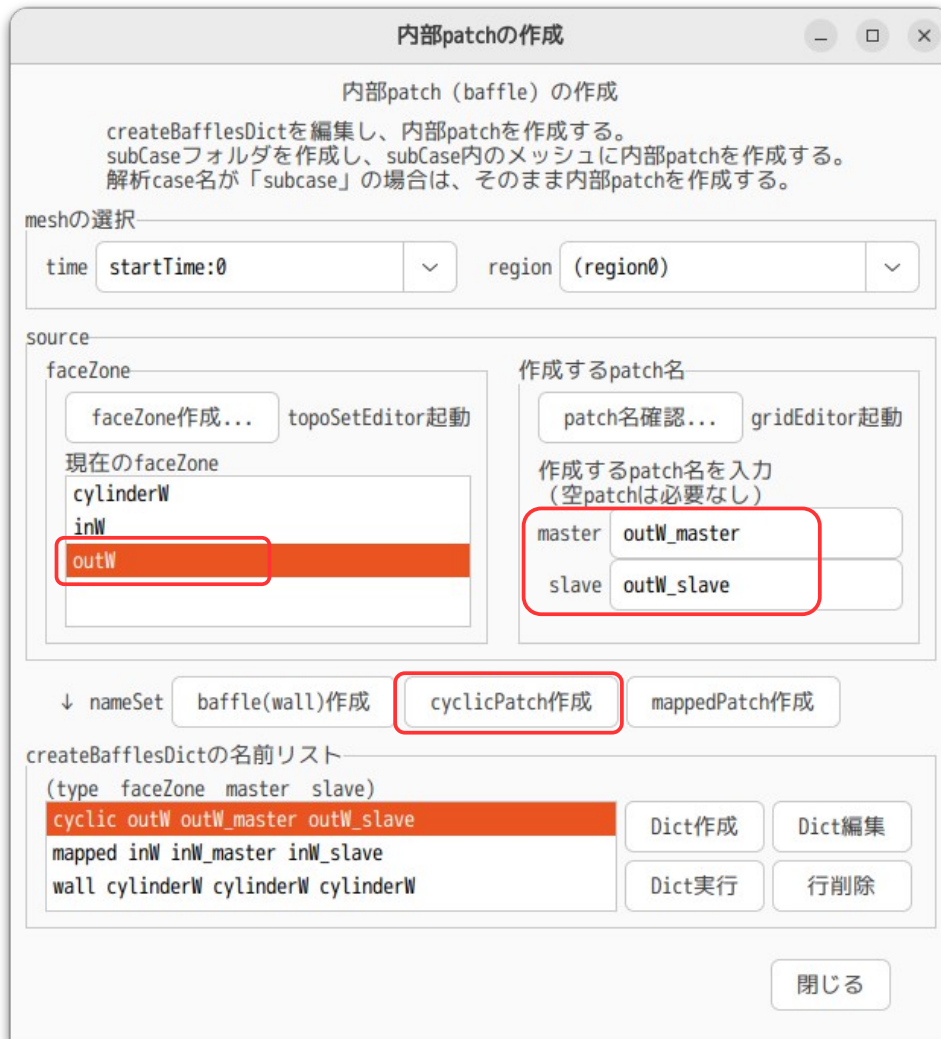
#### 2) inW (mappedPatch) 名前リスト作成

前項と同様に、faceZone 「inW」を選択する。master、slave 名は、そのままにしておく。  
このあと「mappedPatch 作成」ボタンをクリックして、名前リストに登録する。



### 3) outW (cyclic) の名前リスト作成

前項と同様に faceZone「outW」を選択する。master、slave 名は変更せずそのまま。  
このあと、「cyclic 作成」ボタンをクリックして、名前リストに登録する。



#### 4) createBafflesDict 作成、実行

以上の操作により、内部 patch を作成する為の全ての名前リスト（下表）が取得できた事になる。

Type	faceZone	master	slave	備考
cyclic	outW	outW_master	outW_slave	cyclic 用
mapped	inW	inW_master	inW_slave	mappedPatch 用
wall	cylinderW	cylinderW	cylinderW	baffle 用

この情報を元に createBafflesDict を作成する。その方法は、「Dict 作成」ボタンをクリックする。これにより、Dict ができあがる。引き続き「Dict 実行」ボタンをクリックして、内部 patch を作成する。





```

cylinderW    //baffles is created
{
    //- Use predefined faceZone to select faces and orientation.
    type      faceZone;
    zoneName   cylinderW;
    patches
    {
        master
        {
            //- Master side patch
            name      cylinderW;
            type      patch;
        }
        slave
        {
            //- Slave side patch
            name      cylinderW;
            type      patch;
        }
    }
}


inW    //baffles is created
{
    //- Use predefined faceZone to select faces and orientation.
    type      faceZone;
    zoneName   inW;
    patches
    {
        master
        {
            //- Master side patch
            name      inW_master;
            type      patch;
        }
        slave
        {
            //- Slave side patch
            name      inW_slave;
            type      mappedPatch;
            sampleRegion region0;
            sampleMode nearestPatchFace;
            samplePatch inW_master;
        }
    }
}

outW    //baffles is created
{
    //- Use predefined faceZone to select faces and orientation.
    type      faceZone;
    zoneName   outW;
    patches
    {
        master
        {
            //- Master side patch
            name      outW_master;
            type      cyclic;
            neighbourPatch outW_slave;
        }
        slave
        {
            //- Slave side patch
            name      outW_slave;
            type      cyclic;
            neighbourPatch outW_master;
        }
    }
}
}

```

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

## 5) 内部 patch の確認

内部 patch は、「Dict 実行」ボタンをクリックする事で、現在の解析 case 内に、「subCase」フォルダが作成され、この case に内部 patch が追加されたメッシュができあがっている。  
この為、TreeFoam 上で  ボタンをクリックして、Tree 構造を再読み込みし、「subCase」フォルダを解析 case に設定し直して、できあがった内部 patch を確認する。

gridEditor を起動して、内部 patch を確認すると、以下の様に確認できる。  
(下図は、field U, p のみ表示させている。)

gridEditor: subCase/0 (0:0)			
ファイル(F) 編集(E) 表示(V)			
define patch at constant (boundary)			
field type		U (1)	p (2)
dimensions		volVectorField; [0 1 -1 0 0 0 0];	volScalarField; [0 2 -2 0 0 0 0];
internal Field		uniform (0 0 0);	uniform 0;
back	type wall; inGroups List<word> 1(wall);	type zeroGradient;	type zeroGradient;
front	type wall; inGroups List<word> 1(wall);	type zeroGradient;	type zeroGradient;
sideW	type wall; inGroups List<word> 1(wall);	type zeroGradient;	type zeroGradient;
outW_master	type cyclic; inGroups List<word> 2(cyclic outW); matchTolerance 0.0001; neighbourPatch outW_slave; transformType none;	type cyclic;	type cyclic;
outW_slave	type cyclic; inGroups List<word> 2(cyclic outW); matchTolerance 0.0001; neighbourPatch outW_master; transformType none;	type cyclic;	type cyclic;
inW_master	type patch; inGroups List<word> 1(inW);	type zeroGradient;	type zeroGradient;
inW_slave	type mappedPatch; inGroups List<word> 1(inW); name inW_slave; sampleRegion region0; sampleMode nearestPatchFace; samplePatch inW_master;	type zeroGradient;	type zeroGradient;
cylinderW	type wall; inGroups List<word> 2(wall cylinderW);	type zeroGradient;	type zeroGradient;

## 6) 境界条件の設定

内部 patch 付きのメッシュができあがったので、この patch を使って計算できる状態にあるが、計算する為には、境界条件を設定する必要があるので、ここで設定する。

境界条件は、以下の様に設定する。

inW_master	p を「10」	} inW の表裏に圧力差を設定し、cylinder 内に流入させる
inW_slave	p を「0」	
cylinderW	壁	
outW_master	cyclic	} outW の表裏は、cyclic に設定し、流出させる
outW_slave	cyclic	

以下が、設定した結果になる。

この設定は、モデル内部に送風器を設置したイメージで、outW から流出して広がり、そのまま回りこんで

inW 側に流れこむ状態を計算している。

	define patch at constant/. (boundary)	U	p	
dimensions		[0 1 -1 0 0 0 0],	[0 2 -2 0 0 0 0],	
internal Field		uniform (0 0 0);	uniform 0;	
back	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	baffle
front	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	
sideW	type wall; inGroups 1(wall);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	
cylinderW	type patch; inGroups 1(cylinderW);	type fixedValue; value uniform (0 0 0);	type zeroGradient;	
inW_master	type patch; inGroups 1(inW);	type zeroGradient;	type fixedValue; value uniform 10;	mappedPatch
inW_slave	type mappedPatch; inGroups 2(mappedPatch inW); sampleMode nearestPatchFace; sampleRegion region0; samplePatch inW_master;	type mapped; setAverage false; average (0 0 0); value uniform (0 0 0);	type fixedValue; value uniform 0;	
outW_master	type cyclic; inGroups 2(cyclic outW); matchTolerance 0.0001; transform unknown; neighbourPatch outW_slave;	type cyclic;	type cyclic;	cyclic
outW_slave	type cyclic; inGroups 2(cyclic outW); matchTolerance 0.0001; transform unknown; neighbourPatch outW_master;	type cyclic;	type cyclic;	

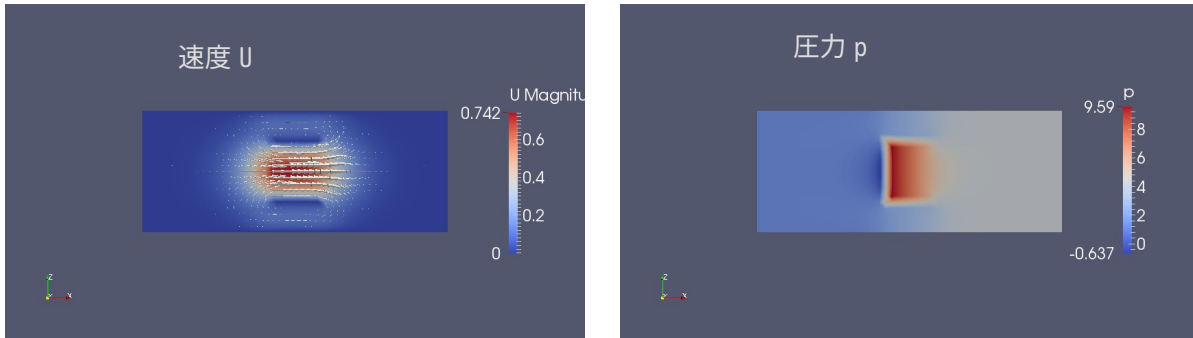
OF-11,12 の場合は、U field の inW\_slave patch の境界条件は、以下で設定する。

```
type mappedValue;           //この内容が変わる
setAverage false;
average (0 0 0);
value uniform (0 0 0);
```

## 6) 計算開始

設定した境界条件で、計算を開始してみる。この case は、元々 tutorials の cavity をコピーして作成した case の為、メッシュ以外は、cavity の設定がそのまま残っている。  
そのままだと、計算がうまく収束しなかった為、controlDict 内の deltaT を 0.005→0.001 に変更して計算させている。

OF-11, 12 の場合は、異なる solver を使った複数の cavity case があり、計算させるには、solver が incompressibleFluid の cavity を使って計算させる。  
また、この時、physicalPropertis の nu を 1e-5→0.01 に変更し、controlDict では、deltaT 0.001, writeControl adjustableRunTime, writeInterval 0.1, adjustTimeStep yes, maxCo 0.5 の設定で、計算を開始させる。(maxCo の設定で計算させないと発散しやすい。)



以上の方法で、cyclic や mapped、baffle の内部 patch を作成する事ができる。

#### 9-4-2. cyclic、baffle を含む mesh の並列計算方法


cyclic や baffle の patch を作成したモデルを並列計算する場合には、注意が必要になる。

cyclic の場合、表裏のペアとなる face を持っている。baffle についても、表裏のペアになる face を持っている。これらペアになる face が並列計算の為にメッシュ分割時に、ペアの face が分離されてしまい、同じ cpu でペアになる相手の face が存在しない事が生じてしまうと、実行時にエラーが発生する。

この時のエラーの症状は、シングルコアでは、計算がうまく走る。並列計算させると、分割数 (cpu 数) によって、エラーが発生したり、しなかったりする。

mappedPatch も同様にペアになる face を持っているが、mappedPatch の場合は、patch 間の計算をする時に、全 cpu が持っている相手側の face の値を取得した上で、自身の face の値を計算しているので、エラーの発生はない。

このようなエラーが発生した場合、以下の設定を行うことで、エラーを回避する事ができる。

TreeFoam 上の  ボタンをクリックして、「並列計算」画面を表示させる。

この後、「preserve」をチェックして、「preserve 設定...」ボタンをクリックし、「preserve 設定」画面を表示させる。この画面上で、設定を行う事になる。

##### 1) cyclic の設定

cyclic の場合、「outW\_master」「outW\_slave」に設定されている。この為、patches 欄で「outW\_master」と「outW\_slave」を選択して「選択>>」ボタンをクリックして、preservePatches 側に移動させる。

##### 2) baffle の設定

「preserveBaffles の設定」にチェックをつける。

以上の設定を行った後、「preserve 設定」ボタンをクリックして、設定を行う。この操作により、decomposeParDict が書き換わる。

修正された decomposeParDict を使って、メッシュ分割をやり直す事によって、表裏のペアになる face が分割されず、ペア間の計算が同じ cpu で計算できる事になる。





以下が、修正された decomposeParDict の内容になる。

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

```

numberOfSubdomains 4;
preserveBaffles true;
preservePatches
(
    outW_master outW_slave
);

```

5行が追加

```

//- Keep owner and neighbour on same processor for faces in zones:
// preserveFaceZones (heater solid1 solid3);

//- Keep owner and neighbour on same processor for faces in patches:
// (makes sense only for cyclic patches)
//preservePatches (cyclic_left cyclic_right);

//- Keep all of faceSet on a single processor. This puts all cells
// connected with a point, edge or face on the same processor.
// (just having face connected cells might not guarantee a balanced
// decomposition)
// The processor can be -1 (the decompositionMethod chooses the processor
// for a good load balance) or explicitly provided (upsets balance).
//singleProcessorFaceSets ((f0 -1));

//- Keep owner and neighbour of baffles on same processor (i.e. keep it
// detectable as a baffle). Baffles are two boundary face sharing the
// same points.
//preserveBaffles true;

method            scotch;
// method          hierarchical;
// method          simple;
// method          scotch;
// method          metis;
// method          manual;

simpleCoeffs
{
    n                ( 2 2 1 );
    delta            0.001;
}

hierarchicalCoeffs
{
    n                ( 2 2 1 );
    delta            0.001;
    order            xyz;
}

/*
scotchCoeffs
{
    processorWeights ( 1 1 1 1 );
}
*/

metisCoeffs
{
    processorWeights ( 1 1 1 1 );
}

manualCoeffs
{
    dataFile         "";
}

distributed        no;

roots              ( );
///// Is the case distributed

```

```
//distributed    yes;
//// Per slave (so nProcs-1 entries) the directory above the case.
//roots
//(
//    "/tmp"
//    "/tmp"
//);

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

以上の設定で、decomposePar を実行（メッシュ分割）すると以下のメッセージが出力される。

```
/*-----*\
|          | F ield      | OpenFOAM: The Open Source CFD Toolbox
|          | O peration  | Version: 2.4.0
|          | A nd        | Web:      www.OpenFOAM.org
|          | M anipulation|
\*-----*/
Build   : 2.4.0-f0842aea0e77
Exec    : decomposePar
Date    : Jul 12 2015
Time    : 15:00:49
Host    : "caeuser-virtual-machine"
PID     : 7268
Case    : /home/caeuser/CAE/CAE-FOAM/cyclicPatch/cyclicMappedBaffle/subCase_copy0
nProcs  : 1
sigFpe  : Enabling floating point exception trapping (FOAM_SIGFPE).
fileModificationChecking : Monitoring run-time modified files using timeStampMaster
allowSystemOperations : Allowing user-supplied system call operations

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

Create time

Decomposing mesh region0

Create mesh

Calculating distribution of cells  
Selecting decompositionMethod scotch

```
Keeping owner of faces in patches
2
(
outW_master
outW_slave
)
on same processor. This only makes sense for cyclics.
Keeping owner of faces in baffles on same processor.
```

cyclic と baffle の設定部分の  
メッセージの内容

Finished decomposition in 0.13 s

Calculating original mesh data

Distributing cells to processors

Distributing faces to processors

Distributing points to processors

Constructing processor meshes  
Reading hexRef8 data : cellLevel  
Reading hexRef8 data : pointLevel  
Reading hexRef8 data : level0Edge

```

Processor 0
  Number of cells = 4864
  Number of faces shared with processor 1 = 477
  Number of faces shared with processor 2 = 444
  Number of processor patches = 2
  Number of processor faces = 921
  Number of boundary faces = 1421
  :
  省略

```

以上の操作で並列計算用にメッシュが分割できた事になる。この分割メッシュを使って並列計算を行う事によって、エラーの発生はなくなる。

## 9-5. multiRegion の case

### 9-5-1. case 作成例


実的な応用として、multiRegion の case を作成してみる。  
解析の内容は、空気と固体間の熱移動を計算する為の case を作成する。この case は、新しいモデルでメッシュを切り直し、region も作り直す事で進める。

#### 9-5-1-1. case の作成

tutorials 内の流体・固体間の熱移動が計算できる case 「multiRegionHeater」をコピーして、これを基本に計算を進めていく。(OF-7.0 以降の場合は、tutorials 中に multiRegionHeater が無くなっており、代わりに「coolingSphere」があるので、これを使う。以後、multiRegionHeater を coolingSphere に置き換える。)

コピー方法は、6-1-2 項と同様な方法でコピーする。  
区分「heatTransfer」、solver「chtMultiRegionFoam」、case「multiRegionHeater」を選択して、この case を myTutorials フォルダにコピーする。(OF-11 の場合は、区分「foamMultiRun:複数領域のCFD」、solver「multiRegion」、case「CHT/coolingSphere」)

コピー後は、「./Allrun」を実行して、case を完成させておく。


case が完成した後は、TreeFoam 上の  ボタンをクリックして、計算結果等の不要なファイルやフォルダを削除して case を初期化しておく。

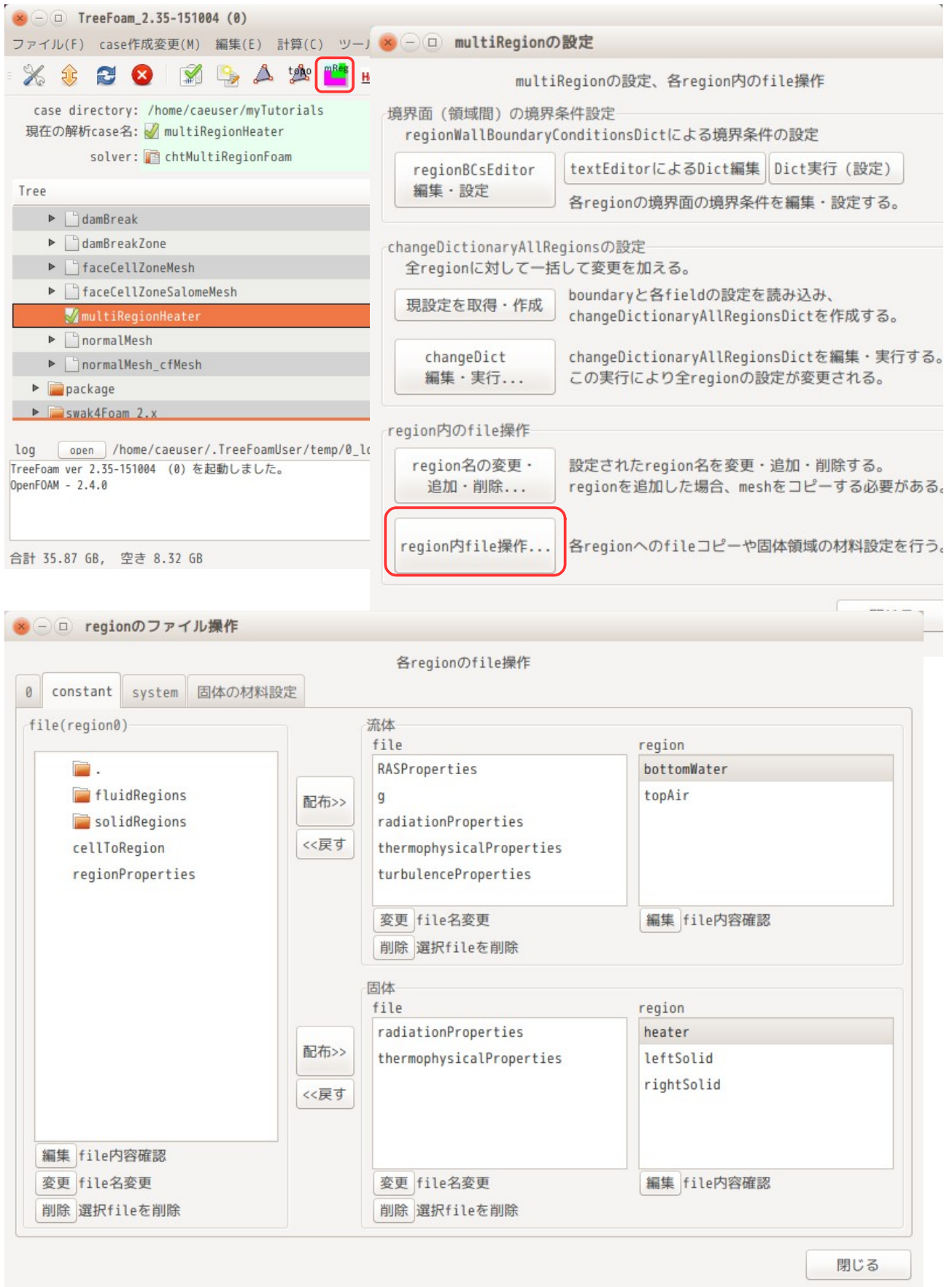
この後は、次項以降に従って、必要なファイルの保存と region を削除して multiRegion 用の masterCase となるものを作成していく。

#### 9-5-1-1-1. region 内のファイルを保存

メッシュを新しく作り直して multiRegion の case を作成する場合、region も作り直す事になる為、今の case 内にある region は、全て削除する事になる。この為、region 内にある properties や fvSchemes 等の必要なファイルを保存した上で region を削除する。

各 region 内にある必要なファイルは、「0」、「constant」、「system」フォルダ内に散らばっている為、TreeFoam では、これらの各フォルダ内に、保存用のフォルダを準備して、この中に保存する方法をとっている。

その保存方法は、以下の様に TreeFoam 上の  ボタンをクリックして、「region 内 file 操作...」ボタンをクリックして現れた「region のファイル操作」画面上で行う。  
この画面上では、region や各フォルダ (0、constant、system) の選択が容易にできるので、ファイルのコピーをスムーズに行うことができる。





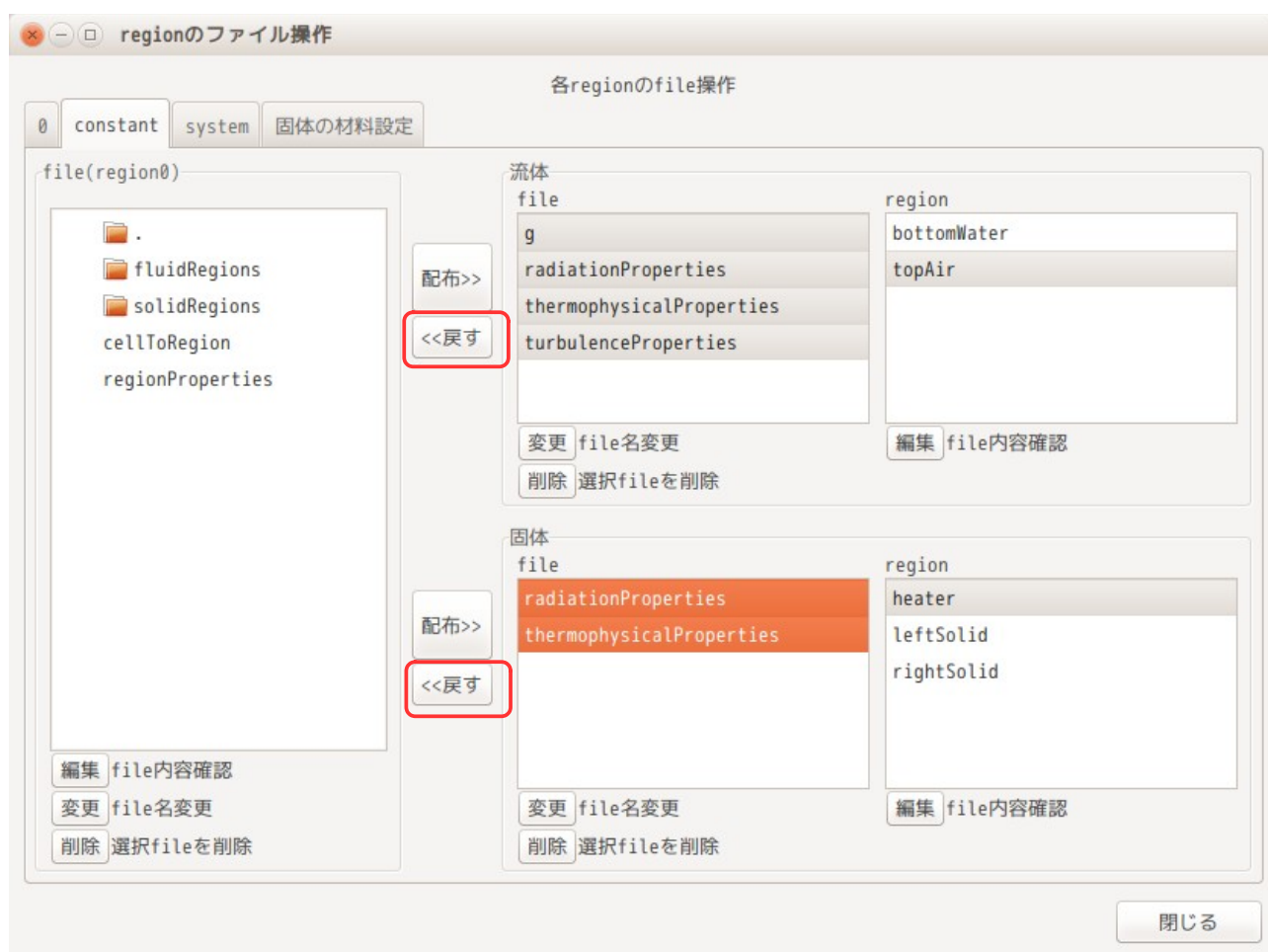
この画面が起動した時点で、region 内の必要なファイルを保存するフォルダ (fluidRegions, solidRegions) が、「0」「constant」「system」フォルダ内に作成されるので、ここに必要なファイルを以下に示す方法で保存しておく。

#### 1) constant フォルダ内のファイル保存 (実体ファイルを保存)

今回の解析では、空気と固体の熱移動を計算する為、以下のファイルが必要になる。

- 流体側：「topAir」region 内の全てのファイル。  
 tutorials では、流体の計算を層流 (laminar) で行っている為、層流の設定で今回計算する。もし、乱流の設定をするのであれば、bottomWater 内の「RASProperties」も保存しておき、乱流の設定をし直す必要がある。
- 固体側：必要なファイルは、どの region でも同じなので「heater」内の全てのファイル。  
 固体側では、材料の物性値を thermophysicalProperties で設定しており、このファイルの内容が各々の region で異なってくるはずだが、この物性値は、9-5-1-7 項で設定するので、この段階では、thermophysicalProperties のファイルが存在していれば問題ない。

これらファイルを保存する為に、「region のファイル操作」画面上で、以下の様に region とファイルを選択する。この後、流体側の「<<戻す」ボタンをクリック、同じく固体側の「<<戻す」ボタンをクリックする事で、選択したファイルが fluidRegions, solidRegions フォルダ内にコピーされる。



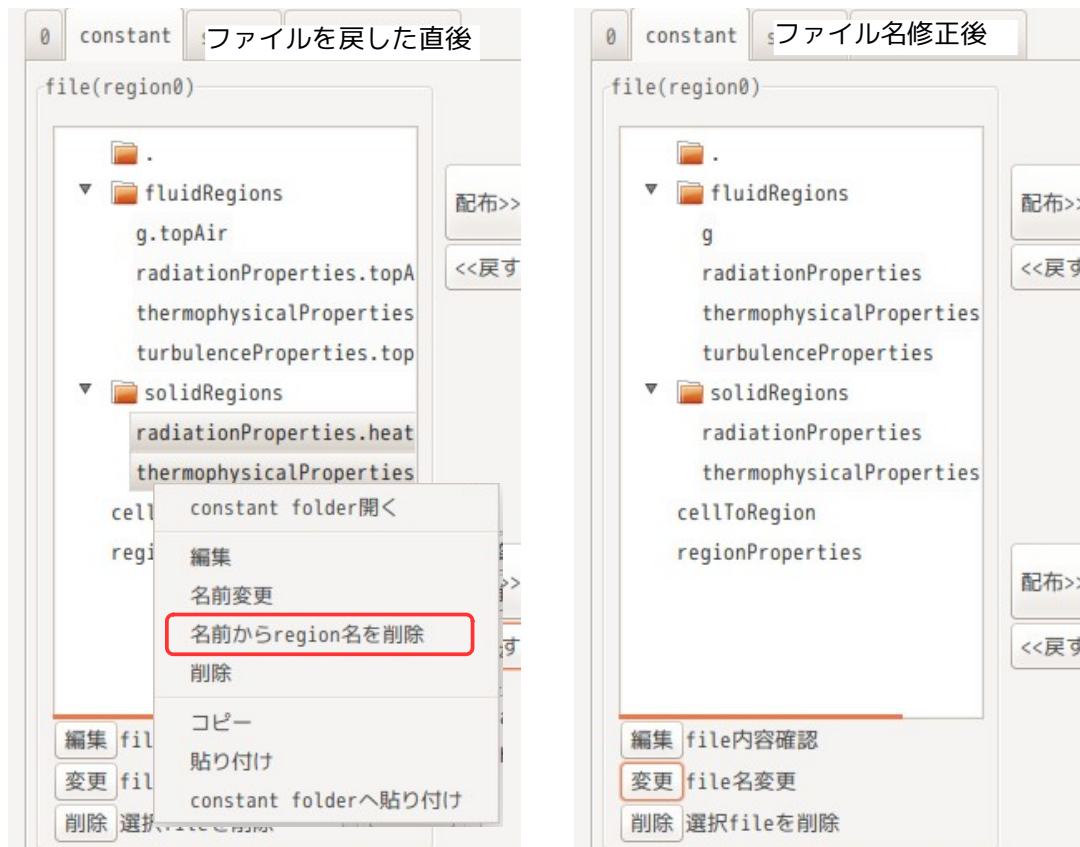
以下がファイルに戻した状態になる。fluidRegion, solidRegions フォルダ内にファイルがコピーされている。今の状態は、ファイル名が「\*\*\*\*.topAir」の様に、どの region からファイルに戻したかが、判る様になっているが、以後の操作の為に、ファイル名に付加されている region 名を削除したファイル名に変更しておく。

ファイル名から region 名を削除する方法は、ファイルを選択 (複数選択可能) 後、ポップアップメニューを表示し、「名前から region 名を削除」を選択して、region 名を削除する。

「<<戻す」ボタンでファイルに戻した場合は、必ず「\*\*\*\*.topAir」の様にファイル名に region 名が付加されてファイルがフォルダ内に戻される。逆に「配布>>」ボタンをクリックした場合は、フォル

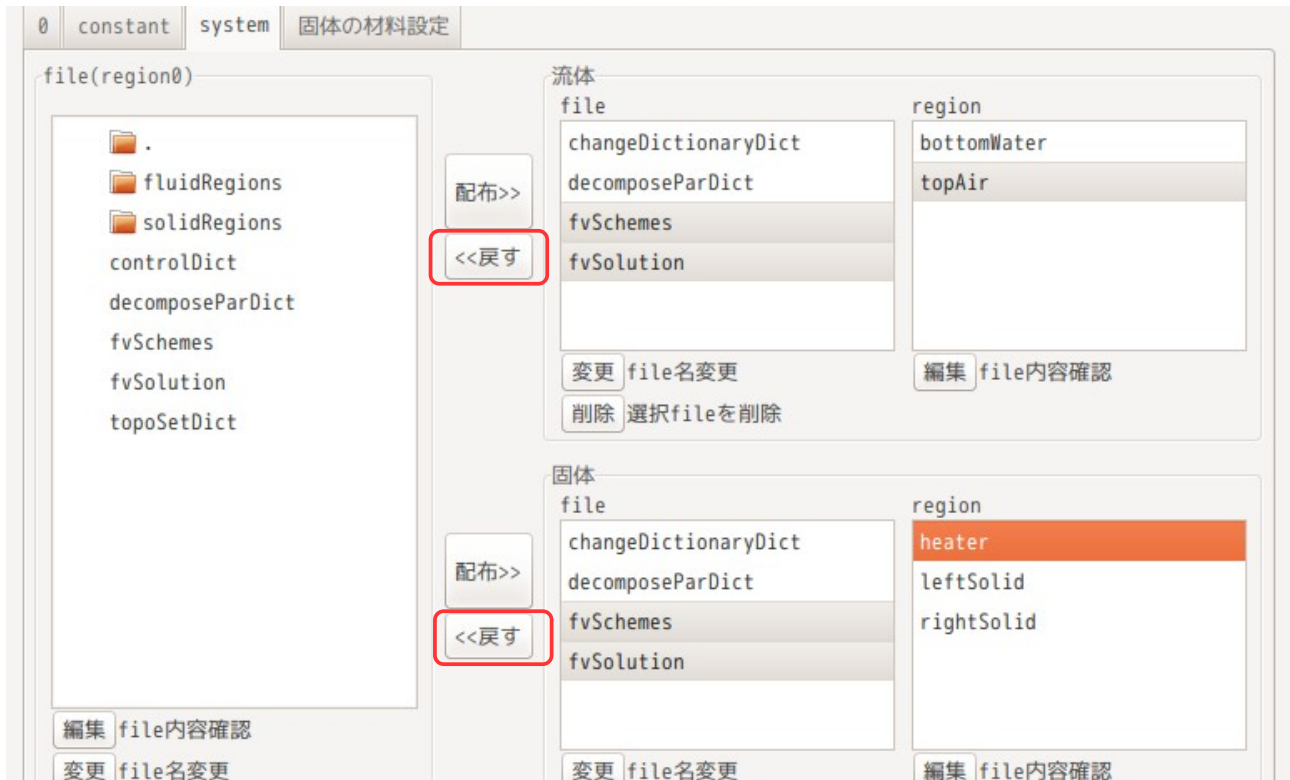
ダ内のファイルを各 region 内にコピー配布するが、「\*\*\*\*.topAir」の様に region 名が付加されたファイルは、その region のみにコピーする。region 名が付加されていないファイルは、全ての region にそのファイルをコピーして配布する。

この為、region 名を削除したファイル名に変更しておく事で、「配布>>」ボタンでそのファイルを全ての region に配布する事ができる。  
意図的に特別な region のみに配布したい時は、ファイル名に region 名を付加しておく事で実現できる。



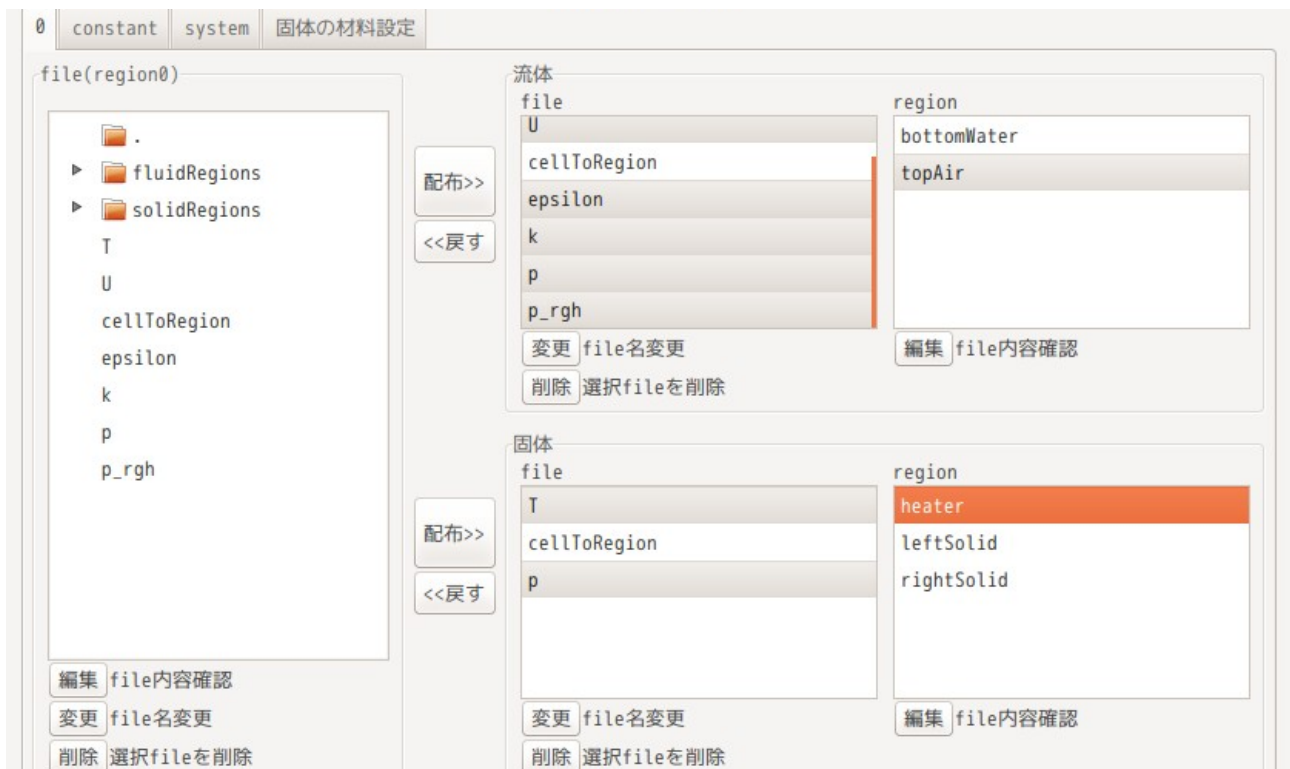
## 2) system フォルダ内のファイル保存

ここでは、必要になるファイルは、fvSchemes, fvSolution のみになるので、これを同様な方法で保存する。以下の様に選択し「<<戻す」ボタンをクリックし、constant フォルダと同様に戻したファイル名から region 名を削除したファイル名に変更しておく。



### 3) 「0」フォルダ内のファイル保存


「0」フォルダについても、topAir と heater 内の cellToRegion を除いた全てのファイルを選択し「<<戻す」ボタンをクリックして戻し、region 名を削除したファイル名に変更しておく。

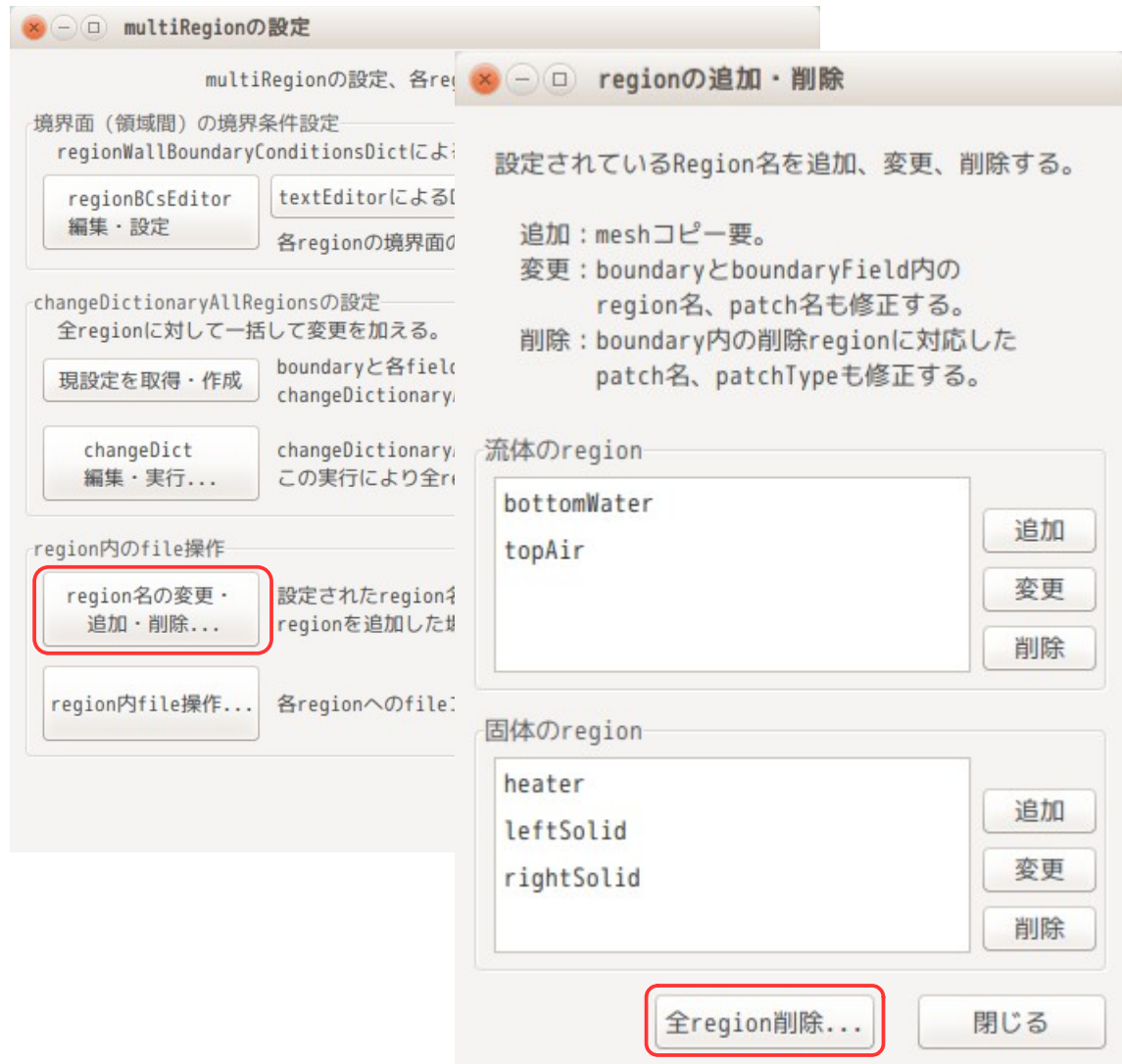



また、各 field は、caseFolder 直下にも全 field をコピーしておく。(上図参照。)ここに、コピーしなくても、領域分割できるが、コピーした方が分割時の処理が楽になる。

## 9-5-1-1-2. case 内の全 region を削除

必要なファイルが保管場所に保存できたので、ここで不要な全 region を削除する。  
今回の場合、新しいモデルでメッシュを作り直し、region も作り直す為、今の状態で存在している全ての region は不要な為、全て削除する。

全ての region を削除する方法は、TreeFoam 上で  ボタンをクリックして、「multiRegion の設定」画面を表示させる。この画面上で、以下の様に、「Region 名の変更・追加・削除」ボタンをクリックして、「region の追加・削除」画面を表示させる。この後、この画面上で「全 region 削除...」ボタンをクリックする事で、全ての region を削除する事ができる。



この操作で、この case が multiRegion の case ではなくなり、通常の case になるので、再び TreeFoam 上の  ボタンをクリックした場合、「現在の case は multiRegion の case ではありません」と言う旨の警告が出るようになる。

以上の操作で、multiRegion 作成用の case ができあがった事になる。  
この case の名称を「multiRegionAirMaster」としておき、後で汎用的に利用できる様にしておく。

## 9-5-1-2. モデルの作成（メッシュ作成）

モデルは、固体（材料：Cu）と空気との間で熱移動が発生する様な、以下のモデルを考える。



解析領域 : 80 x 40 x 40 mm

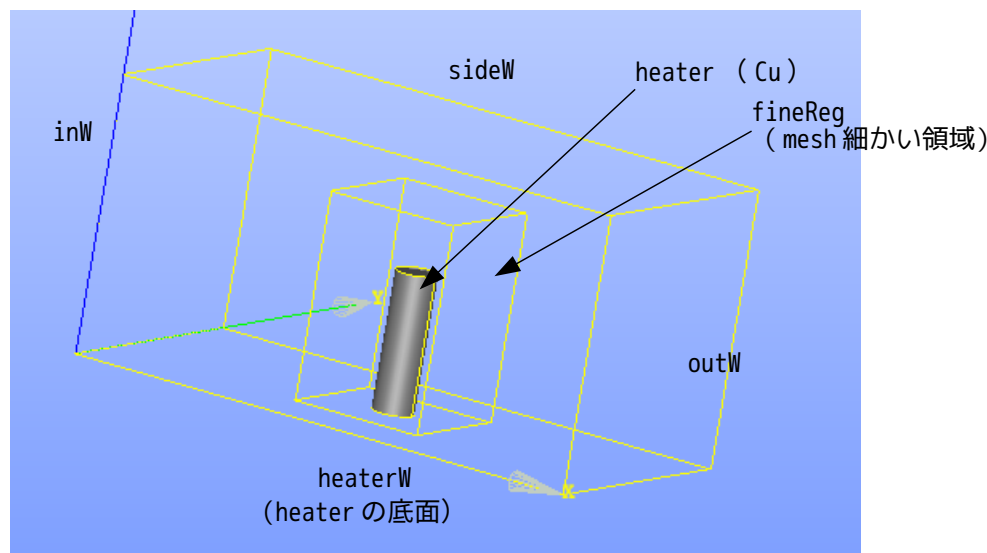
fineReg : 20 x 20 x 30 mm

heater :  $\phi 6$  x 20 mm

底面中央に配置

底面中央に配置

解析内容は、heater 底面 (heaterW) に熱流束を与え、heater を加熱させる。inW から流れを与え、heater を冷やし、outW に抜けていく。この状態の温度分布を求める解析を行ってみる。




上記モデルの為、以下の stl ファイルを作成。

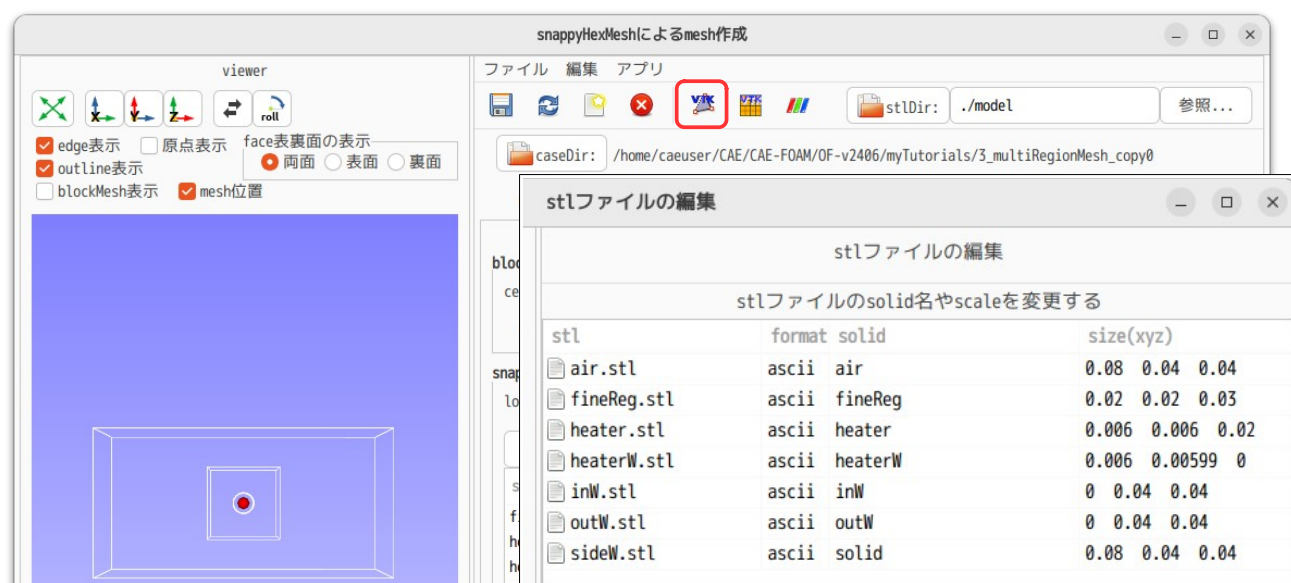
解析領域 : inW.stl, outW.stl, sideW.stl, heaterW.stl 閉じた解析領域を構成  
 その他領域 : air.stl, fineReg.stl, heater.stl

上記の中で、air.stl は、解析領域全体を air.stl としておく。(heater を抜いた領域にしない。)  
 fineReg.stl も同様に heater を抜いた形状にしない。(単純な直方体)  
 これらの stl ファイルでメッシュを作成することになる。メッシュ作成方法は、7-1 項と同様な方法でメッシュを作成する。

上記 stl ファイルは、\$TreeFoamPath/data/stlFiles/multiRegion フォルダ内に保存してあるので、ここから stl ファイルを取得できる。

まず、cavity の case をコピーし、「multiRegionAirMasterMesh」として rename し、ここでメッシュを作成する。作成した stl ファイルを「multiRegionAirMasterMesh/model」フォルダを作成し、この中に作成した全ての stl ファイルを保存しておく。

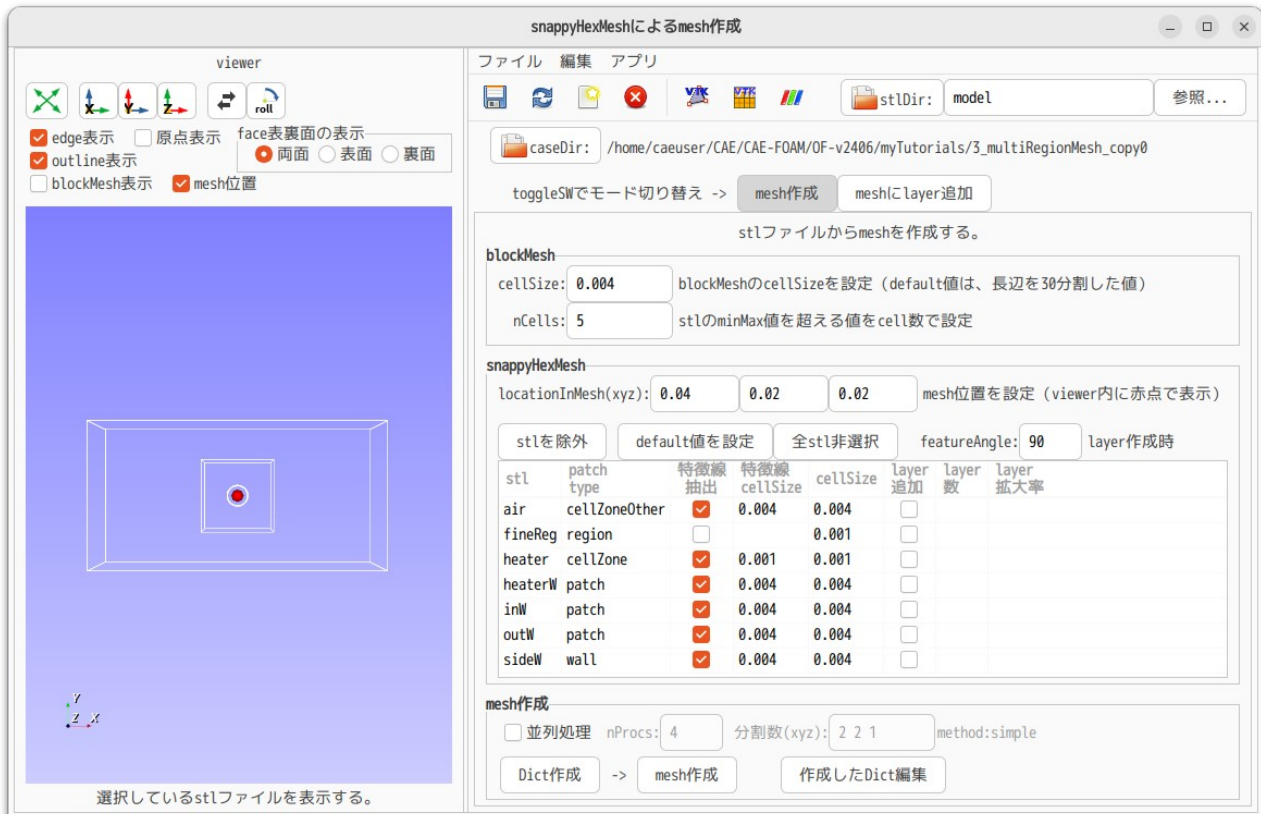
保存した stl ファイルは、「snappyHexMesh による mesh 作成」画面上の  ボタンをクリックして、scale などを確認しておく。今回は、以下で作成している。





この後、blockMeshの cellSize を設定し、各 stl の特徴線抽出、patchType、cellSize を設定する。  
以下で設定した。

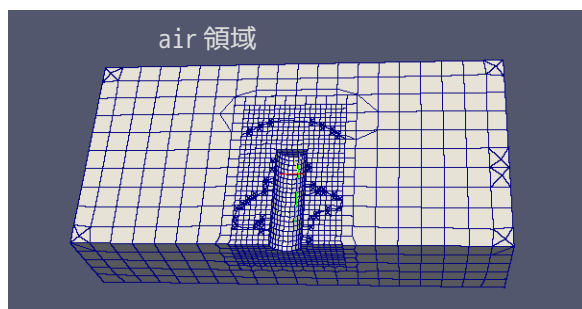
blockMeshの cellSize を入力して確定すると、snappyHexMeshの cellSize も入力した blockMesh の cellSize と同じ値に設定されるが、入力した cellSize が大きすぎる場合は、その 1/2 に設定される事がある。この場合、入力した同じ値に設定したい場合は、「default 値を設定」ボタンをクリックする事で、入力した cellSize と同じ値を snappyHexMesh 側の cellSize に設定することができる。



heater、air は、cellZone を作成する為に準備した stl になる。  
heater の patchType は、cellZone で問題ないが、air の stl は、解析領域全体 (heater を除いていない) として stl を作成しているので、そのまま patchType を cellZone に設定できない。  
この場合は、cellZoneOther として設定する。


cellZoneOther の意味は、今回のように解析領域全体を air とした時、各 cellZone を定義した後に残った領域が、air として定義する意味になる。  
air を予め、air 部分のみで定義していれば、cellZonOther とせず、通常通り cellZone とすれば良い。


以上の設定でメッシュを作成した結果が以下になる。

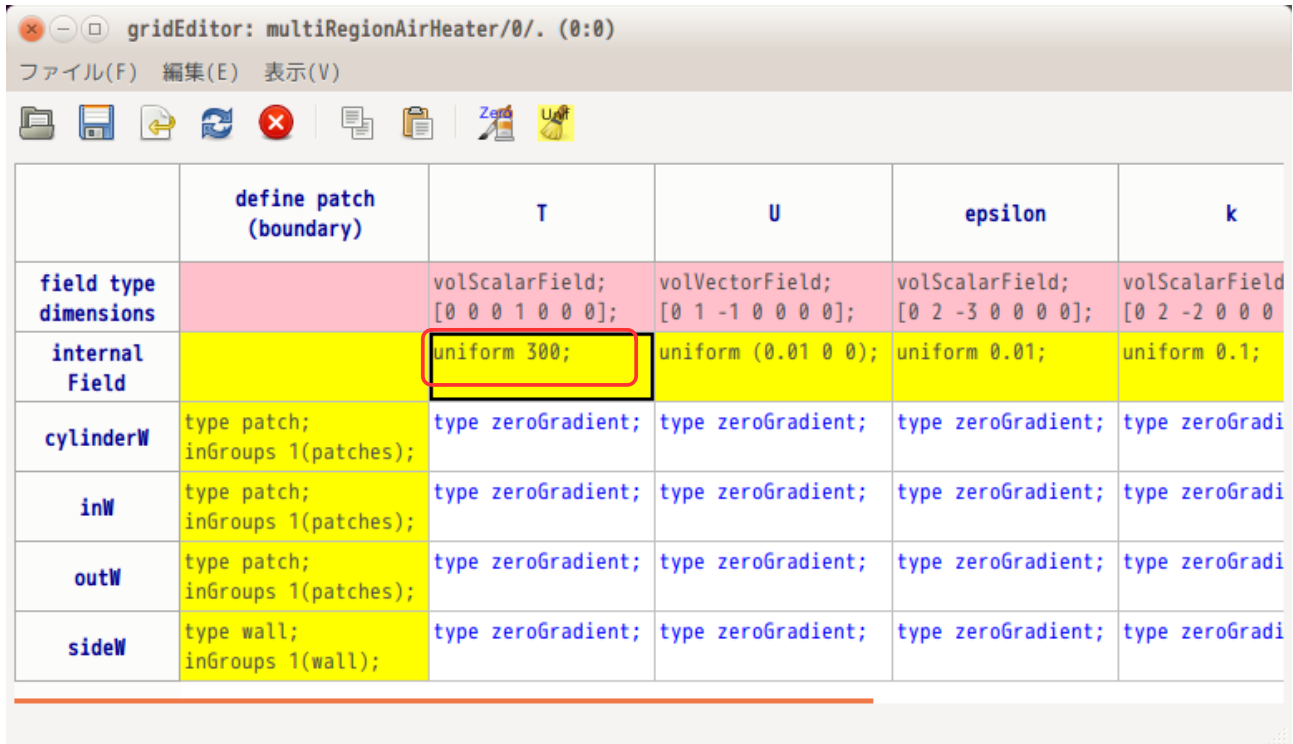


出来上がったメッシュは、「multiRegionAirMaster」case にメッシュコピーしておく。コピーは、各 field の internalField、boundaryField の内容は、全てクリアしておく。

## 9-5-1-3. 温度の初期値の設定

今回の場合、流体、固体とも初期温度 300 K として計算する。この為、今の状態で、TreeFoam 上の  ボタンをクリックして、gridEditor を起動して、温度の初期値を設定する。

領域分割前の段階で、internalField の値を  内の様に「uniform 300;」と設定することで、全 region の初期値を 300K に設定したことになる。初期値の設定は、温度 T のみ設定する。温度以外は、9-5-1-8 項で internalFields の値を変数として設定する為。

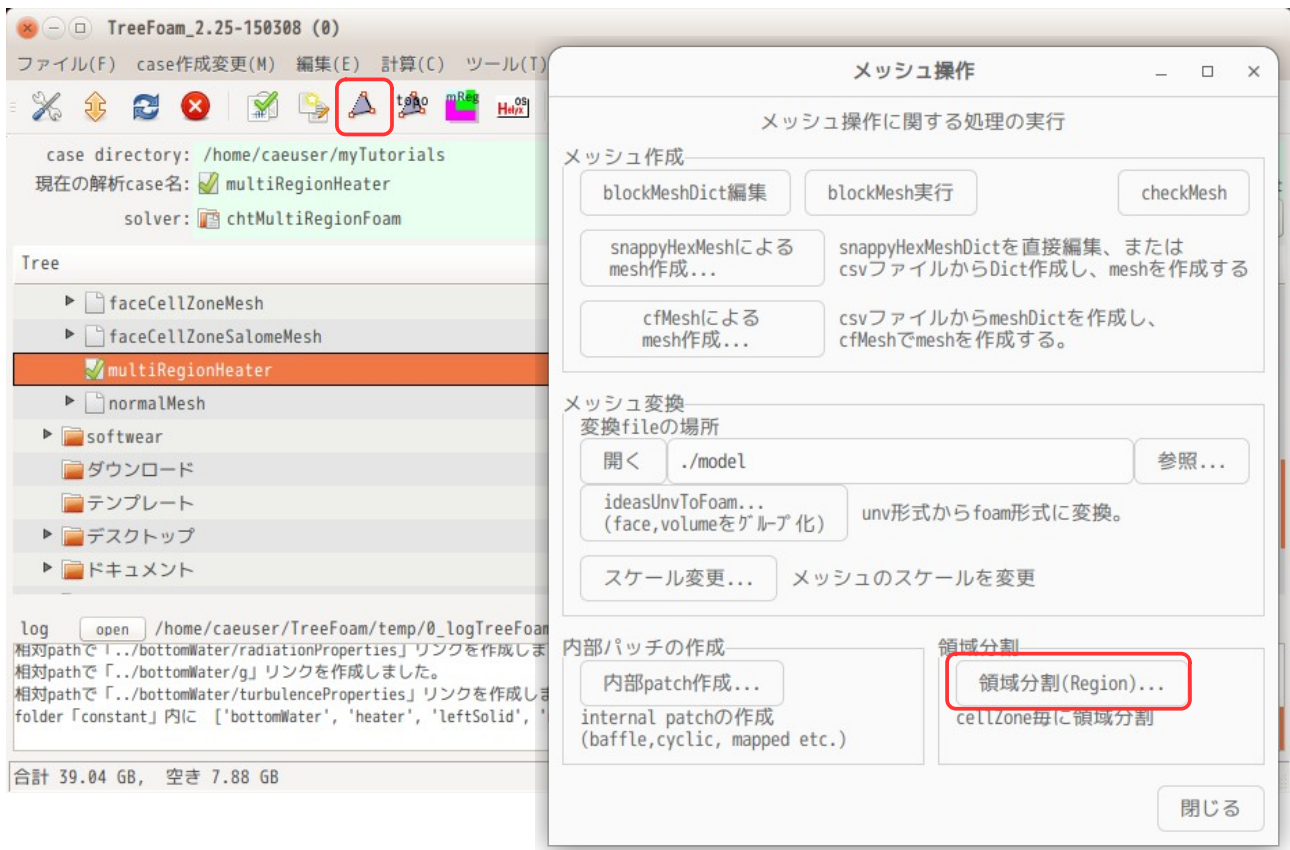


	define patch (boundary)	T	U	epsilon	k
field type		volScalarField;	volVectorField;	volScalarField;	volScalarField
dimensions		[0 0 0 1 0 0 0];	[0 1 -1 0 0 0 0];	[0 2 -3 0 0 0 0];	[0 2 -2 0 0 0
internal Field		uniform 300;	uniform (0.01 0 0);	uniform 0.01;	uniform 0.1;
cylinderW	type patch; inGroups 1(patches);	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradi
inW	type patch; inGroups 1(patches);	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradi
outW	type patch; inGroups 1(patches);	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradi
sideW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zeroGradient;	type zeroGradi

今回の場合は、region 数が少なく、また温度分布が無いので、前記した方法で問題ないが、部分的に温度設定を変えたい場合は、setFields で温度の値を設定する。この方法は、7-2-7、9-3-1 項を参照。

## 9-5-1-4. 領域分割

ここで、cellZone 毎に領域分割する。TreeFoam 上の  ボタンをクリックして、「メッシュ操作」画面を表示する。この後、「領域分割(region)...」ボタンをクリックして、領域分割する。



「領域分割(region)...」ボタンをクリックすると、領域分割が開始される。分割中の log が TreeFoam 下部のテキストボックス中に表示されるので、確認できる。この領域分割は、「splitMeshRegions cellZonesOnly」を実行している。(0F-11の場合、「splitMeshRegions -cellZones」を実行)

領域分割が終了すると、以下の画面が表示されるので、ここで、cellZone を選択して、それを流体、固体に分けて指定する。

設定前画面

cellZoneの分割

全cellZoneから「選択→」ボタンをクリックして、「fluid」「solid」を選択してください。「OK」ボタンで、必要なDictファイル等をsetupします。

全部のcellZone名

- air
- heater

選択>>

<<戻す

fluid Zone (流体)

solid Zone (固体)

選択>>

<<戻す

キャンセル OK

設定後の画面

cellZoneの分割

全cellZoneから「選択→」ボタンをクリックして、「fluid」「solid」を選択してください。「OK」ボタンで、必要なDictファイル等をsetupします。

全部のcellZone名

選択>>

<<戻す

fluid Zone (流体)

- air

solid Zone (固体)

- heater

選択>>

<<戻す

キャンセル OK

以上の操作で、どの region が流体で固体なのか明確になったので、TreeFoam 側は、この情報を元に、regionProperties を作成し、timeFolder、constant、system フォルダ内の fluidRegions と solidRegions に保存されているファイルを各 region にコピー配布する。  
この後、changeDictionary を実行して、各 region 内の boundaryField の整合をとってくれる。  
最後に、regionWallBoundaryConditions を実行して、region 間の境界条件を設定してくれる。

また、領域分割した case 名は「regCase」として、現在の case 「multiRegionAirMaster」内に作成される。この為、領域分割がうまく行かなかった場合は、regCase をフォルダ毎削除すれば、分割前の状態に戻る事ができる。

## 9-5-1-5. 領域分割後の状態


領域分割した後のファイルの構成は、fluidRegions、solidRegions 内のファイルが各 region に配布されているので、以下の構成になっている。

multuRegionAirMaster	親 case
0	
regCase	領域分割した case
0	
air	
T	
U	
epsilon	0/fluidRegions 内の field
k	
p	
p_rgh	
heater	
T	0/solidRegions 内の field
p	
constant	
air	
polyMesh	
g	
radiationProperties	constant/fluidRegions 内のファイル
thermophysicalProperties	
turbulenceProperties	
heater	
polyMesh	
radiationProperties	constant/solidRegions 内のファイル
thermophysicalProperties	
system	
air	
changeDictionaryDict	この内容により親 case の境界条件が継承される
fvSchemes	
fvSolution	
heater	
changeDictionaryDict	この内容により親 case の境界条件が継承される
fvSchemes	
fvSolution	
include	regionWallBoundaryConditions 実行により作成される
boundaryConditionsFluid	この内容により、region 間の境界条件が決定される
boundaryConditionsSolid	
variableSetting	

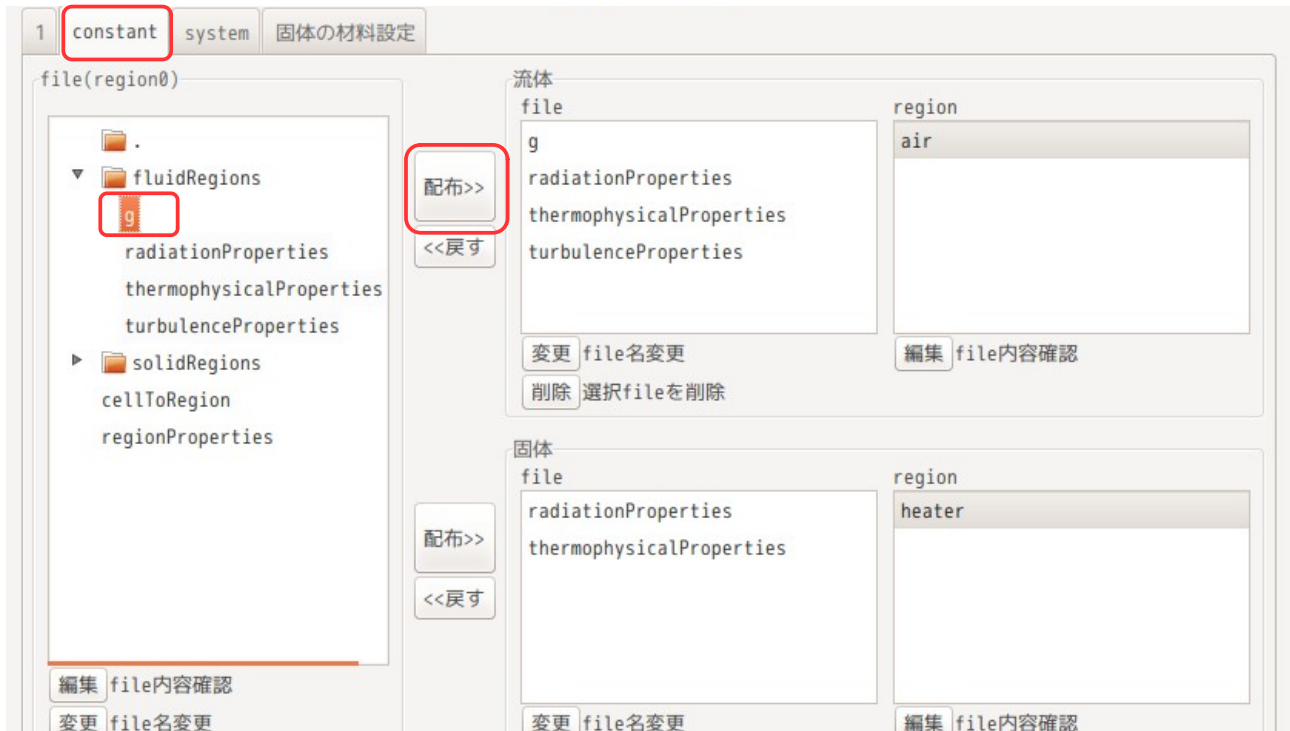
今の状態で、以下の内容が既に設定されている事になる。

- ・各 region が必要としているファイルが既に配布されている。
- ・親 case の境界条件が changeDictionary で各 region に継承されている。  
領域分割前に patch の境界条件を設定すると、この内容が region の境界条件に反映される。  
(分割前に境界条件を設定しても構わない。)
- ・region 間の境界条件が、regionWallBoundaryConditions により設定されている。  
今の設定は、default の設定の為、regionWallBoundaryConditionsDict を修正し、再度実行することで内容を修正できる。修正方法は、9-5-1-8 項参照。

## 9-5-1-6. g の設定

tutorials の重力加速度 g の方向が Y 軸方向になっているので、これを今回のモデル Z 軸方向に合わせる。方法は、まず、「regCase」が解析 case に設定されていることを確認後、TreeFoam 上の  ボタンをクリックして「region のファイル操作」画面を表示させ、「region 内の file 操作...」ボタンをクリック、「constant」タグを選択して、fluidRegions 内の「g」ファイルをダブルクリックして editor で開く。





以下が「g」を開いて、重力加速度の方向を Z 軸方向に修正した状態。修正後、上図の「配布>>」ボタンをクリックして、流体 region にコピー配布する。region「air」内の g ファイルを直接編集しても構わない。

「g」の値は修正する必要があるが、OF-v2106 以降は、「g」の保存場所が、流体 region 内から caseFolder 直下に移動しているので、上記の様な「配布>>」ボタンをクリックして配布する必要はない。

```

/*-----* 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        uniformDimensionedVectorField;
    object       g;
}
// *****


dimensions      [0 1 -2 0 0 0];
value           (0 0 -9.81);           //Z 軸方向に修正

// *****

```

#### 9-5-1-7. 固体領域の物性値設定

固体領域の物性値は、固体 region (heater) 内の thermophysicalProperties に記述するので、この内容を設定する。

設定方法は、TreeFoam 上の  ボタンをクリックして、「region 内 file 操作...」ボタンをクリックし、「region のファイル操作」画面を表示する。この画面上で「固体の材料設定」タグを選択する。以下がこの画面を表示させた状態になる。この画面上で、固体 region の材料を設定する事になる。

TreeFoamは、固体材料のデータベースを持っており、このデータベースを開いた状態が表示されている。この材料データベースの保存場所は、デフォルトでは、「~/OpenFOAM/multiRegionDB」に設定されている。この場所にデータベースが存在しない場合（初めて「regionのファイル操作」画面を表示した時）は、TreeFoamがその場所にデータベースを作成する。

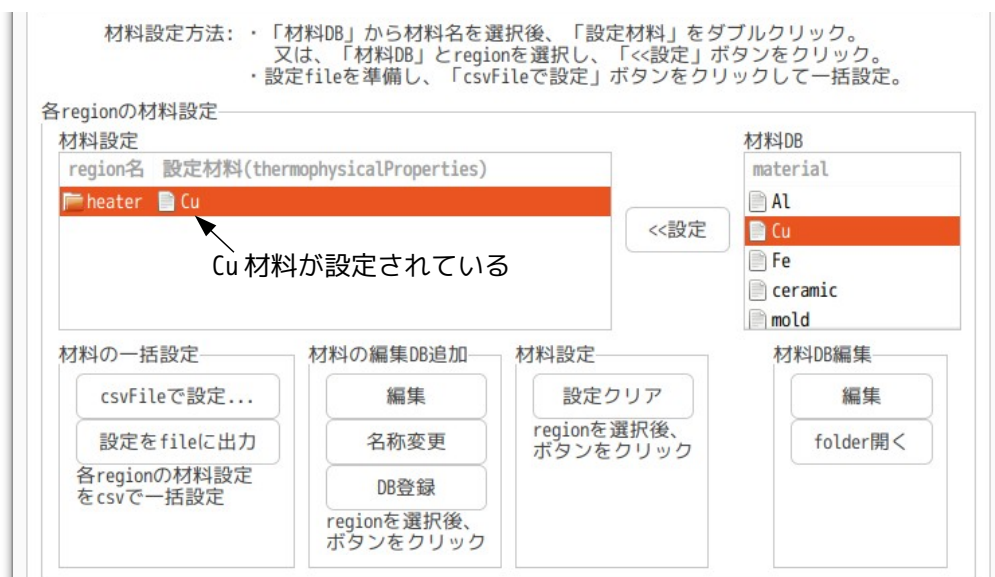
このデータベースを使えば、材料名を指定して、物性値が設定できる。デフォルトの材料は、Al, Cu, Fe, ceramic, moldが設定されているが、新たな材料をデータベースに登録する事もできる。



この画面上で、固体regionのheaterにCu材料を設定する。設定方法は、材料DB内の「Cu」を選択する。この後、材料設定内のheater行の「設定候補」欄をダブルクリックして設定する。（または、材料名とその材料を設定するregionを選択（複数選択可）して「<<設定」ボタンをクリックして設定する。）



この状態で、Cu 材が設定候補として設定されたので、この材料を region に設定する為に、以下の様に「新材料を設定」ボタンをクリックする。この操作で、Cu 材が heater に設定されたことになる。



この内容を editor で確認する場合は、heater 行の「現在の設定」欄をダブルクリックする事で、確認できる。以下が確認した結果になる。

```

/*-----* C++ *-----*/
=====
\\  F ield      OpenFOAM: The Open Source CFD Toolbox
\\  O peration  Version:  2.2.0
\\  A nd        Web:      www.OpenFOAM.org
\\  M anipulation

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

```

```

material    Cu;

thermoType
{
    type            heSolidThermo;
    mixture          pureMixture;
    transport         constIso;
    thermo            hConst;
    equationOfState   rhoConst;
    specie            specie;
    energy            sensibleEnthalpy;
}

mixture
{
    specie
    {
        nMoles       1;
        molWeight     63;
    }

    transport
    {
        kappa        372;
    }

    thermodynamics
    {
        Hf            0;
        Cp            419;
    }



    equationOfState
    {
        rho           8960;
    }
}

// ***** //

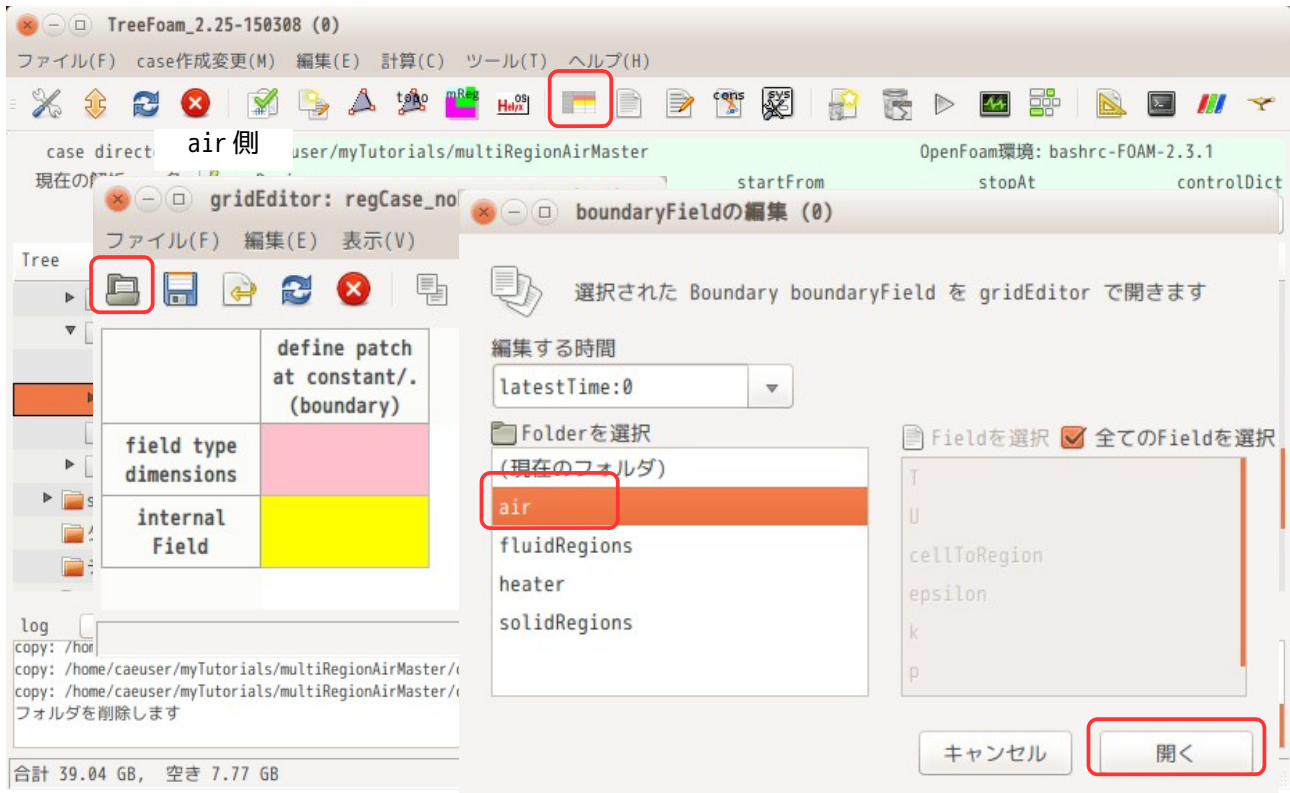
```

#### 9-5-1-8. region間の境界条件

region間の境界条件は、gridEditorで確認できるので、これで確認する。

TreeFoam上からボタンをクリックして、gridEditorを起動する。この状態は、region0の内容が表示されているので、さらに、gridEditor上からをクリックして、region名を指定してこの内容を読み込み、gridEditorで再表示させることになる。

以下は、region「air」を選択して、gridEditorを開こうとしている状態。



以下が、air と heater の境界条件になる。field 内変数が定義され、その変数を使って region 間の境界条件が設定されている。

	define patch at constant/air (boundary)	T	U	cellTo
field type		volScalarField;	volVectorField;	volScalar
dimensions		[ 0 0 0 1 0 0 0 ];	[ 0 1 -1 0 0 0 0 ];	[ 0 0 0 0
otherNames		iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	
internal Field		uniform 300;	uniform \$iniVelocity;	uniform 1
inW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zero
outW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zero
sideW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zero
air_to_hea ter	type mappedWall; inGroups 1(wall); sampleMode nearestPatchFace; sampleRegion heater; samplePatch heater_to_air;	type compressible::turbulentTemperatureCoupledBaffleMixed; value uniform \$iniTemp; Tnbr T; kappa fluidThermo; kappaName none;	type fixedValue; value uniform \$zeroVelocity;	type zero value uni



gridEditor: reqCase/0/heater (0:1)  
heater 側

	define patch at constant/heater (boundary)	T	U	cellToRegion
field type dimensions		volScalarField; [ 0 0 0 1 0 0 0 ];	volVectorField; [ 0 1 -1 0 0 0 0 ];	volScalarField; [ 0 0 0 0 0 0 0 ]
otherNames		iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;		
internal Field		uniform 300;	uniform (0.01 0 0);	uniform 0;
heaterW	type wall; inGroups 1(wall);	type zeroGradient;	type zeroGradient;	type zeroGradient;
heater_to_ air	type mappedWall; inGroups 1(wall); sampleMode nearestPatchFace; sampleRegion air; samplePatch air_to_heater;	type compressible::turbulentTemperatureCoupledBaffleMixed; value uniform \$iniTemp; Tnbr T; kappa solidThermo; kappaName none;	type zeroGradient; value uniform (0 0 0);	type zeroGradient; value uniform 0;

例として、air/T fieldの内容を editor で確認すると、以下の内容で記述されており、シンプルな記述になっている。この内容は、regionWallBoundaryConditionsDictに基づき、regionWallBoundaryConditionsを実行して設定された内容になる。(include 行と「".\*\_to\_\*"」の wildCard が追加される。)

```

/*-----* 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
OpenFOAM: The Open Source CFD Toolbox
Version: 2.3.1
Web: www.OpenFOAM.org
/*-----*/

FoamFile
{
    version      2.0;
    format       ascii;
    class        volScalarField;
    location     "0/air";
    object       T;
}

// *****

dimensions      [ 0 0 0 1 0 0 0 ];
#include "${FOAM_CASE}/include/variableSetting"
#include "${FOAM_CASE}/include/boundaryConditionsFluid"

internalField   uniform 300;

boundaryField
{
    inW
    {
        type      zeroGradient;
    }
    outW
    {
        type      zeroGradient;
    }
    sideW
    {
        type      zeroGradient;
    }
}

```


```

    ".*_to_.*"
    {
        $:wallToRegion.T;
    }
}

// ***** //

```

この設定結果は、gridEditor で確認でき、修正も可能だが、region が多数ある場合は、regionWallBoundaryConditionsDict を修正する方が容易に行う事ができる。

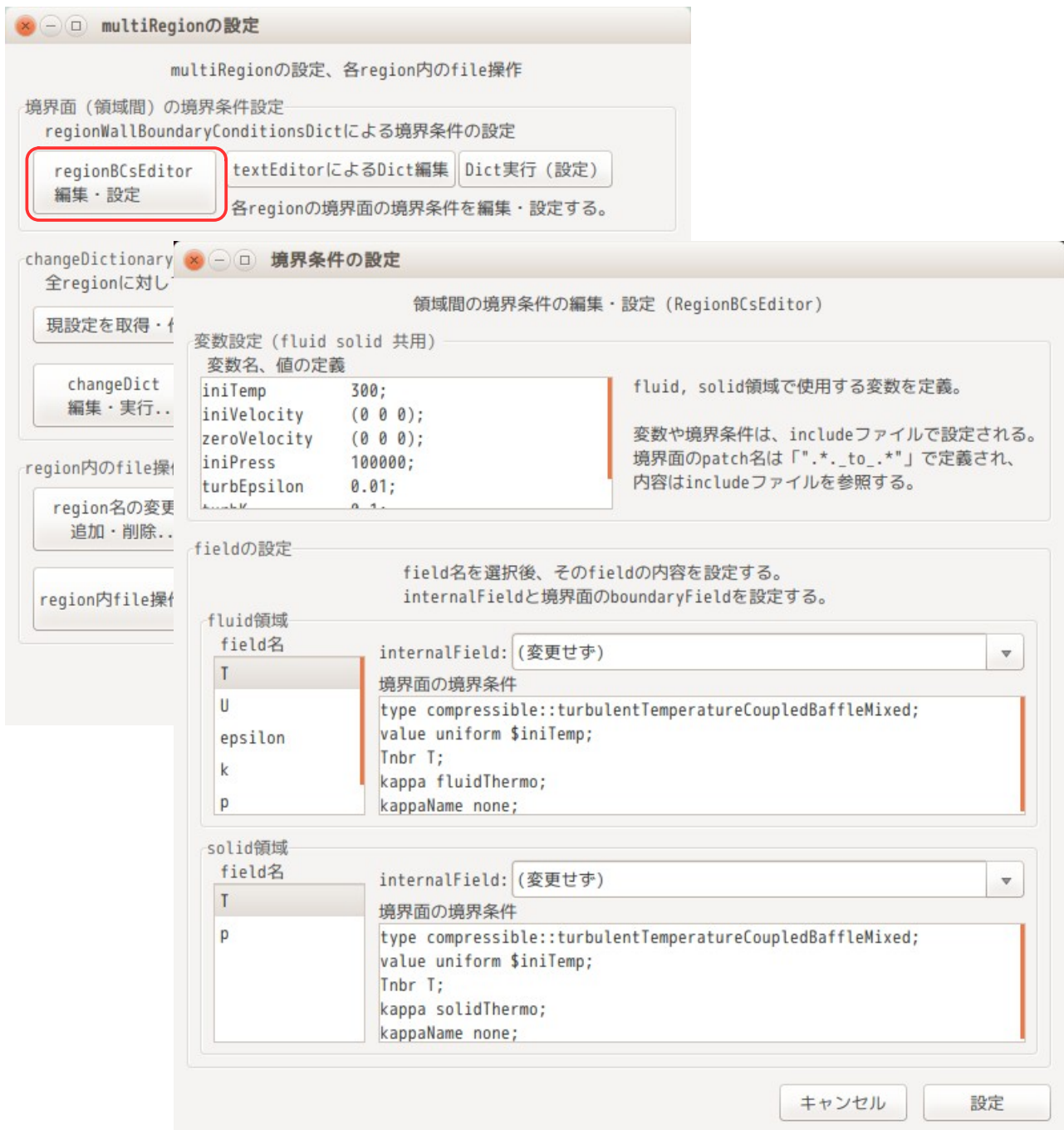
RegionWallBoundaryConditionsDict の修正方法は、TreeFoam 上の  ボタンをクリックして、「regionWallBCsEditor 編集・設定」ボタンをクリックし、「regionWallBCsEditor」を起動して、この画面上で修正する事になる。この画面上では、以下の内容が設定できる。

- ・ 定義する変数名とその値
- ・ 各 field の internalField の値を、定義した変数名の値を使うかどうかを設定
- ・ 各 field の境界条件の内容を設定

以下の画面上では、流体・固体とも T field が選択されているので、それぞれの T field の設定内容が表示されている。

内容を変更したい場合は、変更する field を選択し、その内容が表示されているテキストボックス内を直接編集する。編集後「設定」ボタンをクリックする事で、編集内容が regionWallBoundaryConditionsDict に反映され、各 field 内容も書き換えられ、field の内容が再設定される事になる。

k、epsilon の初期値については、今回は、層流で計算するので、default の設定で問題ない。もし、乱流 (k-ε) で計算する様であれば、ここで修正しておく。修正方法は、「変数名・値の定義」欄の「turbK」と「turbEpsilon」の値を直接修正する。



また、同じ画面上で、以下の「textEditorによるDict編集」ボタンをクリックした場合は、editorでregionWallBoundaryConditionDictが編集できる。



以下が、regionWallBoundaryConditionsDict を editor で開いた内容になる。  
 この内容を直接編集しても構わない。編集した内容を各 field に反映する為には、上図の「Dict 実行（設定）」ボタンをクリックして、各 field にその内容を反映させる。

```

/*-----* C++ *-----*/
|=====|
| \ \ \ \ | F i e l d | OpenFOAM: The Open Source CFD Toolbox
| \ \ \ \ | O p e r a t i o n | Version: 2.3.0
| \ \ \ \ | A n d | Web: www.OpenFOAM.org
| \ \ \ \ | M a n i p u l a t i o n |
/*-----*/

FoamFile
{
    version      2.3;
    format       ascii;
    class        dictionary;
    location     "";
    object       regionWallBoundaryConditionsDict;
}
// ***** //

//directory of include files
// Below include 3 files are created at next below directory.
// If its directory does not exist, its directory is created.
// - variableSetting
// - boundaryConditionsFluid
// - boundaryConditionsSold
//
includeDir      "${FOAM_CASE}/include";

// variable setting
variableSetting
{
    iniTemp      300;
    iniVelocity  (0 0 0);
    zeroVelocity (0 0 0);
    iniPress     100000;
    turbEpsilon  0.01;
    turbK        0.1;
}

//internal fields setting for variableName
// If field does not exist in region, is not apply, so is not change.
// And if internalField type is nonuniform, should not be applied.
//
// example:
//      U      iniVelocity;
//
// InternalField of U is set 'internalField uniform $iniVelocity;'.
//
internalFields
{
    fluidRegions
    {
        U      iniVelocity;
        epsilon  turbEpsilon;
        k      turbK;
        p      iniPress;
        p_rgh  iniPress;
    }
    solidRegions
    {
        p      iniPress;
    }
}

```

```
//boundary conditions of walls between regions
// If the field does not exist in region, its boundary condition is not changed.
//
regionWallBoundaryConditions
{
    fluidRegions
    {
        T
        {
            type compressible::turbulentTemperatureCoupledBaffleMixed;
            value uniform $iniTemp;
            Inbr T;
            kappa fluidThermo;
            kappaName none;
        }

        U
        {
            type fixedValue;
            value uniform $zeroVelocity;
        }

        epsilon
        {
            type compressible::epsilonWallFunction;
            value uniform $turbEpsilon;
        }

        k
        {
            type compressible::kqRWallFunction;
            value uniform $turbK;
        }

        p
        {
            type calculated;
            value uniform $iniPress;
        }

        p_rgh
        {
            type fixedFluxPressure;
            value uniform $iniPress;
            gradient uniform 0;
        }
    }
    solidRegions
    {
        T
        {
            type compressible::turbulentTemperatureCoupledBaffleMixed;
            value uniform $iniTemp;
            Inbr T;
            kappa solidThermo;
            kappaName none;
        }

        p
        {
            type calculated;
            value uniform $iniPress;
        }
    }
}
```



```
//special setting
// If you want to set the special boundaryCondition to special region wall,
// you can get it using wild card of patch name from below setting.
//
// This setting is that heat flux does not flow between *Solid and topAir.
//
// "topAir_to_*.Solid"
//{
//    fluidRegions
//    {
//        T
//        {
//            type zeroGradient;
//        }
//    }
//}
// ".*.Solid_to_topAir"
//{
//    solidRegions
//    {
//        T
//        {
//            type zeroGradient;
//        }
//    }
//}

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

regionWallBoundaryConditionsDict は、region 間の境界条件を「".\*\_to\_.\*"」の様な wildCard (正規表現) で設定している。

このような方法を取った事で、上記リストの最後でコメントアウトしている場所 (「special setting」内) で使っている様な wildCard を追加する事により、特別な region 間が指定でき、これに特別な境界条件を設定する事もできるようになっている。

regionWallBoundaryConditionsDict を直接 editor で編集する場合、これが実現できるメリットがある。

#### 9-5-1-9. 境界条件の設定

region 間以外の境界条件が未設定の為、ここで設定する。

境界条件を設定する patch とその内容は、今回の場合、以下になる。

流体側 (空気)

inW	流入面：温度 300 K、速度 x 方向に 1 m/s を与える
outW	流出面：圧力 1e-5 Pa を設定
saideW	壁

固体側 (Cu)

heaterW	一定の温度勾配 $\nabla T$ 500 K/m を与える 材料が Cu の為、 $\mathbf{q} = -\kappa \cdot \nabla T = -186\text{e}3 \text{ W/m}^2$ の熱流束を与える事になる
---------	--

k, epsilon の設定については、今回は層流で計算する為、default の設定で問題ない。

最終的に、流体・固体に分けて以下の様に設定した。

gridEditor: regCase/0/air (0:0) air側

	define patch at constant/air (boundary)	T	U	
field type dimensions		volScalarField; [ 0 0 0 1 0 0 0 ];	volVectorField; [ 0 1 -1 0 0 0 0 ];	volScalarFi [ 0 2 -3 0
otherNames		iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300 iniVelocity zeroVelocit iniPress 10 turbEpsilon turbK 0.1;
internal Field		uniform 300;	uniform \$iniVelocity;	uniform \$tu
inW	type patch; inGroups 1(patch);	type fixedValue; value uniform 300;	type fixedValue; value uniform (1.0 0 0);	type zeroGr
outW	type patch; inGroups 1(patch);	type inletOutlet; value uniform 300; inletValue uniform 300;	type inletOutlet; value uniform (1 0 0); inletValue uniform (0 0 0);	type zeroGr
sideW	type wall; inGroups 1(wall);	type zeroGradient;	type fixedValue; value uniform \$zeroVelocity;	type zeroGr
air_to_heater	type mappedWall; inGroups 1(wall); sampleMode nearestPatchFace; sampleRegion heater; samplePatch heater_to_air;	type compressible::turbulentTemperatureCoupledBaffleMixed; value uniform \$iniTemp; Tnbr T; kappa fluidThermo; kappaName none;	type fixedValue; value uniform \$zeroVelocity;	type compre value unifo

gridEditor: regCase/0/air (0:0) air側

	epsilon	k	p	p_rgh
field type dimensions	volScalarField; [ 0 2 -3 0 0 0 0 ];	volScalarField; [ 0 2 -2 0 0 0 0 ];	volScalarField; [ 1 -1 -2 0 0 0 0 ];	volScalarField; [ 1 -1 -2 0 0 0 0 ];
otherNames	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;
internal Field	uniform \$turbEpsilon;	uniform \$turbK;	uniform \$iniPress;	uniform \$iniPress;
inW	type zeroGradient;	type zeroGradient;	type calculated; value uniform \$iniPress;	type fixedFluxPressure; value uniform \$iniPress; gradient uniform 0;
outW	type zeroGradient;	type zeroGradient;	type calculated; value uniform \$iniPress;	type fixedValue; value uniform \$iniPress;
sideW	type zeroGradient;	type zeroGradient;	type calculated; value uniform \$iniPress;	type fixedFluxPressure; value uniform \$iniPress; gradient uniform 0;
air_to_heater	type compressible::epsilonWallFunction; value uniform \$turbEpsilon;	type compressible::kqRWallFunction; value uniform \$turbK;	type calculated; value uniform \$iniPress;	type fixedFluxPressure; value uniform \$iniPress; gradient uniform 0;

gridEditor: regCase/0/heater (0:0)

heater 側

ファイル(F) 編集(E) 表示(V)

	define patch at constant/heater (boundary)	T	p
field type dimensions		volScalarField; [ 0 0 0 1 0 0 0 ];	volScalarField; [ 1 -1 -2 0 0 0 0 ];
otherNames		iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;	iniTemp 300; iniVelocity (0 0 0); zeroVelocity (0 0 0); iniPress 100000; turbEpsilon 0.01; turbK 0.1;
internal Field		uniform 300;	uniform \$iniPress;
cylinderW	type patch; inGroups 1(patch);	type fixedGradient; gradient uniform 500; value uniform \$iniTemp;	type calculated; value uniform \$iniPress;
heater_to_ air	type mappedWall; inGroups 1(wall); sampleMode nearestPatchFace; sampleRegion air; samplePatch air_to_heater;	type compressible::turbulentTemperatureCoupledBaffleMixed; value uniform \$iniTemp; Tnbr T; kappa solidThermo; kappaName none;	type calculated; value uniform \$iniPress;

今回、前記したように、領域分割後、境界条件を各 region 毎に設定したが、この境界条件は、領域分割前に設定しても問題ない。むしろこの方が region 毎に設定する必要が無いので、案に行える。

#### 9-5-1-10. 計算開始

以上で全ての設定が終了したので、計算開始させる。設定が誤っていないか確認する為に、まず、▶ ボタンをクリックして、シングルコアで計算させる。問題なく計算できるようであれば、6-2-7 項と同様に、⌘ ボタンをクリックして、並列計算の設定を行う。以下の設定は、scotch で mesh を 4 並列用に分割する設定。

multiRegion の場合、メッシュを分割する為の decomposeParDict が各 region 毎に存在する。

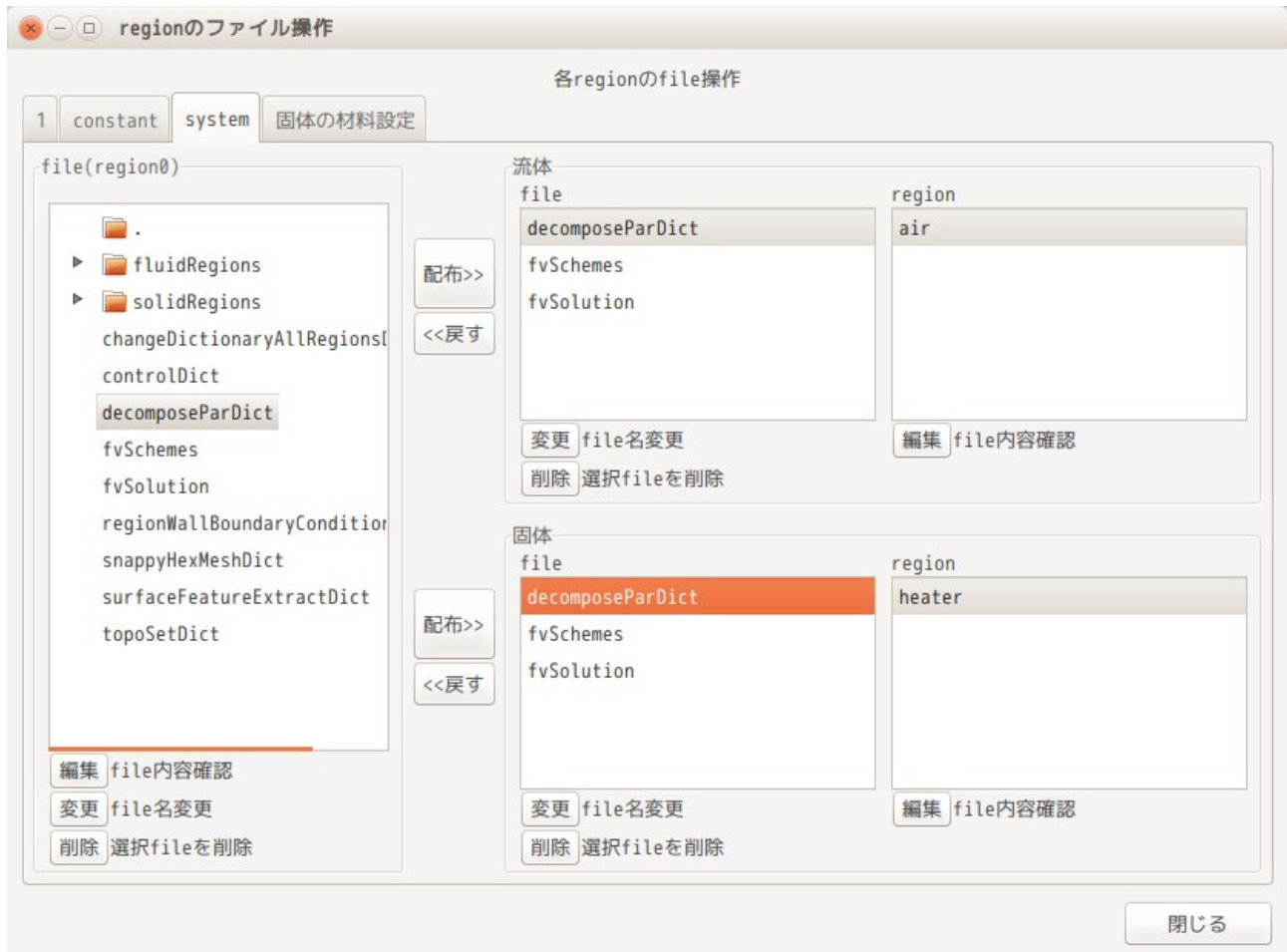
⌘ ボタンをクリックして「並列計算」画面を表示した段階で、各 region 内の decomposeParDict の存在を確認し、存在しない場合は、default の decomposeParDict を各 region 内に作成する。

並列数や、mesh 分割方法を修正するようであれば、「並列計算」画面上で、直接修正し、「nCPU, method 設定」ボタンをクリックする。これにより、全ての region 内の decomposeParDict が修正される。この修正は、decomposeParDict 内の、nCPU と method のみ書き換える為、特別な region に特別な decomposeParDict を作成しても、この内容 (nCPU と method 以外) は保存されることになる。




下図は、「Dict 確認・編集」ボタンをクリックして「region間のファイル操作」画面を表示させた状態。下図の画面の様に、decomposeParDict は、今回の場合、3箇所（選択している場所）存在している。

特別な region に特別な decomposeParDict を設定する場合は、region を選択して、その region 内の decomposeParDict を開き、編集する事になる。



decomposeParDict を完成させた後は、6-2-7 項と同様に「mesh 分割」ボタンでメッシュを processor 毎に分割し、「並列計算開始」ボタンで、計算を開始させる。

計算終了後、「結果の再構築」ボタンで各 processor 毎の計算結果を結合し case フォルダ直下に保存する。計算結果を再構築後は、各 processor に散らばっている結果データを削除しておく。（削除方法は、6-2-7 項を参照。）

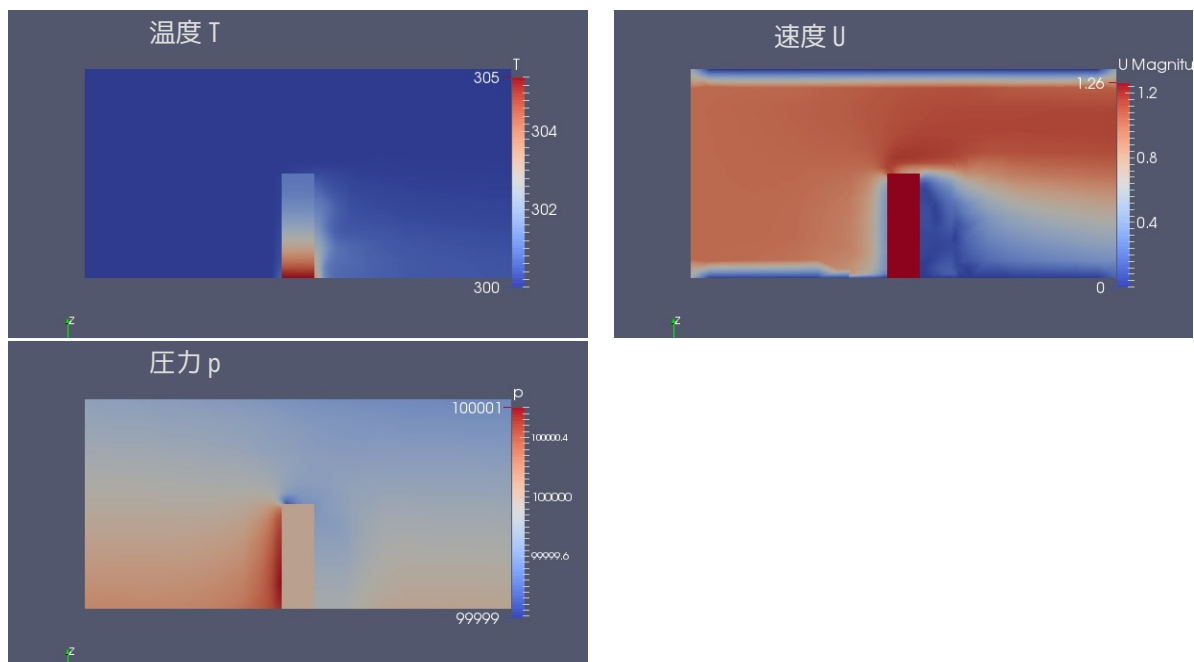
結果を確認する為に、TreeFoam 上の  ボタンをクリックして paraFoam を起動するが、今回の場合、下図の様に「paraFoam -builtin」のオプションを選択して paraFoam を起動した。





multiRegion の case 場合、paraFoam をオプション無しで起動すると、起動後、各 region を読み込む必要があり、region が多くあると、手間がかかる。「paraFoam -builtin」で paraFoam を起動すると、全 region の全 field を読み込む設定で paraFoam が起動するので、結果が直ぐに確認できるが、欠点もあるので、状況に合わせて確認する。(9-5-1-11 項を参照)

下図は、1s 後の計算結果を paraFoam で確認した結果になる。




multiRegion の計算ができたが、今の状態は、流体「air」にレイヤが付いていない状態。

#### 9-5-1-11. paraFoam による計算結果の確認について

multiRegionCase の場合、計算結果が各 region に散らばっているので、通常通りに paraFoam を起動すると、各 region の結果を読み込む必要が生じ、region が多くある場合は、手間が増えてしまう。この手間を省く方法として、TreeFoam では、以下の様に「-builtin」オプション付きで paraFoam を起動する方法を準備している。

##### < 「-builtin」 オプション付きで paraFoam を起動して結果を確認 >

この方法は、最も簡単に結果を確認できる方法になる。その方法は、TreeFoam 上の  ボタンをクリックして、表示された「paraFoam の起動オプション」画面上で、下図の様に「-builtin」オプションを選択して「OK」ボタンをクリックする。これにより、paraFoam が「-builtin」オプション付きで起動する。



この方法で paraFoam を起動すると、各 region、各 field のデータを読み込んだ状態で paraFoam が起動する。

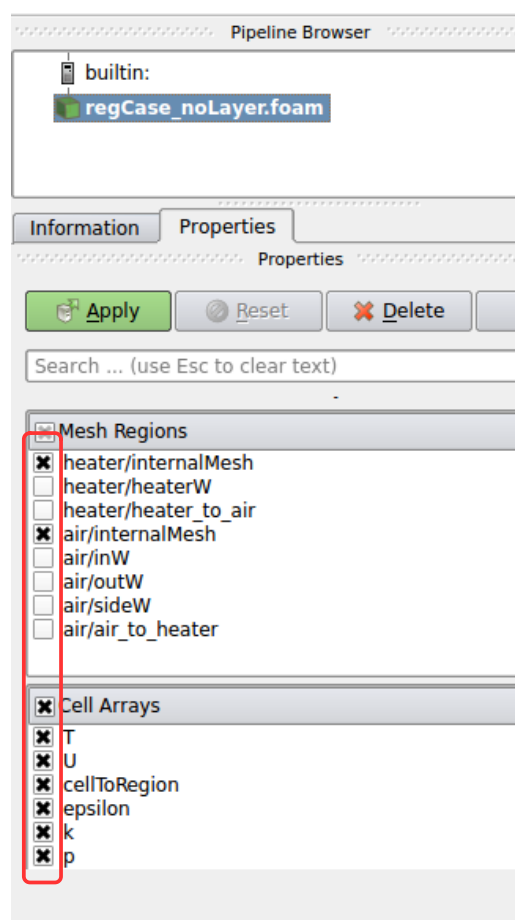
右図は、paraFoam が起動した直後の状態を示しているが、この様に各 region、各 field が既にチェックされた状態で paraFoam が起動している。

この為、後は緑色の「Apply」ボタンをクリックするだけで、結果が確認できる。

ただし、この方法（「-builtin」オプション付きで paraFoam を起動）は、field 内の実データしか読めない。

今回の case の場合、計算前の境界条件を設定している field には、9-5-1-8 項で記している様に、`#include` を使って `$iniTemp` 等の変数を定義しているが、この変数が読めなくなってしまう、エラーが発生する。このような場合は、paraFoam をオプション無しで起動して読み込む。

計算結果が保存された field には、変数はなく、実データが入っているので、そのまま読み込む事ができる。



#### 9-5-1-12. 流体領域にレイヤを追加

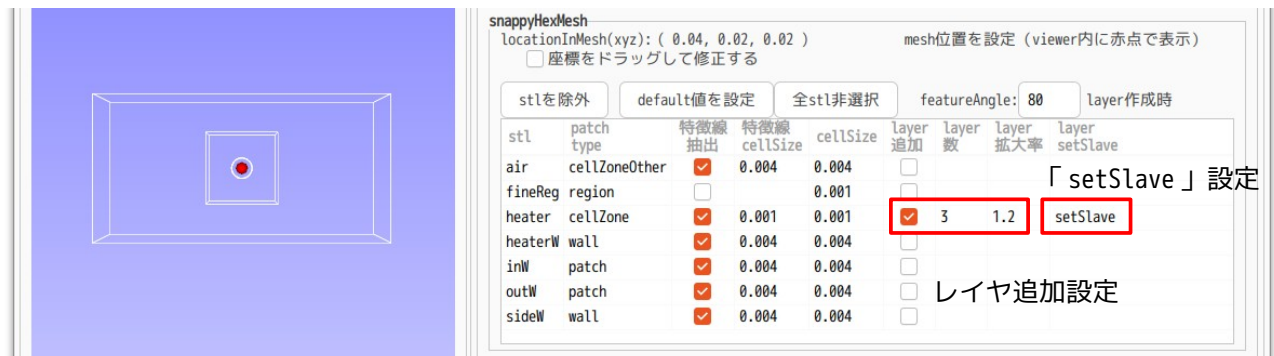
流体・固体の境界面の流体側にレイヤを追加する事を考えてみる。この部分にレイヤを追加する方法は、以下の2種類ある。

- ・領域分割する前のメッシュ作成時にレイヤを追加する。
- ・領域分割後の流体 region にレイヤを追加する。

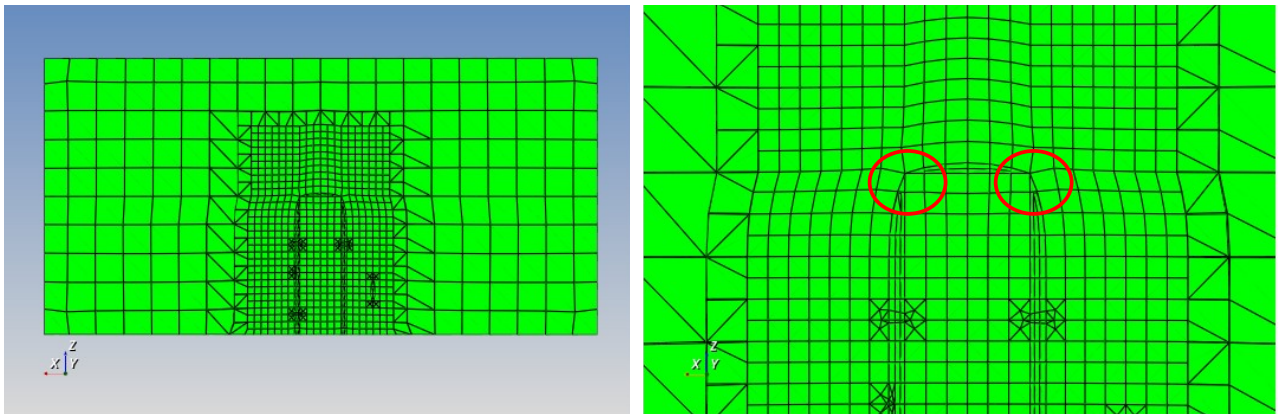
##### 1) メッシュ作成時にレイヤを追加する場合

メッシュ作成時にレイヤを追加する場合は、以下のように設定する。レイヤを追加する場所を「heater」cellZoneに設定しているので、「setSlave」を設定している。この設定により、「heater」cellZoneのslave側（流体側）にレイヤが追加される。

(この設定をしなかった場合は、「heater」cellZone 側にレイヤが追加される。)  
この部分が通常のレイヤ設定と異なる部分になる。



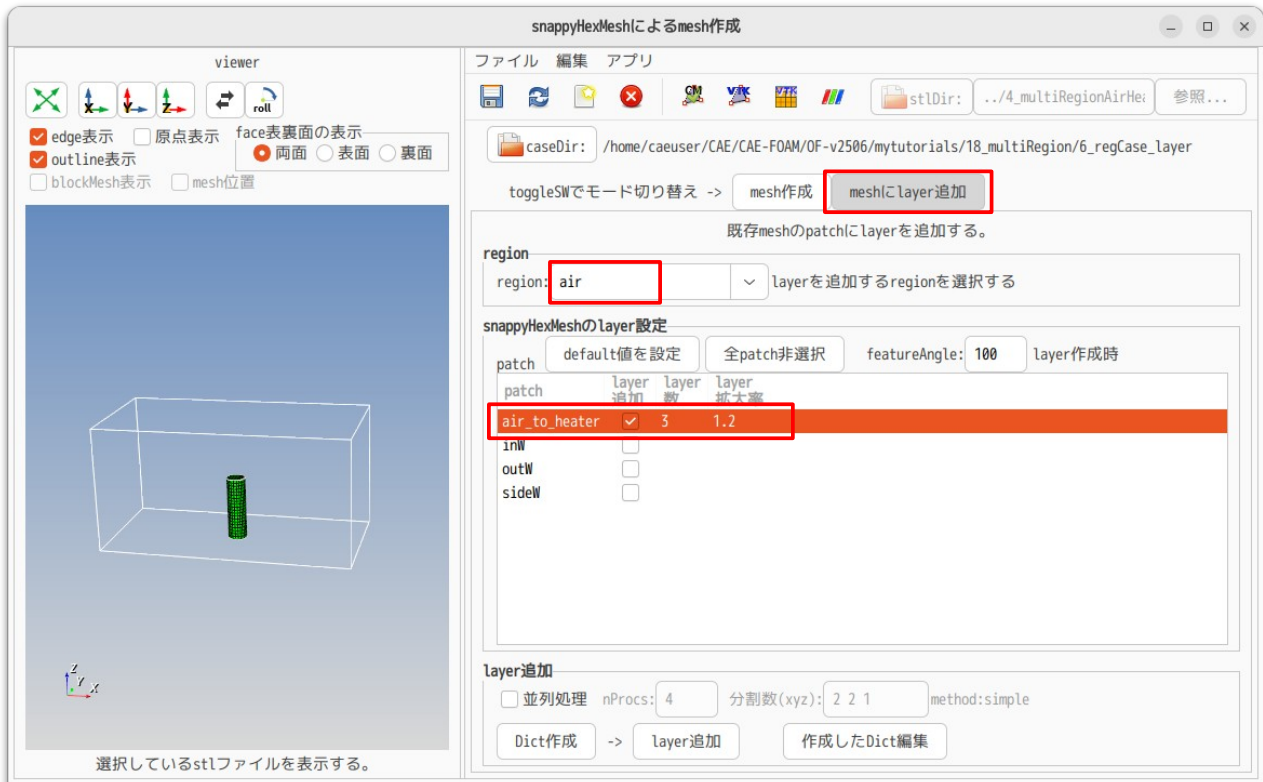
この設定でメッシュ作成した結果が以下になる。  
メッシュ作成時にレイヤを作成すると、featureAngle「80」の設定でメッシュを作成しているので、角部にレイヤが付かない(○内)欠点がある。 7-1-5項を参照。



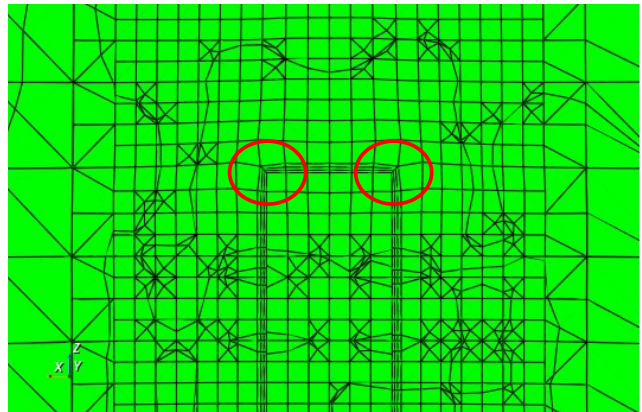
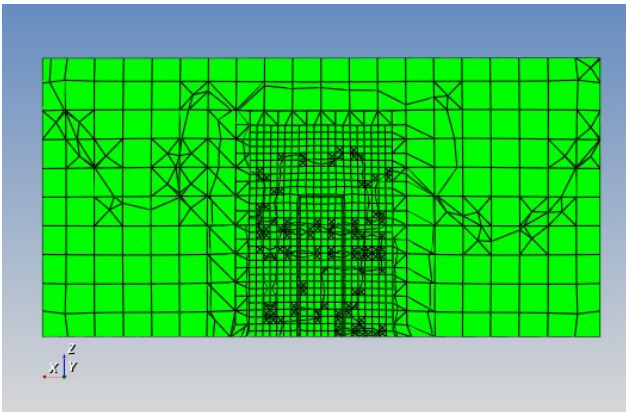
これを避けるためには、次の領域分割後の multiRegionCase の「air」region のメッシュにレイヤを追加する。

## 2) 領域分割後の流体 region にレイヤを追加する場合

領域分割後の multiRegionCase を解析 case に設定し、「snappyHexMesh による mesh 作成」画面上で、「mesh に layer 追加」ボタンをクリックして画面を切り替え、region「air」を選択。  
このあと、layer 追加場所を選択して、layer の設定を行う。



この設定で、できあがった layer が以下になる。



できあがったメッシュに layer を追加すると featureAngle を「100」に設定することができる為、角部にも layer を追加することができる。





## 9-6. 計算サーバを接続する場合

計算サーバを TreeFoam 上で接続し、サーバ内の folder を TreeFoam 上に表示させる事ができる。これにより、TreeFoam 上でサーバ内 folder を操作する事ができる。

gridEditor に関しては、サーバ対応させているので、容易にサーバ内 case を gridEditor で編集できる。また、FOCUS については、サーバに計算させる為の Job の編集や Job 投入実行を容易に行う事ができる。

また、サーバを使って計算するという事は、モデルの規模が大きくなっている場合が多い。モデルの規模が大きくなっている場合（要素数 100 万要素以上）は、メッシュ作成を含めて「binary」形式で作成した方が OpenFOAM や TreeFoam 側の処理も早くなるので、binary 形式に設定しておく方が扱いやすくなる。

binary に設定する方法は、system/controlDict ファイル内の「writeFormat」を「binary」に変更することで設定できる。binary ファイルについての詳細は、9-1-2 項を参照。

### 9-6-1. サーバ接続の為の準備

サーバに接続する場合は、まず「~/.ssh/config」ファイル内に、接続するサーバの情報を記述する必要がある。このファイルは、ssh を使ってサーバに接続する場合に必要な。以下にその例を示しているが、これは、FOCUS に ssh で接続する場合と ssl-vpn 接続する場合の例になる。

尚、当然であるが、サーバに接続する場合は、接続先のアカウントとパスワード、秘密キーと公開キーは、作成しておく必要がある。また、この config、秘密キーのファイルアクセス権限は、自身のみの設定にしておかないと接続できない。

```
----- .ssh/config -----
#-- FOCUS 用 (ssh 接続の例) ----
Host FocusLogin
  HostName      ssh.j-focus.jp
  User          ****0001                #ユーザ名
  Port          22
  IdentityFile  ~/.ssh/ff01Focus_id_rsa  #秘密キー

Host ff01Focus
  HostName      ff01.j-focus.jp
  User          ****0001                #ユーザ名
  ProxyCommand  ssh FocusLogin nc %h %p

#-- FOCUS 用 (vpn 接続の例) ----
Host ff01FocusVpn
  HostName      ff01.j-focus.jp
  User          ****0001                #ユーザ名
  Port          22
  IdentityFile  ~/.ssh/ff01Focus_id_rsa  #秘密キー
-----
```

次に、TreeFoam 側に ssh で接続するサーバの情報や、サーバをマウントするローカル側のディレクトリの設定などの情報を設定する必要がある。この設定は、「~/.TreeFoamUser/data/sshfs\_data」ファイルに記述する。以下にその例を示している。この例は、FOCUS に接続する例で、ssh 接続と ssl-vpn 接続の例になる。ハッチング部は、必須項目になる。

```
----- .TreeFoamUser/data/sshfs_data -----
#
#  sshfs によるサーバマウント
#  -----
#---- FOCUS (ssh 接続の例) ----
Host FOCUS
  HostName      ff01Focus                #~/.ssh/config で定義している host 名
  HostDir       /home1/g***/*0001        #マウントする host 側のディレクトリ
  MountPoint    #マウントする local 側のディレクトリ
  setEnviron    #login 後の環境設定
    . ~/OFv2306terminal
    cd ~

#---- FOCUS (vpn 接続の例) ----
Host FOCUS_vpn                #ssl-vpn 接続用
  HostName      ff01FocusVpn          #~/.ssh/config で定義している host 名
  HostDir       /home1/g***/*0001
  MountPoint
```



```
setEnviron
. ~/OFv2306terminal
cd ~
```

上記の内、任意の項目 (MountPoint、setEnviron) に関しては、必要に応じて設定する。(以下を参照)

MountPoint	TreeFoam が書き換えるので、入力不要。 (マウントする local 側のディレクトリを指定する。)
setEnviron	サーバ側に、ここに記述してある内容で setEnviron ファイルを作成する。 サーバ側の「~/bash_profile」(シェル起動時にシェルが読み込むファイル)の最後に「. setEnviron」の一行を追記しておく事で、login シェルが起動した時に、記述した内容の環境設定で bash シェルを起動させる事ができる。 上記の設定は、OpenFOAM の環境設定と、カレントディレクトリの設定を行っている。 この為、login シェル起動時に、指定したディレクトリに移動して、OpenFOAM の環境設定が済んだ状態で bash シェルが起動する事になる。 「~/OFv2306terminal」は、サーバ側の \$HOME フォルダに、予め OFv2306terminal のスクリプトを作成しておく必要があるが、初回起動は、「cd ~」の行のみに設定。 尚、「cd ~」の行は、移動先のディレクトリを設定する為の行であるが、この行は、TreeFoam が随時書き換えるので、このディレクトリの内容は、何でも可。

ここまでで、TreeFoam 上からサーバに接続し、TreeFoam 上にサーバ側のディレクトリツリーが表示できる状態になっている。しかし、サーバとのデータやり取りは、時間がかかることが多い。これを解決する為には、サーバ側で処理ができるものは、サーバで処理させその結果を受け取る方法をとる様にしている。その方法として、従来は、サーバ側に実行ファイルを置き、その実行ファイルを local 側から起動する方法をとっていたが、この方法は、TreeFoam 側のバージョンアップにより、サーバ側の実行ファイルもバージョンアップが必要になることがあり、管理が難しくなる。

この為、今回 (ver 3.0 以降) から、実行ファイルを local の TreeFoam 内に置き、実行時は、その実行ファイルをサーバに転送して、サーバ側で実行して結果を受け取る方法をとる様にした。これにより、バージョン管理は、local の TreeFoam のみ管理すれば済む。

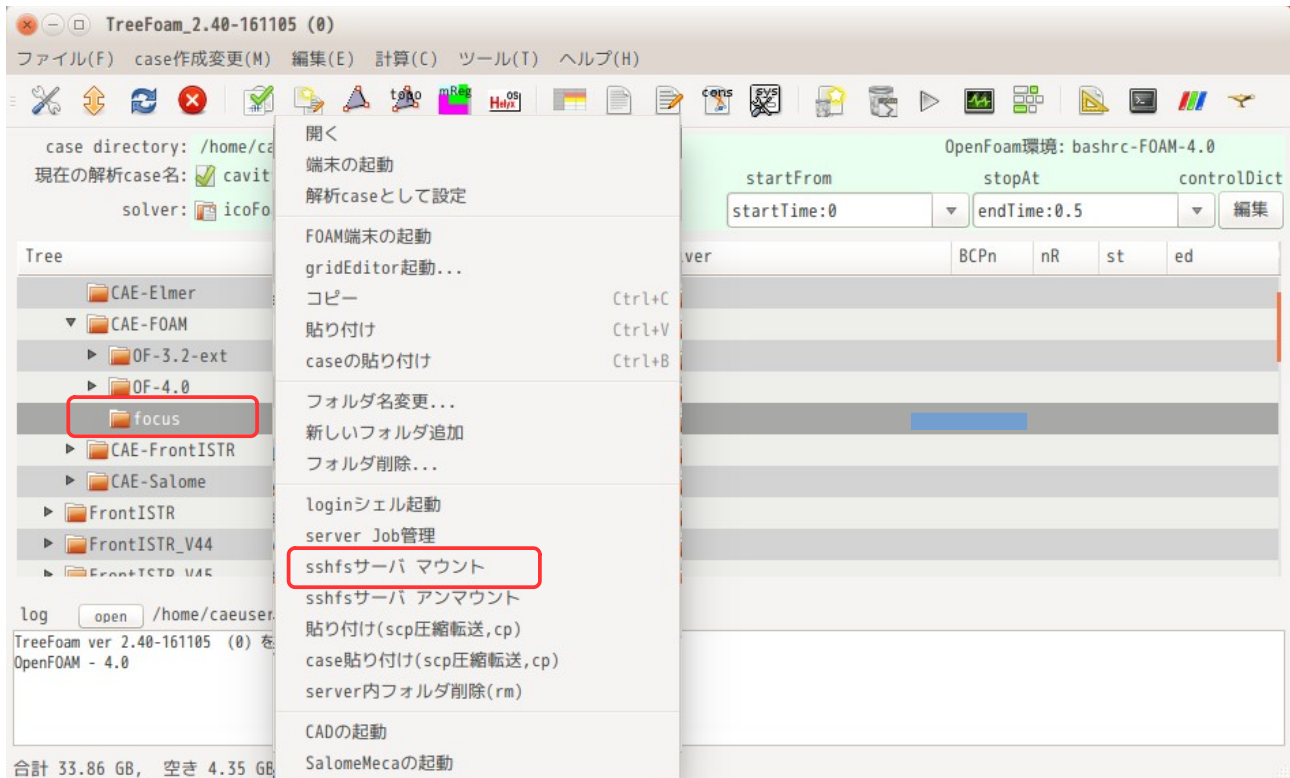
サーバ側に転送する実行ファイルは、bash シェルスクリプトと python スクリプトがあるので、これらが実行できる環境が必要。python スクリプトは、python2, python3 の両方で動かすことができる。実行は、「python \*.py」で実行しているので、サーバ側の python 環境に従って、python2 or python3 で実行される事になる。

## 9-6-2. サーバ接続とサーバのマウント

前項の設定ができた段階で、TreeFoam 上でサーバに接続し、そのサーバをマウントする事ができる。接続したサーバをローカル PC にマウントする為に、マウント用の folder を作成しておく。今回の例では、「focus」フォルダを作成している。(この folder は、空 folder にしておく。) この後、その folder を選択し、右クリックでポップアップメニューを表示させ、「ssh サーバ マウント」を選択する。(下図参照。)

この後、「server マウント」画面上から、接続したい server を選択する。接続できる server は、9-6-1 項で定義した server になる。server 選択後「OK」ボタンをクリックする事で、server がマウントされる。尚、この時、テキストボックスの内容 (hostName、hostDir、setEnviron) を修正すると修正された内容で、server をマウントする事ができる。

これにより、サーバ側の \$HOME ディレクトリの内容が「focus」フォルダにマウントされる。



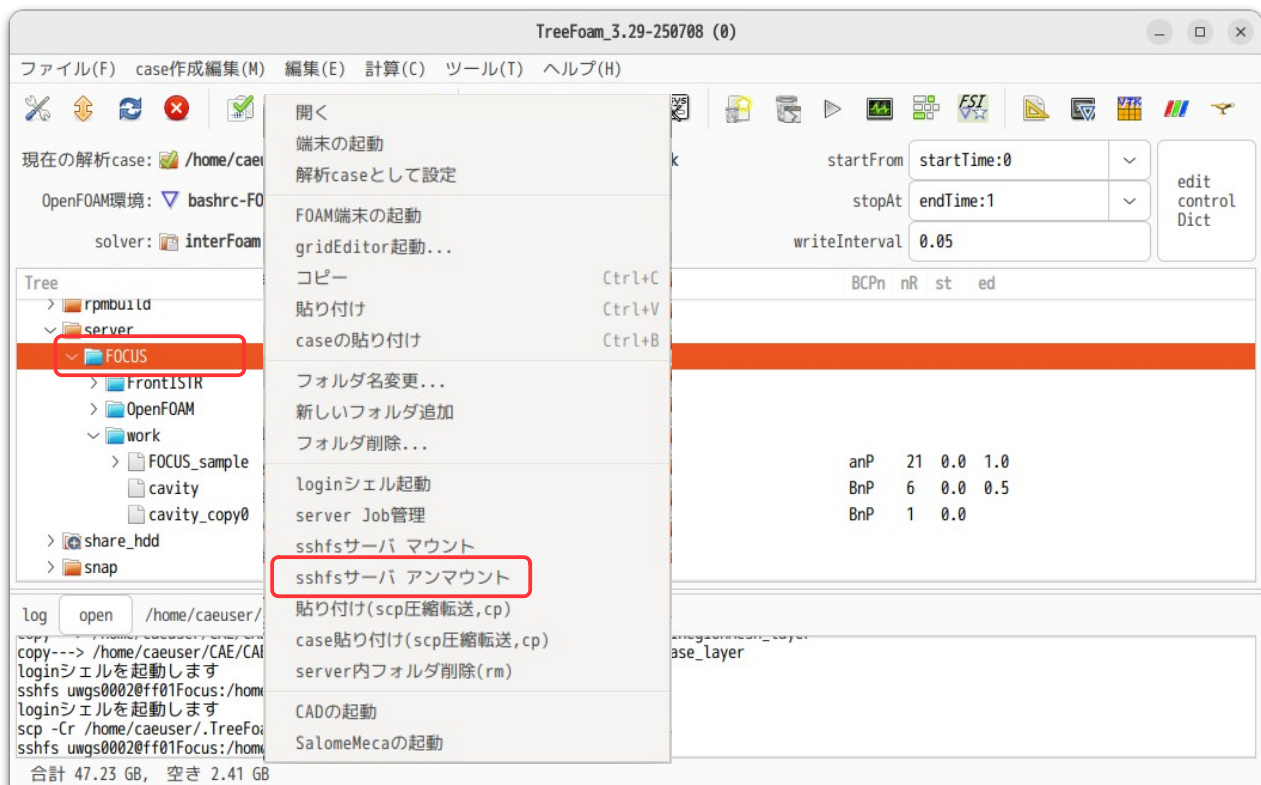
サーバマウント後は、空 folder だった「focus」 folder 内に、下図の様にサーバ側の\$HOME フォルダの内容がそのまま表示される状態になる。(下図参照)



サーバをマウントする事によって、サーバ側の folder や file へのアクセスが、ローカル側と同様な感覚で操作する事が可能になる。  
 ただし、サーバ側のデータを通信で取得している為、サイズの大きなファイル（例：メッシュファイル、計算結果が入っている field など）をオープンする事は難しい。サイズの小さな controlDict や transportProperties などのファイルは、何ら問題なくオープンし、編集・保存する事ができる。

### 9-6-3. サーバ切断とサーバのアンマウント

マウントしたサーバを切断しアンマウントするには、サーバをマウントしたフォルダを選択して、右クリックでポップアップメニューを表示させて、「sshfs サーバ アンマウント」を選択する。  
 これによって、サーバを切断し、アンマウントする事ができる。



サーバをアンマウントしても、「~/ssh/config」は、そのまま残っているので、端末を起動してここから ssh や scp コマンドを使用する事はできる。

### 9-6-4. サーバとローカル間の folder コピー方法

サーバとローカル間のデータコピーは、scp コマンドを使ってコピーしているが、データの転送時間を早める為に、scp コマンドに圧縮オプションを追加したコピーを行う事ができる。

このコピー方法は、ポップアップメニュー上から

ローカル側のコピー元を選択して「コピー」  
 サーバ側のコピー先 folder を選択して「貼り付け(scp 圧縮転送,cp)」  
 又は「case 貼り付け(scp 圧縮転送,cp)」

する事で、scp の圧縮転送で copy&paste する事ができる。  
 上記は、「ローカル → サーバ」にコピーする事を想定しているが、反対に「サーバ → ローカル」にコピーする場合も同様に行うことで、圧縮転送することができる。

また、サーバの folder をサーバ folder にコピーする場合（サーバ内同士のコピー）も同様にする事で copy&paste する事ができる。サーバ内同士の場合は、直接、cp コマンドを ssh で送出しているの、コピー時間は早い。

データを貼り付けている最中は、貼り付けが終わるまで、戻って来ないので、TreeFoam は操作できない状態になってしまうが、この場合は、新しい TreeFoam を起動すれば、他の操作ができる。

### 9-6-5. FOCUS の Job 管理

TreeFoam では、FOCUS 用の Job 管理ツールを準備しているので、Job ファイルの作成・編集や Job の投入が容易に行える。また、FOCUS 内の各システムの使用状況も確認できるので、空いているシステムを選び、queue や並列数、使用 Node 数を設定して Job を投入できる。

FOCUS への Job 投入に関しては、事例があるのでこれを参考にできる。  
 以下は、「OpenFOAM-v2312 の damBreak」を Job 投入するサンプルの例になる。  
 場所は、「/home1/share/x86\_64/el8/OpenFOAM.com/OpenFOAM-v2312/FOCUS\_sample-v2312/go.sh」にある。

----- go.sh の例 (25/07/09 に確認) -----

```
#!/bin/bash
#SBATCH -p s006m_032c    # キュー名
#SBATCH -N 1             # ノード数
#SBATCH -J sampleOF      # ジョブ名
#SBATCH -o %x%.o         # 標準出力ファイル、%J は Slurm が割り当てるジョブ ID に置換
#SBATCH -e %x%.e         # 標準エラー出力ファイル、%J は同上
```

```
#dir: tutorials/multiphase/interFoam/laminar/damBreak/*
```

```
source /etc/profile.d/modules.sh
```

```
### OS7 系の場合 GNU10 必要
str=$( cat /etc/redhat-release | grep -i rocky )
if [[ "${str}" == '' ]]; then
  #echo "This is not Rocky" $str
  module load PrgEnv-gnu-10.3.0
fi
```

```
### IntelOneAPI2024 環境設定
source /home1/share/x86_64/el8/intel/oneapi-2024.1.0/setvars.sh
```

```
### Ssystem の場合は TCP の指定が必要
if [[ "${SLURM_JOB_PARTITION:0:1}" == 's' ]]; then
  export I_MPI_FABRICS=shm:ofi
  export I_MPI_OFI_PROVIDER=tcn
  export I_MPI_COLL_EXTERNAL=no
fi
```

```
### OpenFOAM.com 環境設定
PDIR=/home1/share/x86_64/el8/OpenFOAM.com/
. ${PDIR}/OpenFOAM-v2312/etc/bashrc

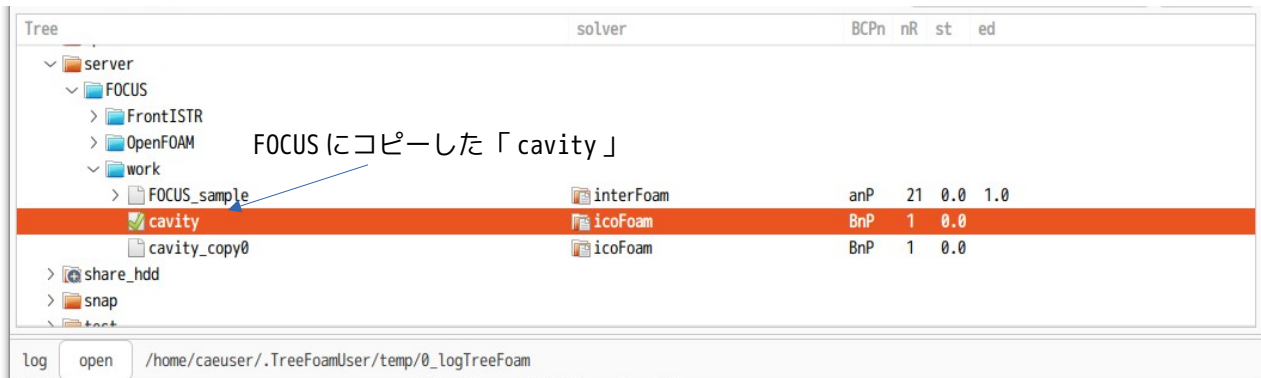
. ${WM_PROJECT_DIR}/bin/tools/CleanFunctions
. ${WM_PROJECT_DIR}/bin/tools/RunFunctions
```

```
### 実行
./Allclean
./Allrun
```

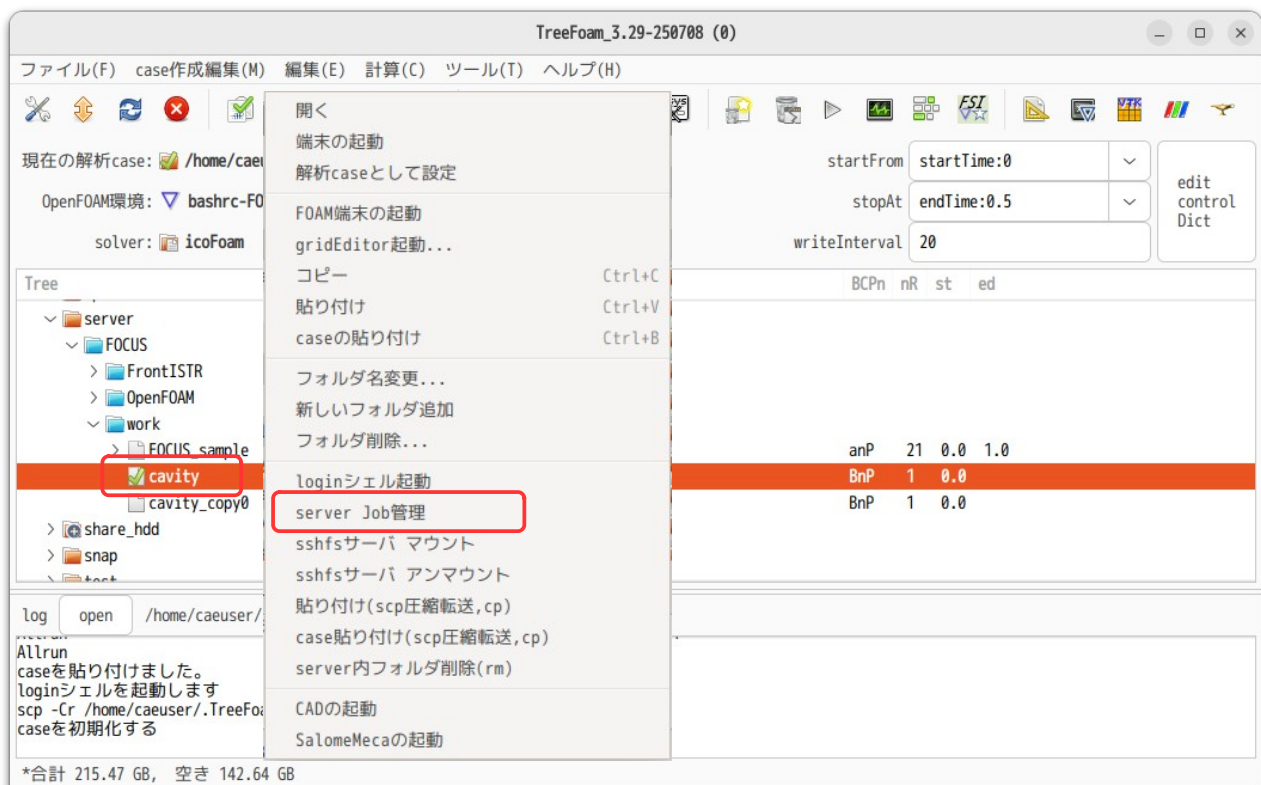
この例を参考に、今回 Job を投入してみる。

#### 9-6-5-1. FOCUS Job 管理の起動

Job を投入したい case を選択し、ポップアップメニューから「server Job 管理」を選択する。以下の例は、OpenFOAM-v2306 の tutorials 内の cavity を実行してみる。  
まず、local 側で実行できる事を確認した「cavity」を準備する。(計算結果は、削除しておく)  
この「cavity」を FOCUS 内の適当な場所にコピーする。(下図参照)



この後、cavity を選択した状態で、右クリックで popup メニューを表示させ、「server Job 管理」を選択する。



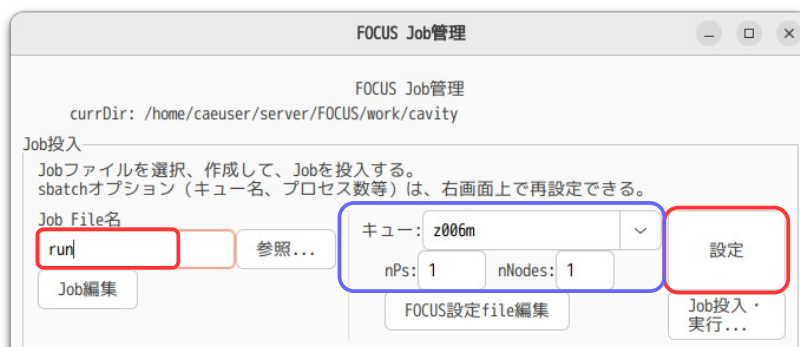
「server Job 管理」を選択すると以下の Job 管理画面が現れる。





#### 9-6-5-2. Job ファイルの選択・編集・実行

FOCUS Job 管理画面の「参照...」ボタンをクリックして、Job ファイル名（バッチファイル）を入力する。この後、青枠内を入力（シングルコアで計算する設定）し、「設定」ボタンをクリックする。これにより、Job ファイル「run」に設定内容（キューや並列数等）が書き込まれる。



「job 編集」ボタンをクリックして、Job ファイルを以下のように完成させる。この内容は、「FOCUS\_sample-v2312/go.sh」を参考に作成している。（赤字部分は、「設定」ボタンで TreeFoam が作成した部分。赤字以外が editor で追記した部分。）

```
----- 最終的な Job ファイル -----
#!/bin/bash
#SBATCH -p z006m
#SBATCH -n 1
#SBATCH -N 1
#SBATCH -J run
```

```

#SBATCH -e run.e%J
#SBATCH -o solve.log

source /etc/profile.d/modules.sh

### OS7 系の場合 GNU10 必要
str=$( cat /etc/redhat-release | grep -i rocky )
if [[ "${str}" == ' ' ]]; then
    #echo "This is not Rocky" $str
    module load PrgEnv-gnu-10.3.0
fi

### IntelOneAPI2024 環境設定
source /home1/share/x86_64/el8/intel/oneapi-2024.1.0/setvars.sh

### Ssystem の場合は TCP の指定が必要
if [[ "${SLURM_JOB_PARTITION:0:1}" == 's' ]]; then
    export I_MPI_FABRICS=shm:ofi
    export I_MPI_OFI_PROVIDER=tcp
    export I_MPI_COLL_EXTERNAL=no
fi

### OpenFOAM.com 環境設定
PDIR=/home1/share/x86_64/el8/OpenFOAM.com/
. ${PDIR}/OpenFOAM-v2312/etc/bashrc

. ${WM_PROJECT_DIR}/bin/tools/CleanFunctions
. ${WM_PROJECT_DIR}/bin/tools/RunFunctions

icoFoam
-----

```

Job ファイル作成後は、上図の「Job 投入・実行…」ボタンをクリックすると、以下の画面が現れるので、「OK」ボタンをクリックすることで、Job が実行される。



尚、キュー名については、TreeFoam 上で設定しているキュー名しか選択できないが、「設定 file 編集」ボタンで、下図の様な設定ファイルが開くので、新しいキュー名を追加したり、log ファイル名も修正することで、追加したキュー名が選択できたり、log ファイル名が変更できる様になる。

```

----- 設定ファイル (~/.TreeFoamUser/focus_data)の内容 -----
#
# FOCUS setting data
#
# Log file names
# These names are able to change at following lines.
logFileName      solve.log
errLogFileName   ${jobName}.e%J
#
# List of queue names
# These names are able to add new queue Names.
queueNames
a006m
a024h
a168h

```

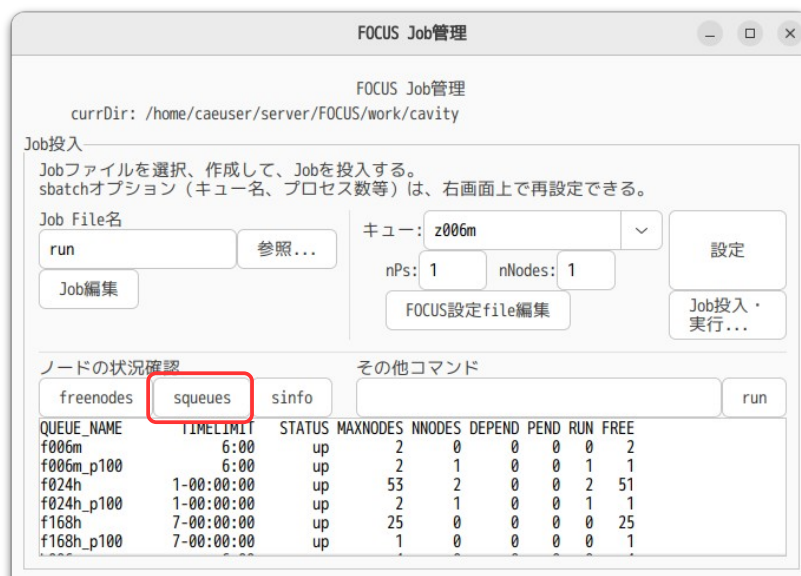
```

b006m
b024h
b168h
f006m
f006m_p100
f024h
f024h_p100
f168h
f168h_p100
h006m
h024h
h168h
p168h
q024h
q024h_a100
r024h
r168h
s006m_032c
s006m_092c
s024h_032c
s024h_092c
s168h_032c
s168h_092c
v024h
v168h
w024h
x024h
z006m
z024h
z168h

```

### 9-6-5-3. 大規模な Job の投入

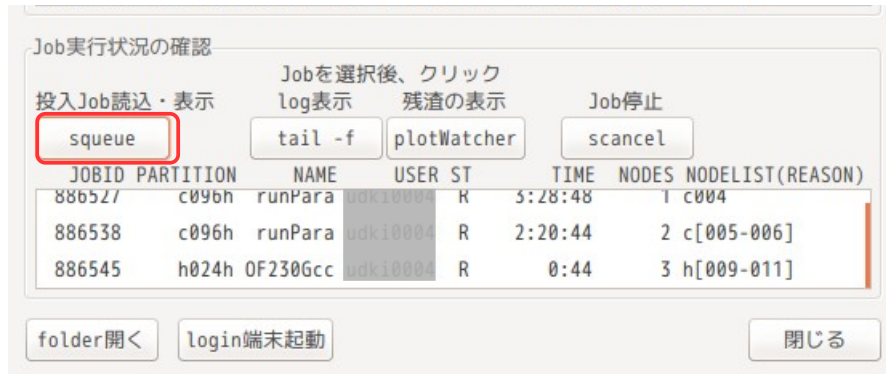
実際の Job を投入する場合は、並列処理を行うことが多い。  
 並列処理するためには、FOCUS 側の各システムの空き状況を調べた上で、Job 投入することになる。  
 この各システムの使用状況を確認する場合には、「freenodes」「squeues」「sinfo」コマンドで確認するが、このコマンドのボタンを準備しているのので、各ボタンをクリックすることで、その状況を知ることができる。  
 下図は、「squeues」ボタンをクリックした状態になる。テキストボックス中にその結果が表示されている。  
 使用状況を確認後、キュー名や使用 node 数 (nNodes) を変更する場合は、前記したようにテキストボックス中の内容を修正して「設定」ボタンをクリックしておく。



「その他コマンド」は、freenodes, squeues, sinfo 以外のコマンド投入する時 (例: uacct コマンド等) に、そのコマンドを入力し、<enter>又は「run」ボタンをクリックする事で、そのコマンドの実行結果が帰ってくる。入力するコマンドは、「ls」コマンドでも実行結果が帰ってくる。

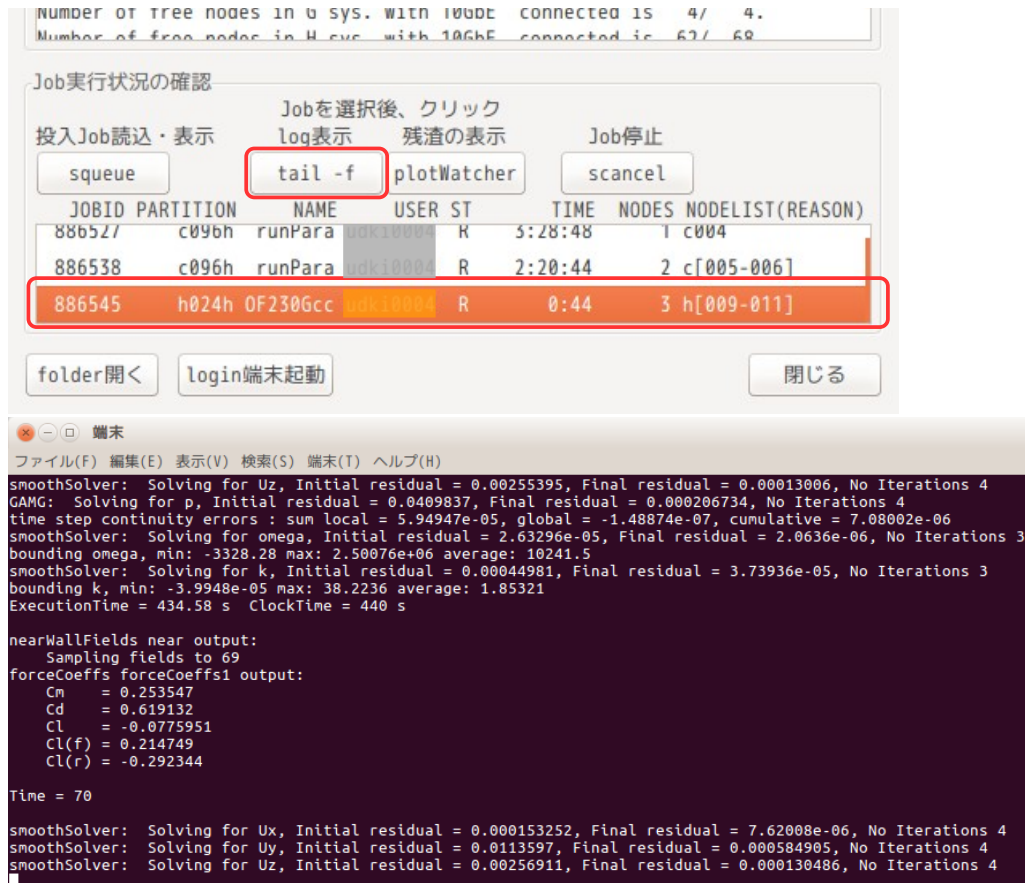
#### 9-6-5.4 実行 Job の管理

Job を投入した後は、その Job の実行状況が確認できる。その確認方法は、「squeue」ボタンをクリックする事で確認できる。(squeue コマンド実行した結果が表示される。)



投入した各々の Job の計算状況を確認する方法として、  
 計算 log の表示  
 残渣の表示  
 で確認することができる。

計算 log の表示は、確認したい Job を選択し、「tail -f」ボタンをクリックする事で、以下の端末が開き、log を確認することができる。

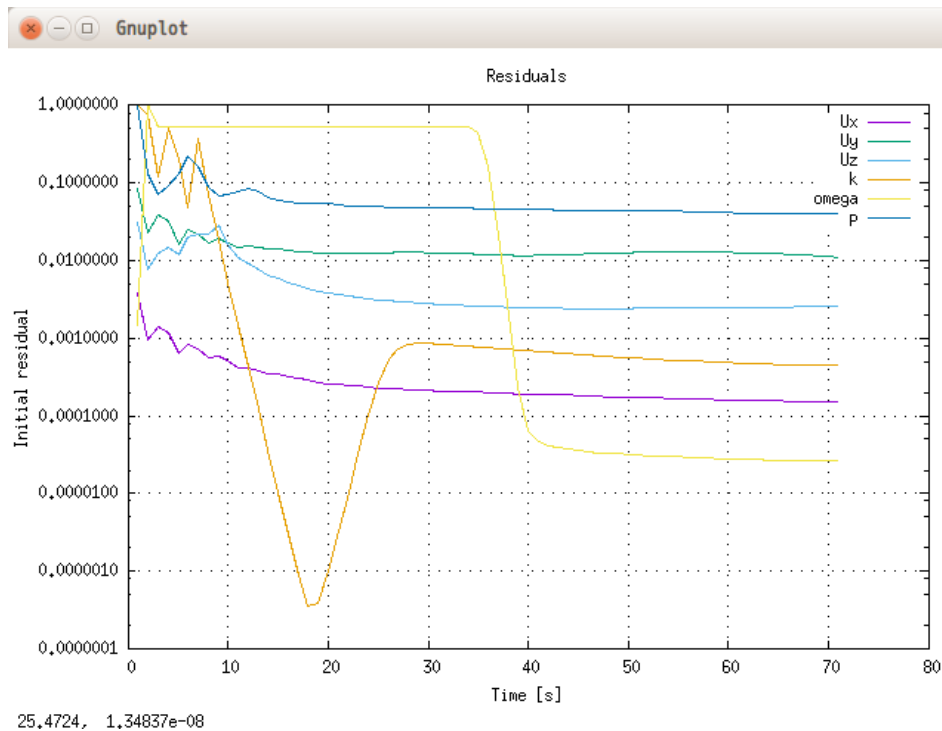


log の表示は、JobId から logFile を検索する必要がある。この為に、サーバ側の\$HOME 直下に「.jobList」

ファイルを作成し、この中に jobId と Job 内容 (caseDirectory 等) を保存するようにしている。job 内容の保存は、サーバに Job を投入した時に jobId と Job 内容を追加保存する。これにより、jobId からその Job の logFile が容易に検索できる。

Job 投入後は、通常、Job 管理のために「squeue」コマンドで job の実行状況を確認するので、このコマンドを実行する時に、全 job の実行状況を確認し、「.jonList」ファイルを更新する様にしている。(squeue コマンド実行すると、「.jobList」ファイルは、最新の状態に更新される。)

また、残渣の表示も同様に Job を選択し、「plotWatcher」ボタンをクリックする事で、plotWatcher が起動し、以下の様に残渣を確認する事ができる。残渣の表示も logFile を検索する必要があるが、上記と同様な方法で、logFile を検索している。



投入した Job を停止 (scancel) したい場合は、その Job を選択して、「scancel」ボタンをクリックすることで、停止できる。停止できたかどうかは、再び「squeue」ボタンをクリックして実行状況を表示させることで、確認できる。

