Basilisk On-water Run by openFoam v2006

23, April.2021 修正

重合格子（oversetmesh）を用いて混相流（multiphase flow）の計算を行う．図に示すように，水槽内（上面が空気，その他は壁）を仮定し，マルチボディ（5 剛体）のバシリスクが，水面で脚を動かし，足裏の水の反力によって走行するシミュレーションを行う．自由表面（気液境界面移動）を扱う VOF（volume of fluid）法を使っている．変形，移動するセルに密度比を与えて計算している．例えば，水のセルは 1（=1000 [kg/m^3]），空気は 0（=1.3 [kg/m^3]），自由表面上にあるセルの半分が水面下にあるときには，0.5 になる．また，物体が大きく移動するため，脚の周りにメッシュを生成し，計算空間のメッシュと重なり合わせ，物性値が一致するように収束計算を行っている．外側の計算空間のメッシュは移動しない．バシリスク周りのメッシュは基準座標系に対しても，物体座標系に対しても移動する（魚座標系から見ても動く）．それぞれの剛体毎重合格子の計算空間が生成される．openfoam のライブラリを修正してトルクによる角度 PD 制御関数をプログラミングする．

以下，計算条件

5 剛体モデル： 5 自由度(x, y 並進，xyzy 回転)で運動するボディ（robobody）と，このボディに対して 1 自由度（y 回転）の揺動運動をする上部左右脚（legR, legL），この脚に対して 1 自由度（y 回転）で揺動運動をする左右の下部脚（footR, footL）で，合計 5 剛体とする（ボディは運動が安定したら（z 方向に沈まないだけの揚力が生み出せるようになったら）6 自由度にする．脚も実際は 7 自由度あるはずなので，今後は増やしていく．これは各脚 2 自由度設定）．

運動：位相の 180deg 異なる左右脚と，揺動運動の中心が異なる上下脚による水面走行により移動する．4 つの脚剛体は，トルク入力による PD 角度制御（10Hz）される．剛体どおしは作用反作用の法則にしたがう．剛体表面には，気液，両流体から反力が帰る．

物体座標系は，それぞれの回転軸を原点にしている．各長さ＝0.05m

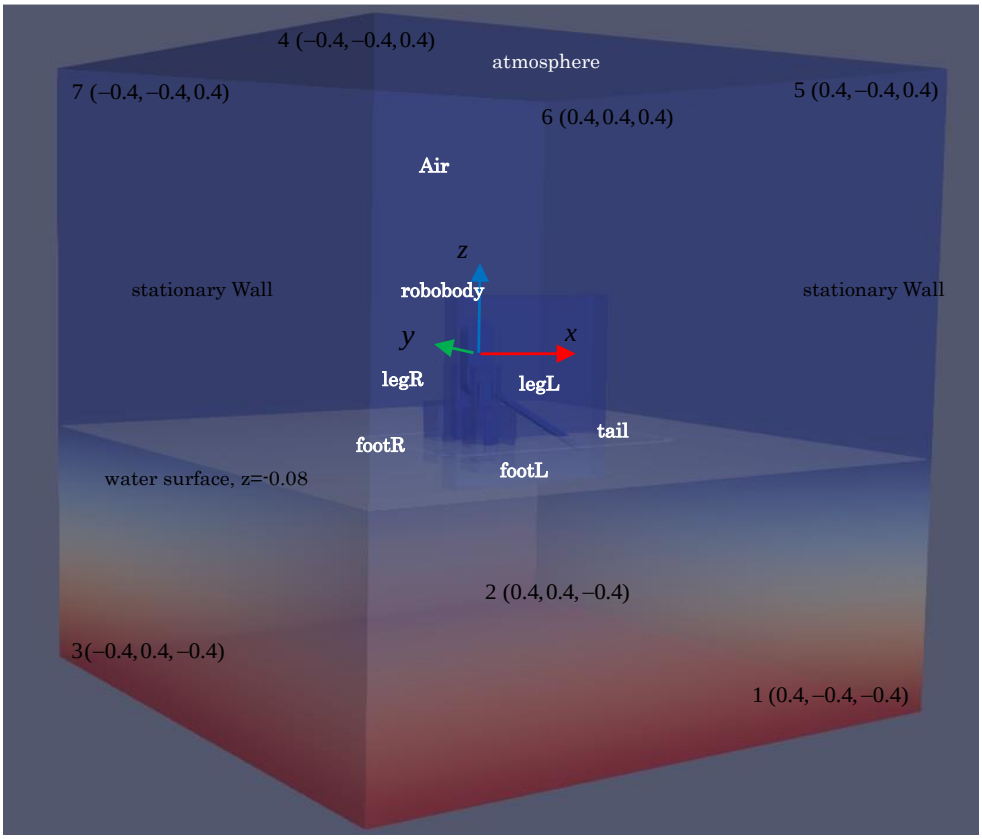
レイノルズ数は，0.1m の脚が 10Hz で 15deg の振幅と想定すると，脚振り速度と水上移動速度が最終的に等しくなると仮定して

水中 ： $\mathrm{Re}=\dfrac{UL}{\nu}=\dfrac{(15*4/180*\pi*10*0.1)*(0.1)}{1.0*10^{-6}}=1.05*10^{5}$

ストローハル数は，

$$\mathrm{Re}=\frac{fL}{U}=\frac{10*0.1}{15*4/180*\pi*10*0.1}=0.955$$

である．マッハ数も低いので，非定常，非圧縮，乱流ということになるが，ここでは，定常速度に達する前の水上移動速度が 0m/s に近い状態のシミュレーションを行うため，境界条件は層流と仮定して以下表のとおりとする．なお，天井が大気，それ以外は壁．ボディの中心を原点として，水面は z=-0.08m である．



boundary name	Type	alpha.water	U 速度	p_rgh 圧力と水圧	pointDisplacement 運動で与える変位	Zone ID
atmosphere	patch	type inletOutlet; inletValue uniform 0; value uniform 0;	type pressureInletOutletVelocity; value uniform (0 0 0);	type totalPressure; p0 uniform 0;	type fixedValue; value uniform (0 0 0);	type zeroGradient
stationaryWalls	wall	zeroGradient	type fixedValue; value uniform (0 0 0);	type fixedFluxPressure;	type fixedValue; value uniform (0 0 0);	type zeroGradient
****Sides	overset			type zeroGradient;	patchType overset; type zeroGradient;	
robobody, legR, legL, footR, footL	wall	zeroGradient	type movingWallVelocity; value uniform (0 0 0);	type fixedFluxPressure;	type calculated; value uniform (0 0 0);	type zeroGradient
oversetPatch	overset			type overset;	patchType overset; type zeroGradient;	
overset				patchType overset; type fixedFluxPressure;		

注：slip は，スカラー値の場合には zeroGradient，ベクトル量の場合には，法線方向が zero,接線方向が zeroGradient とのこと．
oversetPatch はダミーとのことだが，意味は分からず．overset もしかり．．．空欄部分は，
#includeEtc "caseDicts/setConstraintTypes"
で設定しているようだ．

1. Tutorial とライブラリのコピー

最も内容が近い tutorial の重合格子を利用した混相流浮遊物計算である以下を windows の適当な作業ディレクトリ（ex. “basilisk03”）にコピーしてくる．

¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2006¥tutorials¥multiphase¥overInterDyMFoam¥floatingBody

以下のような構成になっているが，

background/
floatingBody/
Allclean
Allrun

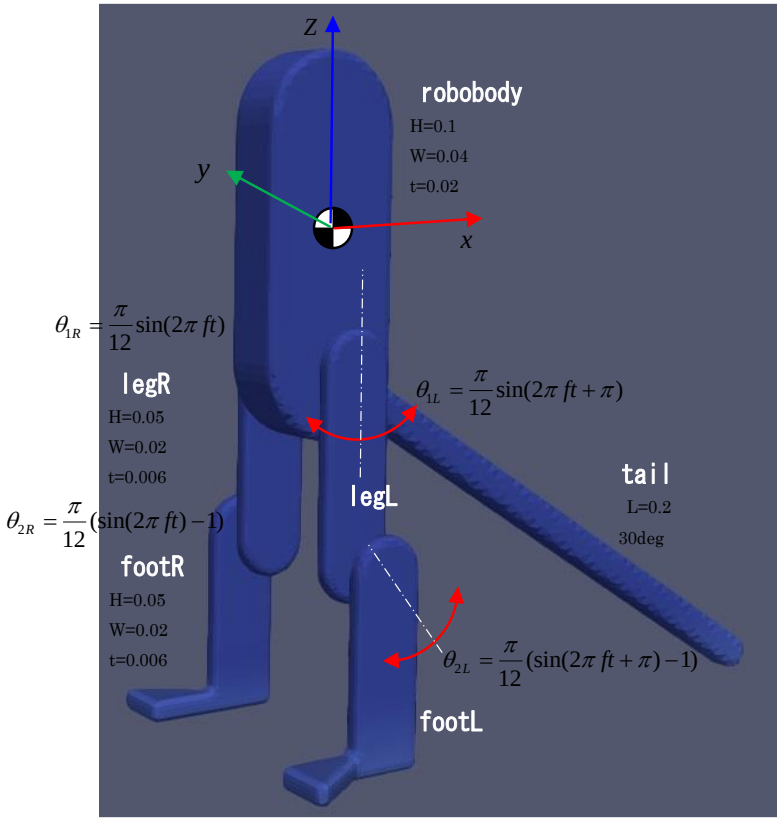
いつもの構成に合わせるために，

0.org/
constant/
lib/
oversetmeshbody/
oversetmeshfootL/
oversetmeshfootR/
oversetmeshlegL/
oversetmeshlegR/
system/
Allrun
Librun
basilisk.foam
motion.xlsx

とする．flaotingBody を oversetmesh***にディレクトリ名を変更し，background の中にある 0.org, constant, system という三つのディレクトリを外に出し，background ディレクトリは消してある．5 つある oversetmesh***は，それぞれ重合格子の計算空間を作成するディレクトリである．lib, Librun はライブラリ関係であり後述する．basilisk.foam は，paraview を呼び出すための空ファイルである．エクセルファイルは，脚の目標角度の時系列の設定ファイルである．

2. CAD モデルと運動重合格子の計算空間の生成

CAD で 5 つの剛体モデル（ex. “body.stl”, “legR.stl”, “legL.stl”, “footR.stl” , “footL.stl”）を作成する．ボディはピッチ姿勢安定のためのしっぽ付き．CAD は，1/1000 スケールで作成し，stl ファイルにする（mm 設定，ASCII 設定に注意）．後述する snappyHexMesh が鋭角を得意としていないようなので，とがっている部分にはフィレットをつけてある．脚の運動は，図中の式のとおり．エクセルの時系列入力のため，最終的には，実際のバシリスクのモデルを入力するのがよい．水面に足が 2cm 沈んでいる設定にしてある．ここでは，上半身の重心を z 固定しているため実際は沈まない．姿勢もほとんど崩れない．安定して計算ができるようになったら，これらの設定を実際のバシリスクモデルに戻す．



3. 各重合格子の計算空間の生成

上記剛体モデルそれぞれの重合格子の計算空間を生成する．まず，robobody から作成する．

oversetmeshbody/constant/triSurface/

というディレクトリを作成し，CAD で作ったボディのモデル（ex. “body.stl”）をここにコピーする．以下，oversetmeshbody/system 内のファイルを変更する．

．system/blockMeshDict

重合格子用の計算空間の設定ファイルを変更する．なお，blockMeshDict とは，bolockMesh 関数が呼び出す辞書（dictionary）という意味である．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      blockMeshDict;
}

scale   1;              // スケールは変更なし．等倍

vertices    // 計算空間の頂点座標 8 つ．他の計算空間と重ならないようにしている．
(
    (-0.04 -0.0148 -0.14)
    ( 0.20 -0.0148 -0.14)
    ( 0.20  0.0148 -0.14)
    (-0.04  0.0148 -0.14)
    (-0.04 -0.0148  0.11)
    ( 0.20 -0.0148  0.11)
    ( 0.20  0.0148  0.11)
    (-0.04  0.0148  0.11)
);
```

```
blocks
(               // 6 面体(hexahedron)の頂点番号．上記頂点の順番になる．ゾーン名 robobodyZone は，この後使わないが，つけておく．
x, y, z 方向にそれぞれ 48, 10, 50 分割．そのあとは各方向の分割比率 1, 1, 1．この設定で 5mm 四方のメッシュになる．
    hex (0 1 2 3 4 5 6 7) robobodyZone (48 10 50) simpleGrading (1 1 1)
);

edges
(
);

boundary   // 境界の設定
(
    robobodySides    // 重合格子の境界面の名前
    {
        type overset; // 境界層タイプを overset に設定
        faces         // 境界面を頂点 4 つでそれぞれ設定
        (
            (0 3 2 1)
            (2 6 5 1)
            (1 5 4 0)
            (3 7 6 2)
            (0 4 7 3)
            (4 5 6 7)
        );
    }
    robobody          // ボディ境界面
    {
        type wall;    // 境界タイプは壁
        faces ();     // 後で設定されるので今は指定保留．
    }
);
```

．snappyHexMeshDict

snappyHexMesh を使うため，topoSetDict は消す．以下の SnappyHexMeshDict ファイルをまるまる作成する．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      snappyHexMeshDict;
}

// Which of the steps to run
castellatedMesh true;  // 重合格子空間と CAD モデルのブーリアン演算の実行宣言
snap            true;          // ブーリアン演算後の境界面の補正の実行宣言
addLayers       false;         /* いつものように，境界層は断念  */

geometry
{
    robobody          // 保留したボディ境界面の名前
    {
        type    triSurfaceMesh;
        file    "body.stl";     // CAD ファイル名
    }
};
```

```
castellatedMeshControls
{
    maxLocalCells 100000;
    maxGlobalCells 2000000;
    minRefinementCells 10;
    nCellsBetweenLevels 2;
    features ();

    refinementSurfaces              /* 境界面メッシュ分割回数（細分化） */
    {
        robobody
        {
            level (1 1);            // テスト計算なので粗目．1 回のみ分割．よって，5mm のメッシュは 2.5mm になる．
        }
    }

    // Resolve sharp angles on fridges
    resolveFeatureAngle 30;

    refinementRegions
    {
    }

    locationInMesh (0 0 0.105);                 /* ブーリアン引き算後の残す側のメッシュ指定．ボディのない重合格子内を選択 */
    allowFreeStandingZoneFaces true;
}

snapControls
{
     nSmoothPatch 3;
     tolerance 2;
     nSolveIter 30;
     nRelaxIter 5;
}

addLayersControls     /* 境界層をつけないことにしたので，これ以下は使わない．一応修正してある */
{
     relativeSizes true;

    layers
    {
        robobody
        {
            nSurfaceLayers 3;
        }
    }

    expansionRatio 1.0;
    finalLayerThickness 0.5;
    minThickness 0.25;
    nGrow 0;
    featureAngle 360;
    nRelaxIter 5;
    nSmoothSurfaceNormals 16;
    nSmoothNormals 3;
    nSmoothThickness 10;
    maxFaceThicknessRatio 0.5;
    maxThicknessToMedialRatio 0.3;
    minMedianAxisAngle 90;
    nBufferCellsNoExtrude 0;
    nLayerIter 50;
}
```

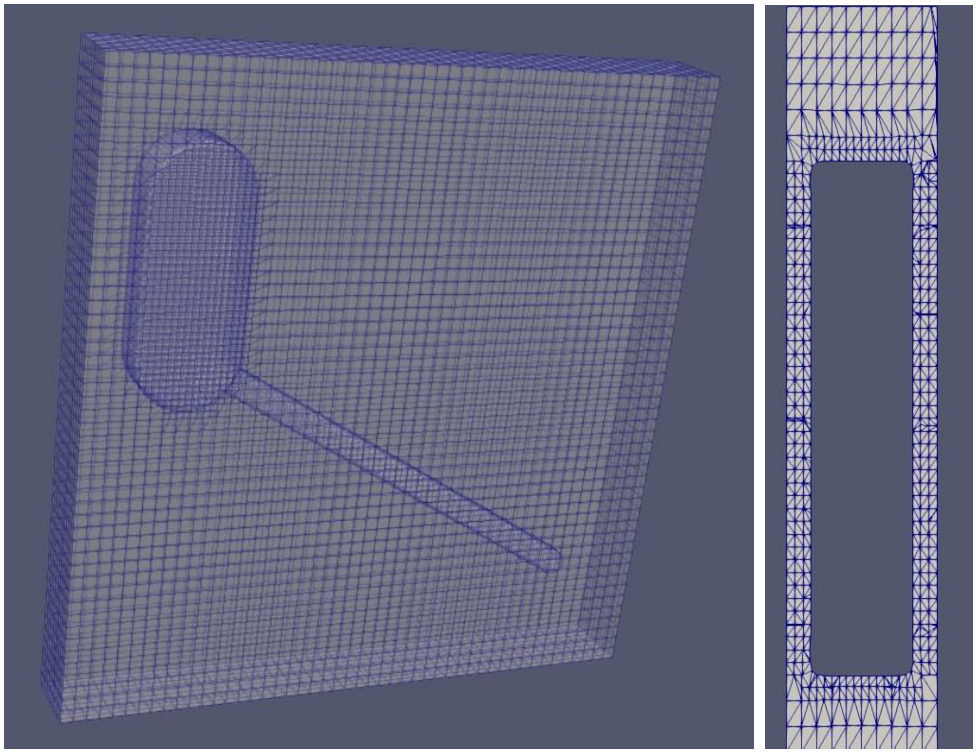
```
meshQualityControls
{
    maxNonOrtho 65;
    maxBoundarySkewness 20;
    maxInternalSkewness 4;
    maxConcave 80;
    minVol 1e-13;
    minTetQuality 1e-30;
    minArea -1;
    minTwist 0.05;
    minDeterminant 0.001;
    minFaceWeight 0.05;
    minVolRatio 0.01;
    minTriangleTwist -1;
    nSmoothScale 4;
    errorReduction 0.75;
}

mergeTolerance 1e-6;
```

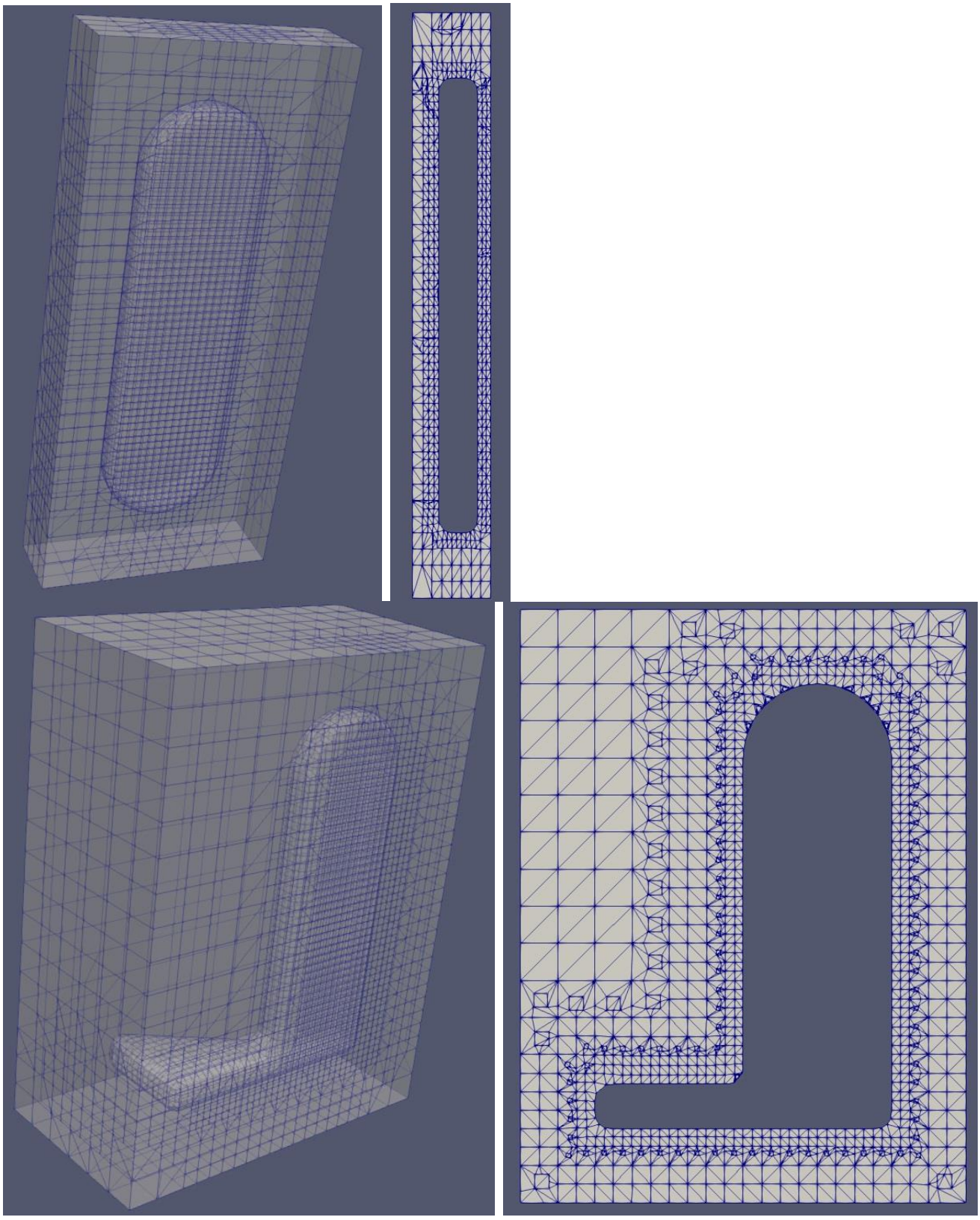
system 内のこれ以外のファイルはそのまま．なお，この時点で，linux において

```
> blockMesh
> snappyHexMesh –overwrite
```

を実行して paraview で確認すると，以下のとおり．相当粗い．．．理想的な設定からは程遠い．paraview の使い方に関しては後述．



同様に，oversetmeshlegR，oversetmeshlegL, oversetmeshfootR, versetmeshfootL 内のディレクトリも変更する．以下のような重合格子空間になっている．



4. 外側計算空間の設定

一つ上のディレクトリに移動し，system 内のファイルを変更して外側計算空間を生成する．

．blockMeshDict

計算空間の作成ファイル

```
FoamFile
{
    version         2.0;
```

```
    format          ascii;
    class           dictionary;
    object          blockMeshDict;
}

scale   1;

vertices    // 計算空間の 8 頂点座標の指定．この順番が節点番号
(
    ( -0.4    -0.4      -0.4)
    (  0.4    -0.4      -0.4)
    (  0.4     0.4      -0.4)
    ( -0.4     0.4      -0.4)
    ( -0.4    -0.4       0.4)
    (  0.4    -0.4       0.4)
    (  0.4     0.4       0.4)
    ( -0.4     0.4       0.4)
);

blocks
(
    hex (0 1 2 3 4 5 6 7) (50 50 50)            // 頂点となる節点番号．x, y, z 方向分割数．これで 1.6cm 四方のメッシュ
    simpleGrading (((0.3 0.25 0.5) (0.4 0.5 1) (0.3 0.25 2))
                   ((0.3 0.25 0.5) (0.4 0.5 1) (0.3 0.25 2))
                   ((0.3 0.25 0.5) (0.4 0.5 1) (0.3 0.25 2)))
        // 分割比率を変更し，中心ほどメッシュを細かくする．（空間比率，メッシュ数比率，メッシュスケール比率） (0.3 0.25 0.5)は，x 方向開始 30%に，全体の 25%のメッシュを割り当て，x に進むほどメッシュを細かくし，最初のメッシュ長さに対して最終的なメッシュ長さを 0.5 倍にする．
);

edges
(
);

boundary
(
    // Dummy patch to trigger overset interpolation before any other bcs　このダミーのパッチの意味は分からず．が，書かないとエラーがでる．
    oversetPatch
    {
        type overset;           /* 重合格子の境界面の指定 */
        faces ();
    }
    stationaryWalls             /* 天井以外は壁境界 */
    {
        type wall;              // 境界タイプは壁
        faces
        (
            (0 3 2 1)           //境界面の節点番号．内側から見て時計回り．表と裏の概念がある．
            (2 6 5 1)
            (1 5 4 0)
            (3 7 6 2)
            (0 4 7 3)
        );
    }
    atmosphere                  /* 天井は大気 */
    {
        type patch;
        faces
        (
```

```
            (4 5 6 7)
        );
    }
);

mergePatchPairs
(
);
```

．topoSetDict

境界面などの設定．重合格子と外側の計算空間のゾーンの ID 指定のための設定を行っている．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}

actions
(
    {
        name    calspacecell;  /* 計算空間の名前*/
        type    cellSet;       /* 操作タイプはセルのセット*/
        action  new;           /* 新しく作る  */
        source  regionToCell;  /* ソールは領域から設定  */
        insidePoints ((0.4 0 0)); /*   この点が含まれる領域を選択*/
    }

   {
        name    robobodycell;          /* 仮のボディの格子空間の名前   */
        type    cellSet;
        action  new;
        source  cellToCell;
        set     calspacecell;          /* 上記外側計算空間をそのままコピー  */
    }

    {
        name    robobodycell;
        type    cellSet;
        action  invert;                /* 外側計算空間の補集合．つまり，外側計算空間以外の領域  */
    }

    {
        name    legRcell;              /* 右脚空間名  */
        type    cellSet;
        action  new;
        source  regionsToCell;
        set     robobodycell;          /* 上記ボディ格子空間から指定した点を含むゾーンを選択  */
        insidePoints
        (
            (0 0.02 0.015)
        );
    }

    {
        name    legLcell;              /* 左脚空間設定  */
        type    cellSet;
```

```
    action    new;
    source    regionsToCell;
    set       robobodycell;
    insidePoints
    (
        (0 -0.02 0.015)
    );
}

{
    name      footRcell;              /* 右下部脚設定 */
    type      cellSet;
    action    new;
    source    regionsToCell;
    set       robobodycell;
    insidePoints
    (
        (0 0.035 -0.035)
    );
}

{
    name      footLcell;              /* 右下部脚設定 */
    type      cellSet;
    action    new;
    source    regionsToCell;
    set       robobodycell;
    insidePoints
    (
        (0 -0.035 -0.035)
    );
}

{
    name      robobodycell;           /* 仮ボディ空間 -= 右脚 */
    type      cellSet;
    action    subtract;
    source    cellToCell;
    set       legRcell;
}
{
    name      robobodycell;           /*仮ボディ空間 -= 左脚 */
    type      cellSet;
    action    subtract;
    source    cellToCell;
    set       legLcell;
}
{
    name      robobodycell;           /* 仮ボディ空間 -= 右脚下部 */
    type      cellSet;
    action    subtract;
    source    cellToCell;
    set       footRcell;
}
{
    name      robobodycell;           /* 仮ボディ空間 -= 左脚下部．最終的に，ボディ空間のみが残る．  */
    type      cellSet;
    action    subtract;
    source    cellToCell;
    set       footLcell;
}
```

);

．setFieldsDict

気相領域と，液相領域の設定．VOF（volume of fluid）法では，異なる流体の密度を α という係数で表現している．それぞれのメッシュの流体の密度は，

$$\rho = \alpha\rho_{water} + (1-\alpha)\rho_{air}$$

となっている．α=1 が水，α=0 が空気，0<α<1 が境界面を含むメッシュということになる．これより，メッシュを動かさなくても，混相流を移動させることができる．境界面をメッシュの辺にして移動させる手法もある．VOF では，この α に伴い，クーラン数は，0<α<1 の範囲用もあり，常に監視している．この値は，計算精度を保証するため 0.5 以下にしておく（らしい）．

```
defaultFieldValues
(
    volScalarFieldValue alpha.water 0                   /* 0: 空気設定    */
    volScalarFieldValue zoneID 123                      /*  以下で使う 0 と 1 以外の数字で初期化  */
);

regions
(
    boxToCell
    {
        box ( -0.4 -0.4 -0.4 ) ( 0.4 0.4 -0.08 );                   /* 1: 水設定．水面は z=-8 [cm]    */
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }

    cellToCell
    {
        set calspacecell;           // 外側計算空間は ID=0 に設定
        fieldValues
        (
            volScalarFieldValue zoneID 0
        );
    }

    cellToCell
    {
        set robobodycell;           // ボディ計算空間は ID=1 に設定
        fieldValues
        (
            volScalarFieldValue zoneID 1
        );
    }

    cellToCell
    {
        set legRcell;               // 右脚上部計算空間は ID=2 に設定
        fieldValues
        (
            volScalarFieldValue zoneID 2
        );
    }

    cellToCell
    {
        set legLcell;               // 左脚上部計算空間は ID=2 に設定
        fieldValues
        (
```

```
            volScalarFieldValue zoneID 3
        );
    }

    cellToCell
    {
        set footRcell;          // 右脚下部計算空間は ID=3 に設定
        fieldValues
        (
            volScalarFieldValue zoneID 4
        );
    }

    cellToCell
    {
        set footLcell;          // 左脚下部計算空間は ID=4 に設定
        fieldValues
        (
            volScalarFieldValue zoneID 5
        );
    }
);
```

．controlDict

数値計算の設定，ソルバの選択，パラメータの指定，ファイル保存設定など．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      controlDict;
}

libs            (overset fvMotionSolvers);          /* この重合格子用のライブラリが v2006 にしかない．  */

application     overInterDyMFoam;
startFrom       latestTime;             /* 計算開始時間．前回から */
startTime       0.0;
stopAt          endTime;
endTime         1;                      /* 終了時間．10Hz であるので，10 回の走行*/
deltaT          0.0001;                 /* 初期計算時間刻み．一周期 1000 イテレーション  */
writeControl    adjustable;             /* 計算刻み可変*/
writeInterval   0.005;                  /* データ保存間隔．1 周期 20 ファイル */
purgeWrite      0;
writeFormat     ascii;
writePrecision  6;
writeCompression off;
timeFormat      general;
timePrecision   6;
runTimeModifiable yes;
adjustTimeStep  yes;            // 時間刻み可変設定．クーランによる数制御

maxCo           1.0;                    /* 全体用の最大クーラン数．速度×時間刻み／メッシュスケール */
maxAlphaCo      0.5;                    /* 0<α<1 用（水面）の最大クーラン数．  */
maxDeltaT       0.001;                  /* 時間刻みの最大値．クーラン数によって変更されるので，上限を小さめにしておく．
流体計算が安定していても，剛体計算がそうでない可能性があるので，剛体の時間刻みを流体の時間刻みにあまり左右されないようにする．最低でも一周期 100 回にしてある  */
```

．fvSchemes

計算スキームの選択．ここも層流用に変更しておく．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSchemes;
}

ddtSchemes
{
    default         Euler;
}

gradSchemes
{
    default         Gauss linear;
}

divSchemes          // 層流を選択したので，k-ε計算ルーチンは消しておく
{
    div(rhoPhi,U)   Gauss upwind;
//  div(U)          Gauss linear;       // このルーチンを消してよいかどうか，後日確認
    div(phi,alpha)  Gauss vanLeer;
    div(phirb,alpha) Gauss linear;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
/*
    div(phi,k)      Gauss upwind;
    div(phi,epsilon) Gauss upwind;
    div(phi,omega) Gauss upwind;        */
}

laplacianSchemes
{
    default         Gauss linear corrected;
}

interpolationSchemes
{
    default         linear;
}

snGradSchemes
{
    default         corrected;
}

oversetInterpolation
{
    method          cellVolumeWeight;
}

oversetInterpolationSuppressed
{
    grad(p_rgh);
    surfaceIntegrate(phiHbyA);
}

fluxRequired
```

```
{
    default         no;
    p_rgh;
    pcorr;
    alpha.water;
}
```

．fvSolution

逆行列計算スキーム選定，および，収束判定値設定．ここも層流用に変更しておく．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      fvSolution;
}

solvers
{

    "cellDisplacement.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-06;          // 誤差許容値
        relTol          0;              // 初期値からの相対的許容値
        maxIter         100;
    }

    "alpha.water.*"
    {
        nAlphaCorr      3;
        nAlphaSubCycles 2;
        cAlpha          1;
        icAlpha         0;

        MULESCorr       yes;
        nLimiterIter    5;
        alphaApplyPrevCorr  no;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
    }

    "pcorr.*"
    {
        solver          PCG;
        preconditioner  DIC;
        tolerance       1e-9;
        relTol          0;
    }

    p_rgh
    {
        solver          PBiCGStab;
        preconditioner  DILU;
        tolerance       1e-9;
        relTol          0.01;
    }
```

```
    p_rghFinal
    {
        $p_rgh;
        relTol          0;
    }

//      "(U|k|omega|epsilon).*"
    "U.*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }
}

PIMPLE
{
    momentumPredictor   no;
    nOuterCorrectors    2;
    nCorrectors         2;
    nNonOrthogonalCorrectors 0;

    ddtCorr                     yes;
    correctPhi                  no;

    moveMeshOuterCorrectors no;
    turbOnFinalIterOnly no;

    oversetAdjustPhi    no;
}

relaxationFactors
{
    fields
    {
    }
    equations
    {
        ".*" 1;
    }
}

cache
{}
```

. topoSetDict2

重合空間のメッシュスケールと外部計算空間のメッシュスケールは収束計算の関係上同程度がよい．ここでは，前者が 5mm 以下，後者が 1cm 程度になっている．よって，refineMesh 関数でさらに外部空間の指定部分を半分に細分化する．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      topoSetDict;
}

actions
(
    {                                           /* 細分化する領域の名前    */
```

```
        name    refineCells;
        type    cellSet;
        action  new;
        source  boxToCell;      /* 立方体空間指定*/
        sourceInfo
        {
            box (-0.1 -0.1 -0.18) (0.24 0.1 0.15);      // 立方体の対角頂点座標指定
        }
    }
);
```

．refineMeshDict

topoSet で指定された空間を細分化する辞書.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "system";
    object      refineMeshDict;
}

set                 refineCells;    // この名前が付けられた領域を細分化する

coordinateSystem global;

globalCoeffs
{
    tan1                (1 0 0);
    tan2                (0 1 0);
}

directions      ( tan1 tan2 tan3 );         // 3 次元分割指定

useHexTopology  yes;                        // 6 面体で分割.
geometricCut    no;
writeMesh       no;
```

5. 境界条件（0.org ディレクトリファイル）の修正

境界条件の設定．この時点で決定できない境界があるので，メッシングの後に 0 ディレクトリを作るため，設定ファイルを作成する．

．alpha.water

境界条件における α（水，空気）の初期設定ファイル．壁面では勾配を 0 にしているようだ.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       volScalarField;
    object      alpha.water;
}

dimensions      [0 0 0 0 0 0 0];  /* 無次元    */

internalField   uniform 0;          // 定数定義

boundaryField
```

```
{
    #includeEtc "caseDicts/setConstraintTypes"          // include している境界条件もあるようだ

    "(robobody|legR|legL|footR|footL|stationaryWalls)"  // 境界複数指定
    {
        type                zeroGradient;    // 勾配が 0
    }

    atmosphere          // 天井
    {
        type                inletOutlet;     // 入出力条件
        inletValue          uniform 0;       // 入力の場合 0
        value               uniform 0;       // いや，結局空気以外考えられないので 0
    }
}
```

．pointDisplacement

メッシュの境界移動の境界条件ファイル．RigidBodyMotion が要求してくるファイル．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       pointVectorField;
    object      pointDisplacement;
}

dimensions      [0 1 0 0 0 0 0];

internalField   uniform (0 0 0);

boundaryField
{
//      #includeEtc "caseDicts/setConstraintTypes"          あるとエラーを出すので消した．全く意味わからず...

    stationaryWalls
    {
        type                fixedValue;      // 側面の壁は動かない
        value               uniform (0 0 0);
    }

    atmosphere
    {
        type                fixedValue;      // 天井も動かない
        value               uniform (0 0 0);
    }

    "(robobody|legR|legL|footR|footL)"      // 物体は運動による計算値に基づき動く
    {
        type                calculated;
        value               uniform (0 0 0);
    }

    oversetPatch                            // 重合空間境界の呪文
    {
        patchType       overset;
        type            zeroGradient;
    }
```

```
    "(robobodySides|legRSides|legLSides|footRSides|footLSides)"   // 重合空間境界設定．かならず勾配 0 とのこと．
    {
        patchType           overset;
        type                zeroGradient;
    } }
```

. p_rgh
圧力境界は，混相流の場合には，静水圧を考慮して P-ρ gh となっている．

```
FoamFile
{
    version         2.0;
    format          ascii;
    class           volScalarField;
    object          p_rgh;
}

dimensions          [1 -1 -2 0 0 0 0];

internalField     uniform 0;

boundaryField
{
    oversetPatch
    {
        type                overset;
    }

    stationaryWalls
    {
        type                fixedFluxPressure;          // 固定流束
    }

    atmosphere
    {
        type                totalPressure;
        p0                  uniform 0;
/*          U                   U;
        phi                 phi;
        rho                 rho;
        psi                 none;
        gamma               1;
        value               uniform 0;
```

Supersonic compressible 用の設定であるが，マッハ数の極めて小さい流れ場にこの設定の意味が分からない．さらに，γ=1 の時点で速度など他の設定は無意味になる． 式は以下の通り：

$$P = \frac{P_0}{(1+\frac{\gamma-1}{2\gamma}\varphi\|U\|^2)^{\frac{\gamma}{\gamma-1}}} \quad where \ \gamma = \frac{C_p}{C_v} > 1$$

```
*/
    }

    "(robobody|legR|legL|footR|footL)"
    {
        type                fixedFluxPressure;          // 流速固定
    }

    overset
    {
        patchType           overset;
        type                fixedFluxPressure;
```

```
    }
}
```

以下のファイルの中身は，”floatingobject”を””(robobody|legR|legL|footR|footL)””にだけ変更しておく．速度条件 U 以外は乱流計算用ソルバの係数設定ファイルである．

. U
.zoneID

6. 物性・運動設定（constant ディレクトリファイル）の修正

脚先の逆運動学問題は下図の通り．角速度ωでαのピッチ角を持つ長軸 a，短軸 b の楕円上の位置 P は，以下の通り．なお，回転は y 軸から見て反時計回りが正なので，下図で見ると，y 軸回転なので時計回りが正．よって，角度に全てマイナスが付いている．

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} a\cos(-\omega t) \\ -b\sin(-\omega t) \end{bmatrix} - \begin{bmatrix} 0 \\ H \end{bmatrix} \qquad (1)$$

リンクマニピュレータの運動学から，脚先 P の位置は，

$$\begin{bmatrix} x \\ z \end{bmatrix} = \begin{bmatrix} L_1\sin(-\theta_1) + L_2\sin(-\theta_1-\theta_2) \\ -L_1\cos(-\theta_1) - L_2\cos(-\theta_1-\theta_2) \end{bmatrix} \qquad (2)$$

となる．逆問題は，移項して 2 乗して足すと

$$x^2 + 2xL_1\sin\theta_1 + z^2 + 2zL_1\cos\theta_1 + {L_1}^2 = {L_2}^2$$

となるので，

$$2xL_1\sin\theta_1 + 2zL_1\cos\theta_1 = -{L_1}^2 + {L_2}^2 - x^2 - z^2$$

より公式から，

$$\theta_1 = a\tan 2(2xL_1, 2zL_1) \pm a\tan 2(\sqrt{(2xL_1)^2 + (2zL_1)^2 - (-{L_1}^2 + {L_2}^2 - x^2 - z^2)^2}, -{L_1}^2 + {L_2}^2 - x^2 - z^2) \qquad (3)$$

$$= a\tan 2(x, z) \pm a\tan 2(\sqrt{(2xL_1)^2 + (2zL_1)^2 - (-{L_1}^2 + {L_2}^2 - x^2 - z^2)^2}, -{L_1}^2 + {L_2}^2 - x^2 - z^2)$$

となる．次に，(2)式を展開して，

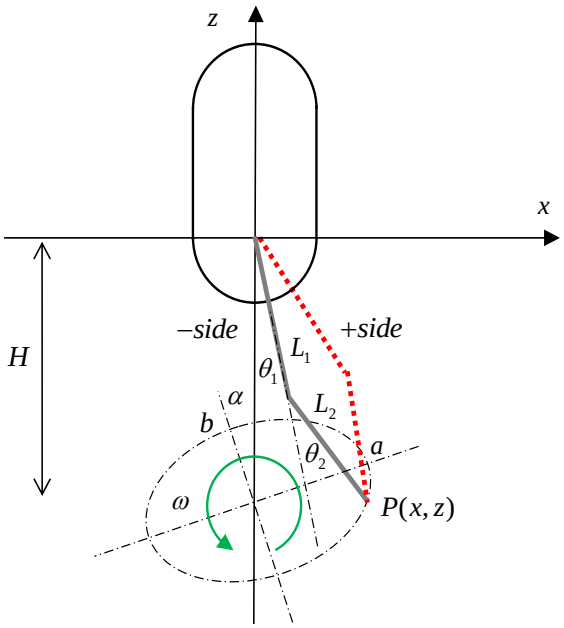
$$x = -L_1\sin\theta_1 - L_2\sin\theta_1\cos\theta_2 - L_2\cos\theta_1\sin\theta_2$$
$$z = -L_1\cos\theta_1 - L_2\cos\theta_1\cos\theta_2 + L_2\sin\theta_1\sin\theta_2$$

$$x\cos\theta_1 = -L_1\sin\theta_1\cos\theta_1 - L_2\sin\theta_1\cos\theta_1\cos\theta_2 - L_2\cos^2\theta_1\sin\theta_2$$
$$z\sin\theta_1 = -L_1\sin\theta_1\cos\theta_1 - L_2\sin\theta_1\cos\theta_1\cos\theta_2 + L_2\sin^2\theta_1\sin\theta_2$$
$$x\cos\theta_1 - z\sin\theta_1 = -L_2\sin\theta_2$$

$$x\sin\theta_1 = -L_1\sin^2\theta_1 - L_2\sin^2\theta_1\cos\theta_2 - L_2\sin\theta_1\cos\theta_1\sin\theta_2$$
$$z\cos\theta_1 = -L_1\cos^2\theta_1 - L_2\cos^2\theta_1\cos\theta_2 + L_2\sin\theta_1\cos\theta_1\sin\theta_2$$
$$x\sin\theta_1 + z\cos\theta_1 = -L_1 - L_2\cos\theta_2$$

$$\theta_2 = a\tan 2(-\frac{x\cos\theta_1 - z\sin\theta_1}{L_2}, -\frac{x\sin\theta_1 + z\cos\theta_1 + L_1}{L_2}) = a\tan 2(-x\cos\theta_1 + z\sin\theta_1, -x\sin\theta_1 - z\cos\theta_1 - L_1)$$

となる．(1)式に基づいて，脚先の運動を決めて，(3)式でθ_1，θ_2を求めて，制御という流れである．答えが二つあることに注意する．ここでは，-側を使う．図の灰色の実線がマイナス側の解．赤い破線がプラス側の解である．左右の脚は，ωの位相を 180deg ずらす．また，エクセルの atan2 の定義が x, y 逆になっているのに注意する．計算は，$\theta_1 = \theta_2 = 0$ から開始するので，初期の脚位置は大きく離れていることになる．入力トルクに上限を設けることにより極端に大きなトルクがかかることを回避しておく．



．dynamicMeshDict

剛体の物性と運動指定ファイル．メッシュ移動の設定も担っている．

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      dynamicMeshDict;
}

motionSolverLibs    (rigidBodyMeshMotionbasilisk03);        /* 剛体運動メッシュライブラリ．このあとこのライブラリをプログラミングする．このライブラリが rigidBodyDynamicsbasilisk03 を呼び出す．*/
dynamicFvMesh       dynamicOversetFvMesh;
motionSolver        rigidBodyMotion;

report              on;
g                   ( 0 0 -9.81 );          // 構造の重力定数．流体と同じにする．

solver
{
    type Newmark;   /* 剛体運動の積分方法は，ニューマークベータ法 */
    gamma   0.75;       // Velocity integration coefficient:        "gamma > 0.5" is unconditionary stable, but low accuracy.
    beta    0.390625;       // Position integration coefficient:        beta = (gamma+0.5)^2/4
        /* 一般的には，γ=0.5，β=0.25 の中立安定条件で計算するが，openfoam の剛体計算ソルバが不安定すぎるので，計算精度を犠牲にして安定性を優先した．効果は微小であったが有効．力制御の場合はデフォルトで問題ないかも． */
}

bodies
{
    robobodypart    /* ボディ設定 */
    {
        type            rigidBody; /* 剛体としての定義 */
        parent          root;                   /* 座標系を基準座標系に */
        mass            0.008;      /* 質量．だいたい比重 0.8 設定． */
        inertia         (0.00006 0 0    0.0001 0    0.0004);        /* 慣性テンソル（対称）は適当に入れた．計算を安定させるため大きめにした．ただしく計算すること */
        centreOfMass    (0 0 0.05);         /* 重心 */
```

```
        // Transformation tensor and centre of rotation (CoR)
        transform       (1 0 0    0 1 0     0 0 1) (0 0 0.05);      /* 姿勢の基準と，回転中心の設定．重心を回転中心にする */

        joint       /* 自由度設定．openfoam の人たちは，関節と呼んでいるようだ */
        {
            type            composite;          /* 運動は並進と回転の合成*/
            joints
            (
                {
                    type Px;            /* 転ばぬように z は固定した．計算が安定したら，Pxyz にする */
                }
                {
                    type Py;
                }
                {
                    type Rxyz;                  /* xyz 軸回転可 */
                }
            );
        }

        patches         (robobody);     /* 運動対象はボディ境界  */
        innerDistance   100;
        outerDistance   200;
          // 回転中心から半径 innerDistance 内のメッシュは，剛体と共に並進（moving）する．innerDistance を超え，outerDistance
内のメッシュは歪を伴って動く（morphing）．この 100, 200 設定は，全てのボディ回りメッシュが並進することを意味する．
    }

     legRpart       // 右上部脚
    {

        type            rigidBody;
        parent          robobodypart;   // 参照座標系は，ボディ座標系

        centreOfMass    (0 0.02 -0.075);    // ボディ座標系における，ボディ重心からの相対位置ベクトル
        mass            0.003;
        inertia         (0.00003 0 0 0.00005 0 0.00002);        // ボディ座標系から見た慣性テンソル：  R^-1 I R．R は姿勢
        // transform and CoR - relative to parent CoR
        transform       (1 0 0 0 1 0 0 0 1) (0 0.02 -0.05);     // 参照座標系から見た姿勢行列．参照座標系からみた，ボディの
回転中心から右脚下部の回転中心の相対ベクトル
        patches         (legR); // 運動の対象の境界
        innerDistance   100;
        outerDistance   200;
        joint
        {
                type Ry;  // ボディに対する自由度は y 回転のみ．
        }
    }

    legLpart        // 左上部脚
    {

        type            rigidBody;
        parent          robobodypart;

        centreOfMass    (0 -0.02 -0.075);   // Relative to parent CoM
        mass            0.003;
        inertia         (0.00003 0 0 0.00005 0 0.00002);
        // transform and CoR - relative to parent CoR
```

```
        transform           (1 0 0 0 1 0 0 0 1) (0 -0.02 -0.05);
        patches             (legL);
        innerDistance       100;
        outerDistance       200;
        joint
        {
                        type Ry;
        }
    }

    footRpart           //  右下部脚
    {

        type                rigidBody;
        parent              legRpart;           //  参照座標系は，右上部脚

        centreOfMass        (0 0.015 -0.05);        // Relative to parent CoM
        mass                0.004;
        inertia             (0.00004 0 0 0.00006 0 0.000024);
        // transform and CoR - relative to parent CoR
        transform           (1 0 0 0 1 0 0 0 1) (0 0.015 -0.05);
        patches             (footR);
        innerDistance       100;
        outerDistance       200;
        joint
        {
                        type Ry;
        }
    }
    footLpart           //  左下部脚
    {

        type                rigidBody;
        parent              legLpart;           //  参照座標系は，左上部脚

        centreOfMass        (0 -0.015 -0.05);       // Relative to parent CoM
        mass                0.004;
        inertia             (0.00004 0 0 0.00006 0 0.000024);
        // transform and CoR - relative to parent CoR
        transform           (1 0 0 0 1 0 0 0 1) (0 -0.015 -0.05);
        patches             (footL);
        innerDistance       100;
        outerDistance       200;
        joint
        {
                        type Ry;
        }
    }}

restraints
{
    legROscilation    /* 右上部脚運動設定．  */
    {
        type            torqueControl;          // プログラミングする関数名
        body            legRpart;   /* 上記で定義した legRpart が対象.ID は，関節数によって記載順にナンバリングされる．Px
も,Pxyz も 1 とカウントされる．逆に，Px と Py を設定すると 2 になる．自由度と関節数は異なっている．*/
        reaction        robobodypart;           // 反トルクを与える剛体の名前．基本的には，参照座標系の物体に返す．
        gain            (2      0.02        0.4);   /*P ゲイン [Nm]，D ゲイン [Nms]，最大トルク   */
        target          table       /* 時間   （角度，角速度，ダミー），間は線形補間．エクセルで作成   */
        (
```

```
        (       0       (       0       0       0       )       )
        (       0.001   (       1.64369E-06     0.001643687     0       )       )
        ....
        );
    }

        legLOscilation          // 左上部脚設定
    {
        type        torqueControl;
        body        legLpart;
        reaction    robobodypart;
        gain        (2 0.02 0.4);       /* (P gain, D gain, max torque )    */
        target      table               /* target amplitude [rad], target velocity, dummy */
        (
        (       0       (       0       0       0       )       )
        (       0.001   (       -2.45336E-05    -0.024533629    0       )       )
        ......
        );
    }

    footROscilation   // 右下部脚設定
    {
        type        torqueControl;
        body        footRpart;
        reaction    legRpart;   // 反トルクを返す右脚上部
        gain        (2 0.02 0.2);       /* (P gain, D gain, max torque )    */
        target      table               /* target amplitude [rad], target velocity, dummy */
        (
        (       0       (       0       0       0       )       )
        (       0.001   (       -1.64367E-06    -0.00164367     0       )       )
        (       0.002   (       -1.31195E-05    -0.011475875    0       )       )
        .......
        );

    footLOscilation   // 左下脚設定
    {
        type        torqueControl;
        body        footLpart;
        reaction    legLpart;   // 反トルクを返す左脚上部
        gain        (2 0.02 0.2);       /* (P gain, D gain, max torque )    */
        target      table               /* target amplitude [rad], target velocity, dummy */
        (
        (       0       (       0       0       0       )       )
        (       0.001   (       -2.7821E-05     -0.027820985    0       )       )
        ....
        );
    }
}
```

．transportProperties

流体の物性値の設定ファイル.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    location    "constant";
    object      transportProperties;
}

phases (water air);
```

```
water
{
    transportModel   Newtonian;
    nu               1e-06;    /* 粘性係数 */
    rho              998.2;    /* 密度 */
}

air
{
    transportModel   Newtonian;
    nu               1.48e-05;
    rho              1;
}

sigma               0.072; /* surface tension [N/m]. tutorial は間違っているので修正する. */
```

. turbulenceProperties /* 層流モデル*/

遊泳速度が速くなり，Re が 3000 あたりを超えたら，RAS モデルに戻す方がよい.

```
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}

simulationType  laminar;        /* 魚の遊泳速度があがったら，乱流モデルに変更する */
```

.g

この重力設定ファイルはそのまま.

7. ライブラリのプログラミング

剛体運動のライブラリである

librigidBodyDynamics.so

を書き換えるわけであるが，これは，

librigidBodyMeshMotion.so

から呼び出されているので，結局，この二つのライブラリを再構築することになる．まず，以下を上記 lib ライブラリにコピーする.

¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2006¥src¥rigidBodyDynamics
¥¥wsl$¥Ubuntu-18.04¥opt¥OpenFOAM¥OpenFOAM-v2006¥src¥rigidBodyMeshMotion

権限などで Windows でコピーできない場合には，Linux の cp コマンドでコピーする．なお，”lnInclude”のディレクトリに Linux 側が制限をかけているようであるが，書き換えないのでコピーしなくてもよい（メイク中にたぶん呼び出す）．作業領域にコピーしたら

basilisk03¥lib¥rigidBodyDynamics¥restraints¥externalForce

をそっくりそのまま，

basilisk03¥lib¥rigidBodyDynamics¥restraints¥torqueContorol

としてコピーし，中身のファイル名を

externalForce.C -> torqueControl.C
externalForce.H -> torqueControl.H

と変更する．以下，中身をプログラミングする．openfoam 内では，呼び出される関数毎ディレクトリが作られている．

まず，座標変換の記述を確認する．openfoam は，横ベクトルを基本としているため，姿勢の行列などが転置されてる．これより，一般的なロボティクスにおける以下の座標変換

$$ {}^0\boldsymbol{x} = {}^0R_B \ {}^Bx \rightarrow \begin{bmatrix} {}^0x \\ {}^0y \\ {}^0z \end{bmatrix} = \begin{bmatrix} & & \\ {}^0\boldsymbol{x}_B & {}^0\boldsymbol{y}_B & {}^0\boldsymbol{z}_B \\ & & \end{bmatrix} \begin{bmatrix} {}^Bx \\ {}^By \\ {}^Bz \end{bmatrix} $$

は，以下のようになる．

$$ {}^0\boldsymbol{x} = {}^0R'_B \ {}^Bx \rightarrow \begin{bmatrix} {}^0x & {}^0y & {}^0z \end{bmatrix} = \begin{bmatrix} {}^Bx & {}^By & {}^Bz \end{bmatrix} \begin{bmatrix} & {}^0\boldsymbol{x}_B & \\ & {}^0\boldsymbol{y}_B & \\ & {}^0\boldsymbol{z}_B & \end{bmatrix} $$

0 は基準座標系，B は物体座標系．R と R’は，以下の関係がある．

$ {}^0R'_B = ({}^0R'_B)^T $

ということで，転置すれば，ロボティクスと同じになる．なお，Vector 型に縦ベクトル，横ベクトルの概念はない．よって，

Vector a;
Tensor<scalar> R;
で宣言したとき，

aR　　(1)
Ra　　(2)

両者とも計算できる．(1)は a を横ベクトルとして，(2)は a を縦ベクトルとして計算している．角速度はロボティクスにおける方法 II（ベクトルの足し算ができるが，積分ができない手法）で表現されている．なお，作用反作用の概念もあまりないようである．ほとんどの計算が基準座標系で記述されているので，物体座標系基準の考え方の関数が極めて少ない．よって，作るしかない．ここでは，y 軸回転のみ制御するプログラムにする．X,Z や，汎用性に関しては後日．

. torqueControl.H

ヘッダファイル．関数名を変更し，剛体名を示す reaction を登録し，位置ベクトル location をゲイン配列 gain と変更し，時間変数ベクトル force を時系列目標値 target と変更して利用している．

```
#ifndef RBD_restraints_torqueControl_H
#define RBD_restraints_torqueControl_H

#include "rigidBodyRestraint.H"
#include "Function1.H"

namespace Foam
{
namespace RBD
{
namespace restraints
{

class torqueControl
:
    public restraint
{
    // Private data

        //- Control Torque (Nm)
        autoPtr<Function1<vector>> torqueControl_;

        //- gain vector (stiffness, damper, dummy)
        vector gain_;

          //- body ID for the reaction force
          word        reaction_;  // word 型の変数定義．剛体の名前が word 型として定義されている．

public:
```

```
        //- Runtime type information
        TypeName("torqueControl");              // 辞書の type から呼び出される名前の登録

        // Constructors
            //- Construct from components
            torqueControl
            (
                const word& name,
                const dictionary& dict,
                const rigidBodyModel& model
            );

            //- Construct and return a clone
            virtual autoPtr<restraint> clone() const
            {
                return autoPtr<restraint>
                (
                    new torqueControl(*this)
                );
            }

        //- Destructor
        virtual ~torqueControl();

        // Member Functions

            virtual void restrain       // これが実際に呼び出される関数
            (
                scalarField& tau,
                Field<spatialVector>& fx,
                const rigidBodyModelState& state
            ) const;

            //- Update properties from given dictionary
            virtual bool read(const dictionary& dict);          // 辞書からパラメータを呼び出す関数

            //- Write
            virtual void write(Ostream&) const;                 // 書き込み関数
};

} // End namespace restraints
} // End namespace RBD
} // End namespace Foam

#endif
```

. torqueControl.C

与えられた目標角度，目標角速度に対して，トルク入力で PD 制御する関数をプログラミングする．左上部脚に着目し，幾何的関係を以下に示す．ここで，

Σ：基準座標系＝外部計算座標系

Σr：参照座標系＝ボディ座標系

Σb：物体座標系＝左上部脚座標系

R_r：基準座標系から見た参照座標系の姿勢（基準座標系は添え字を省略してある）

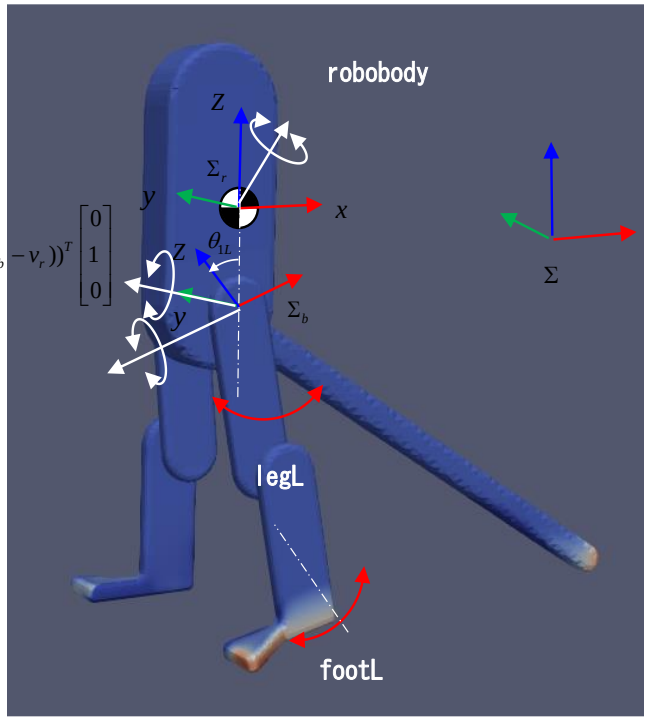
R_b：基準座標系から見た物体座標系の姿勢（基準座標系は添え字を省略してある）

${}^{r}R_b = R_r{}^{T} R_b$：参照座標系から見た物体座標系の姿勢（ボディから見た左上部脚の姿勢）

v_r: 基準座標系で表現した参照座標系の角速度ベクトル（ボディの速度）

v_b: 基準座標系で表現した基準座標系の角速度ベクトル（左上部の速度）

θ_{1L}：脚の曲げ角度．ここでは，参照座標系から見た物体座標系の z 軸より求める．$\theta = a\tan 2(z_x, z_z)$
である．
$\dot{\theta}_{1L} = (R_r^T(v_b - v_r)) \cdot \boldsymbol{e}_y$：参照座標系から見た，参照座標系に対する物体座標系の角速度ベクトルの y 成分



なお，openfoam の仕様として姿勢行列が転置されていることは述べたが，内部の要素の取り出し方は以下のとおりであり，注意が必要である．

$R = \begin{bmatrix} r_{xx} & r_{xy} & r_{xz} \\ r_{yx} & r_{yy} & r_{yz} \\ r_{zx} & r_{zy} & r_{zz} \end{bmatrix}$ であるとき，r_{xy} = R.xy(); である．

また，転置 R^T は，R.T()である．ロボティクスの概念に合わせるためにつねに転置して使うことになる．

& は，ベクトルどうしの掛け算で内積のみ，もしくは，行列どおしの掛け算，もしくは，行列とベクトルの掛け算を示す演算子．

ただし，openfoam に縦横ベクトルの概念がないので，縦ベクトルに右から横ベクトルをかけて，行列を得るような計算はできない．行列との掛け算の時，ベクトルの左右位置をひっくり返しても答えは同じ．注意が必要である．

なお，外積は ^

以上，一次元配列に縦横の概念はないが，ここでは，ベクトルは縦ベクトルとして考える．

.v()は，角速度ベクトルと，速度ベクトルを 6 要素ベクトルとして返してくる．以下のようにして 3 要素ベクトルとして取り出す．

.v().w()で 6 要素の前半分を取り出して，角速度ベクトルが得られる．

.v().l()で 6 要素の後半半分を取り出して，速度ベクトルが得られる．

model_.v(ID).w()で，ID が指し示すモデルの基準座標系における角速度ベクトルが得られる．

以上を理解した上で，以下のようにプログラミングする．

```
#include "torqueControl.H”          // 上記ヘッダの呼び出し
#include "rigidBodyModel.H"
#include "rigidBodyModelState.H"
#include "OneConstant.H"
#include "addToRunTimeSelectionTable.H"

#define      deg(a)       ((a)*180.0/3.141592)   // rad を deg に変換する関数

// * * * * * * * * * * * * * * * Static Data Members * * * * * * * * * * * * * * * //
namespace Foam
{
```

```
namespace RBD
{
namespace restraints
{
    defineTypeNameAndDebug(torqueControl, 0);

    addToRunTimeSelectionTable
    (
        restraint,
        torqueControl,
        dictionary
    );
}
}
}

// * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * //
Foam::RBD::restraints::torqueControl::torqueControl
(
    const word& name,
    const dictionary& dict,
    const rigidBodyModel& model
)
:
    restraint(name, dict, model),
    torqueControl_(nullptr),
    gain_(Zero)
{
    read(dict);
}

// * * * * * * * * * * * * * * * * Destructor  * * * * * * * * * * * * * * * * //
Foam::RBD::restraints::torqueControl::~torqueControl()
{}

// * * * * * * * * * * * * * * * Member Functions  * * * * * * * * * * * * * * //
void Foam::RBD::restraints::torqueControl::restrain
(
    scalarField& tau,
    Field<spatialVector>& fx,
    const rigidBodyModelState& state
) const
{
        label referenceID = model_.bodyID(reaction_);          // reaction_で示される剛体名に対応する bodyID
        const vector target = torqueControl_().value(state.t()); /* 線形補完された target[0]=目標角度，target[1]=目標角速度の読み込み*/
        vector moment = Zero;           // モーメント初期化
        Tensor<scalar> Rr = model_.X0(referenceID).E().T(), Rb = model_.X0(bodyID_).E().T();  // 参照座標系と物体座標系の姿勢行列
        Tensor<scalar> rRb = Rr.T() & Rb;           // 参照座標系からみた物体座標系の姿勢行列
        vector angular_vel = Rr.T() & ( model_.v(bodyID_).w() - model_.v(referenceID).w() );    /* 参照座標系で記述した，参照座標系に対する物体座標系の角速度ベクトル */

        moment[1] = gain_[0] * ( target[0] - atan2( rRb.xz(), rRb.zz() ) ) + gain_[1] * ( target[1] - angular_vel[1] );  // 参照座標系で記述した PD 入力トルク
        if( gain_[2] < moment[1] )              moment[1] = gain_[2];           // 最大トルク設定
        else if( moment[1] < -gain_[2] )    moment[1] =-gain_[2];

//      if (model_.debug)           デバック用表示
```

```
//      {
            Info<< " kikutA: " << endl
            << " ID:    " << bodyID_ << "reference: " << reaction_ << referenceID << endl
            << " gain " << gain_
            << " target angle " << deg(target[0]) << ", " << deg(atan2( rRb.xz(), rRb.zz() )) << endl
            << " target vel " << deg(target[1]) << ", " << deg(angular_vel[1]) << endl
            << " Rr    " << Rr << endl
            << " Rb    " << Rb << endl
            << " rRb " << rRb << endl
            << " P    " << (gain_[0] * ( target[0] - atan2( rRb.xz(), rRb.zz() ) ))
            << " D    " << (gain_[1] * ( target[1] - angular_vel[1] ) )

            << " moment(P+D) " << moment[1] << endl;
//      }

    // Accumulate the ONLY MOMENT for the restrained body
    fx[bodyIndex_]    += spatialVector( Rr & moment, Zero /*force*/ );          // 入力トルクを参照座標系から基準座標系に変換
    fx[referenceID] -= spatialVector( Rr & moment, Zero /*force*/ );          // 反トルクを参照座標系に出力
}

bool Foam::RBD::restraints::torqueControl::read                          // 辞書からパラメータを読み込む関数
(
    const dictionary& dict
)
{
    restraint::read(dict);
    coeffs_.readEntry("gain", gain_);

    torqueControl_ = Function1<vector>::New("target", coeffs_);
    coeffs_.readEntry("reaction", reaction_);

    return true;
}

void Foam::RBD::restraints::torqueControl::write                         // 書き込み関数
(
    Ostream& os
) const
{
    restraint::write(os);
    os.writeEntry("gain", gain_);
    os.writeEntry("reaction", reaction_);
    torqueControl_().writeData(os);
}
```

次に，ライブラリをメイクするために，以下のファイルを書き換える．

basilisk03¥lib¥rigidBodyDynamics¥Make¥files

. . . . .
LIB = $(FOAM_USER_LIBBIN)/librigidBodyDynamicsbasilisk03

ライブラリをおく場所である．ユーザー用に$FOAM_USER_LIBBIN に場所が決められている．もとのライブラリの格納場所よりも優先順位が高くされているので，たとえ同じファイル名のライブラリだとしてもこちらの方が優先される．
また，このライブラリを呼び出す以下のライブラリもメイクする．こちらは実際には何も書き換えられていない．

basilisk03¥lib¥rigidBodyMeshMotion¥Make¥files

LIB = $(FOAM_USER_LIBBIN)/librigidBodyMeshMotionbasilisk03

basilisk03¥lib¥rigidBodyMeshMotion¥Make¥options
このファイルも書き換えておく．

```
EXE_INC = ¥
    -I$(LIB_SRC)/finiteVolume/lnInclude ¥
    -I$(LIB_SRC)/fileFormats/lnInclude ¥
    -I$(LIB_SRC)/meshTools/lnInclude ¥
    -I../rigidBodyDynamics/lnInclude ¥
    -I$(LIB_SRC)/functionObjects/forces/lnInclude ¥
    -I$(LIB_SRC)/dynamicMesh/lnInclude

LIB_LIBS = ¥
    -lfiniteVolume ¥
    -lmeshTools ¥
    -L$(FOAM_USER_LIBBIN) -lrigidBodyDynamicsbasilisk03 ¥ /* このライブラリは，上記で作成したライブラリ．-L はパス指定 */
    -lforces ¥
    -ldynamicMesh
```

作業が煩雑なので，一気にこれらを実行するバッチファイルを作っておく．

.Librun

```
rm -r lib/rigidBodyDynamics/Make/linux64Gcc63DPInt32Opt        /* 以前に作った三つのファイルを消す． OS によって異なるので注意 */
rm -r lib/rigidBodyMeshMotion/Make/linux64Gcc63DPInt32Opt
rm $FOAM_USER_LIBBIN/librigidBodyDynamicsbasilisk03.so
rm $FOAM_USER_LIBBIN/librigidBodyMeshMotionbasilisk03.so

cd lib                          /*  rigidBodyDynamics ライブラリ作成ルーチン */
cd rigidBodyDynamics
wmake libso                     /* ライブラリ作成コマンド */

cd ..                           /* rigidBodyMeshMotion ライブラリ作成ルーチン */
cd rigidBodyMeshMotion
wmake libso

cd ..
cd ..
```

> Librun

でライブラリができる．実際の場所は，以下．kikut はユーザー名

home¥kikut¥OpenFOAM¥kikut-v2006¥platforms¥linux64Gcc63DPInt32Opt¥lib

8. 実行

上記において，重合格子計算空間は oversetmesh***ディレクトリ内にそれぞれ生成されているとして，外側の計算空間を生成する．system ディレクトリが存在している basilisk03/の場所から

> blockMesh

を行うと，外側の計算空間が constant ディレクトリの下に生成される．中央部を細分化するために，まず，細分化する空間を設定する．

> topoSet -dict ./system/topoSetDict2

で topoSetDict2 の辞書に書かれた空間を設定する．次に，

> refineMesh -overwrite

によって，その内部を細分化する．次に，重合格子計算空間を外部計算空間と合成する．

> mergeMeshes . ./oversetmeshbody -overwrite

を実行すると，./constant 以下にある外側の計算空間と，./oversetmeshbody/constant 以下にある重合格子の空間を結合し，./constant 以下のファイルに上書きする．同様に，

> mergeMeshes . ./oversetmeshlegR -overwrite
> mergeMeshes . ./oversetmeshlegL -overwrite
> mergeMeshes . ./oversetmeshfootR -overwrite
> mergeMeshes . ./oversetmeshfootL -overwrite

で 5 つすべての重合格子空間を統合する．次に，

> topoSet

で結合した境界面に名前を付ける．これで境界面が確定したので，境界値を設定できるようになる．

> . $WM_PROJECT_DIR/bin/tools/RunFunctions

を実行して，以下のコマンドを実行できるようにする．

> restore0Dir

これで，0.org の定義に従い，初期時間における境界値を生成し，0 ディレクトリを作成してくれる．

> setFields

を実行し，空気と水の設定をする．

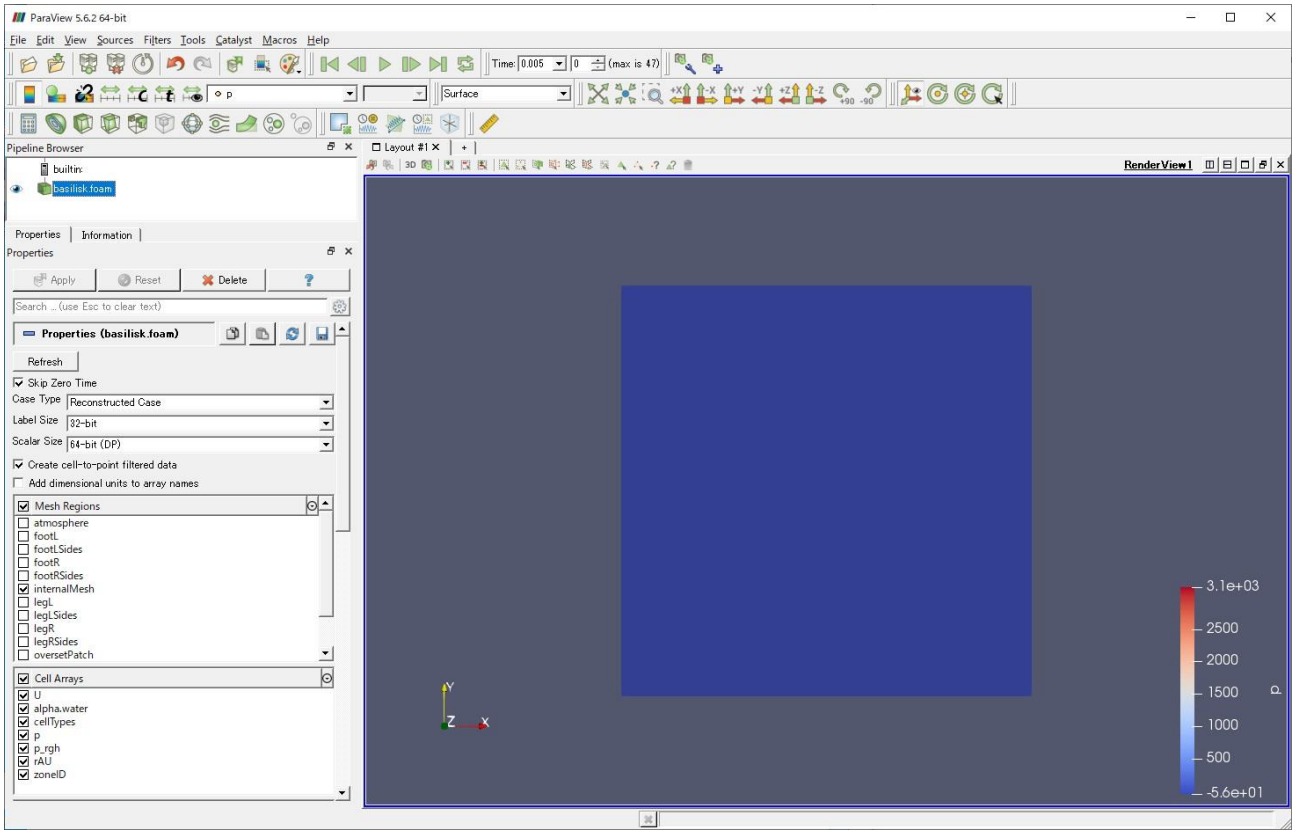
> overInterDyMFoam > faom.log

を実行し数値計算を行う．結果が foam.log に書き込まれる．並列化に関しては別資料参照のこと．また，再実行に関しては，実行により生成されたファイルを全て消しておかなければならない．

9. paraview による可視化

Linux から paraFoam を実行し，XWindow で Windows 上で表示する方法と，Windows 版の paraview を実行するやり方がある．インストール方法，実行方法に関しては，別ファイル参照のこと．ここでは，Windows 版で説明する．

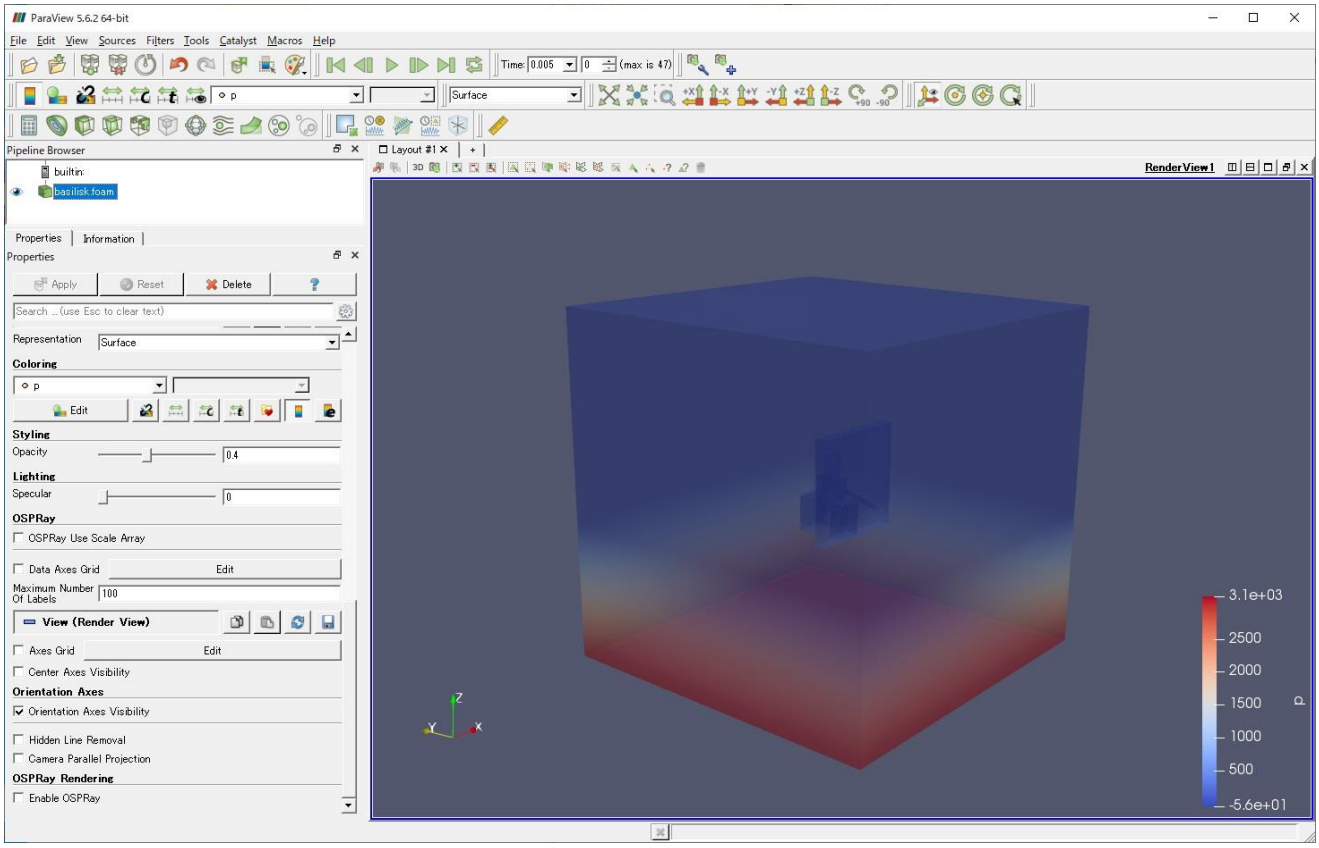
　まず，basilisk.foam という空ファイルをダブルクリックして paraview を立ち上げ，「Apply」をクリックすると，計算結果ファイルが自動で読み込まれる．



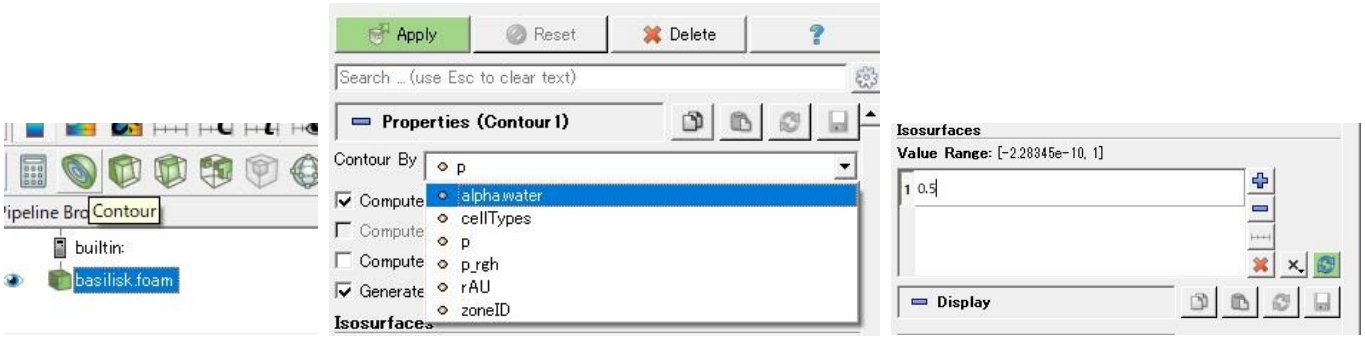
画面内で，物体を左クリックしながら動かすと回転でき，真ん中クリックしながら動かすと並進でき，右クリックでスケールを変更できる．「Opacity」（不透明性）を「0.4」（40%）にすると，以下のような，内部の重合格子空間が透過された図になる．境界面の圧力が表示されている．右のスケール（レジェンド）を見ると，-5.6*10^1～3.1*10^3 Pa であることが分かる．下の赤が3100Pa ということになる．水の密度を 998.2kg/m^3，重力を 9.81m/s^2 と設定したので，水深 0.4-0.08=0.32 は，

ρ gh=998.2*9.81*0.32=3133.55...

ということで正しく計算されていることが分かる．



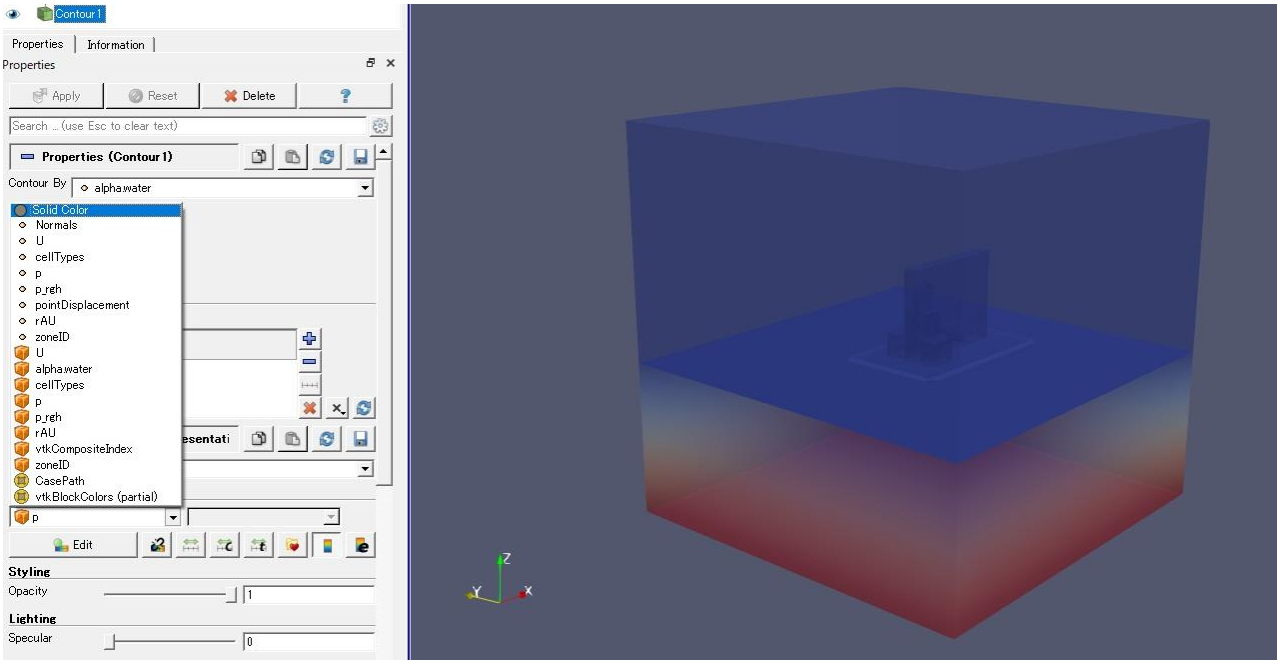
次に，水面を表示する．コンタ図を描くために「Contour」を選択し，表示物理量として「alpha.water」を選択し，表示面として「0.5」の値を入力して，「Apply」する．

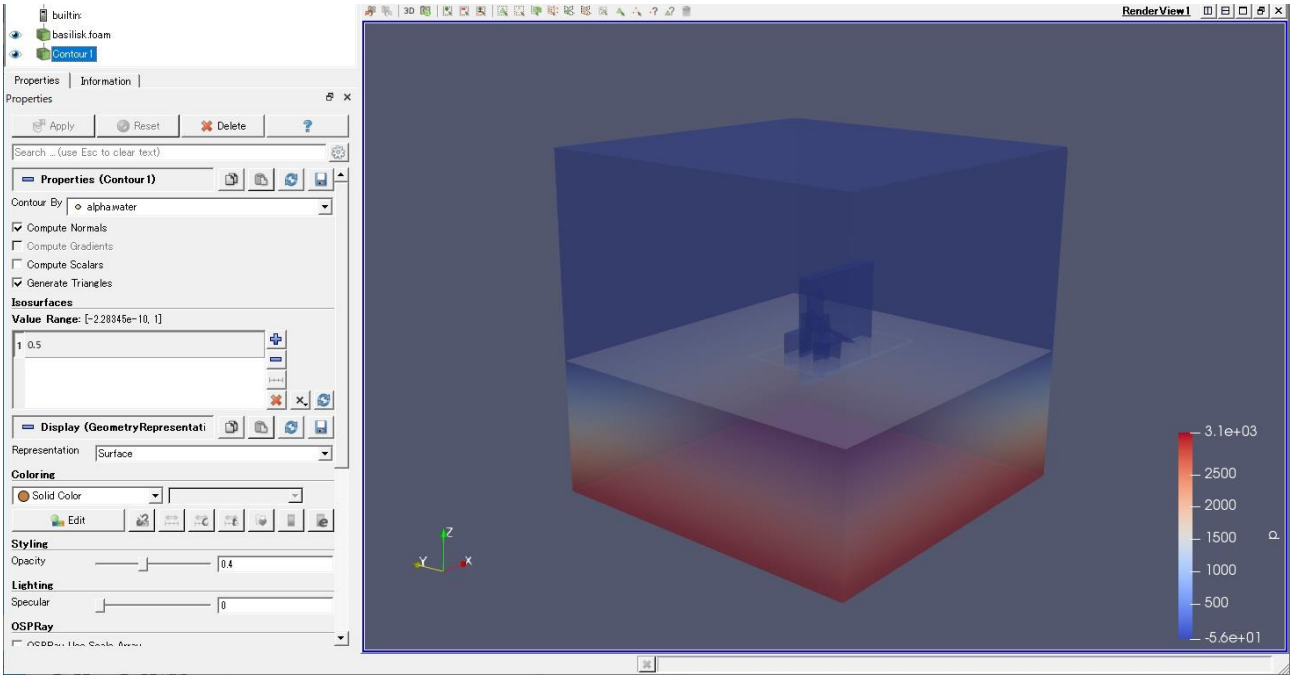


次に色の表示として，「Solid Color」を選ぶと，水面が表示されるので，こちらも，「Opacity」（不透明性）を「0.4」（40%）にして透過色にしておく．なお，alpha.water は，密度を表す式

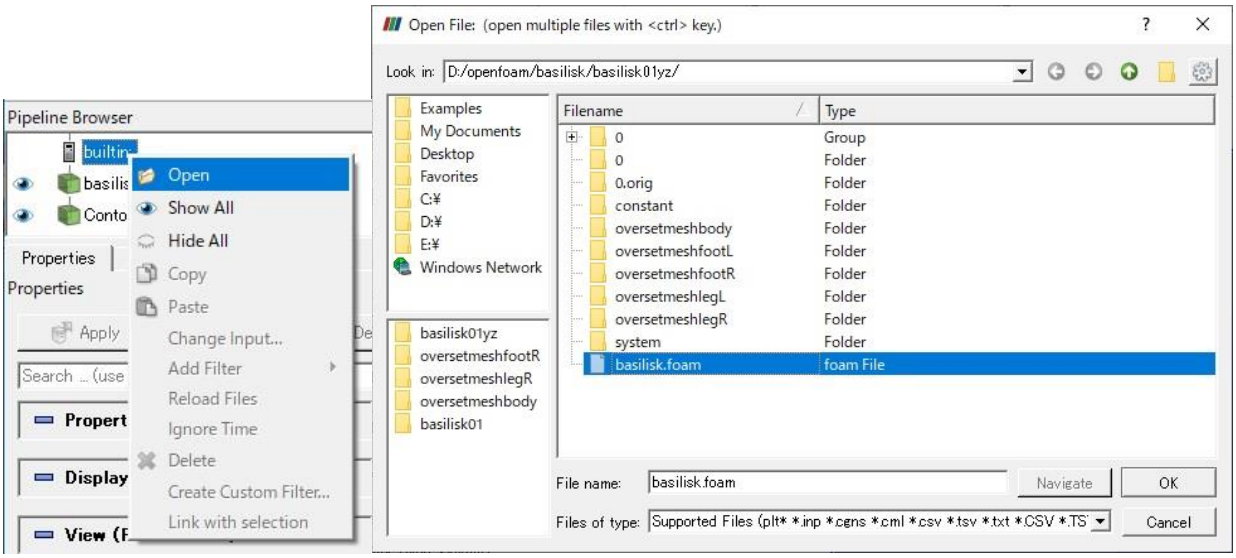
$\rho = \alpha\ \rho_{\mathrm{water}} + (1-\alpha)\,\rho_{\mathrm{air}}$

における α である． $\alpha=1$ が水， $\alpha=0$ が空，$0<\alpha<1$ が水面ということになる．なお，透過色にすると，重合格子の部分は複数回描写されるため濃く表示される．重なっている外側の計算空間を表示しない方法があるかもしれない．

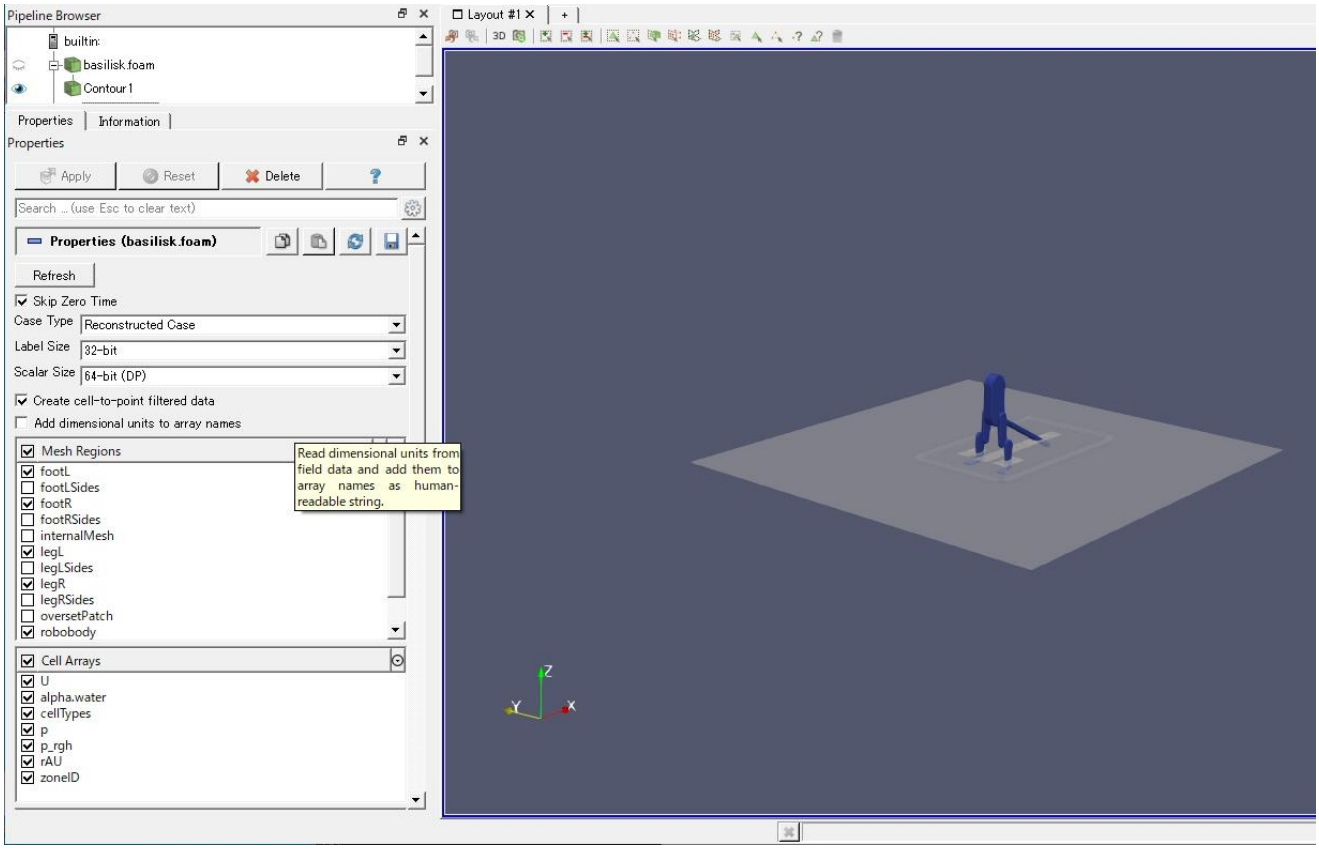




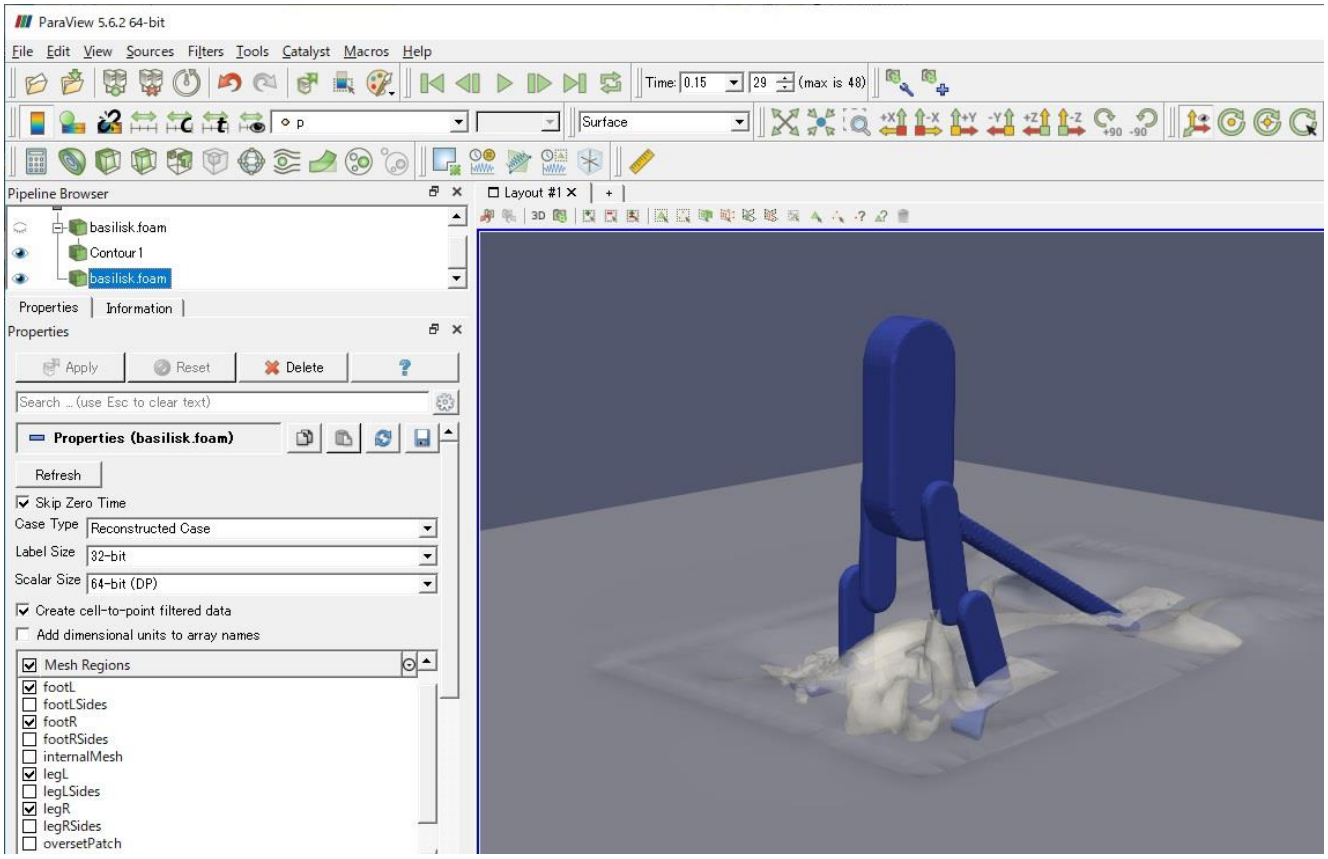
次に，バシリスクを表示する．「builtin:」を右クリックし，「Open」を選択し，もう一度，「basilisk.foam」を読み込む．



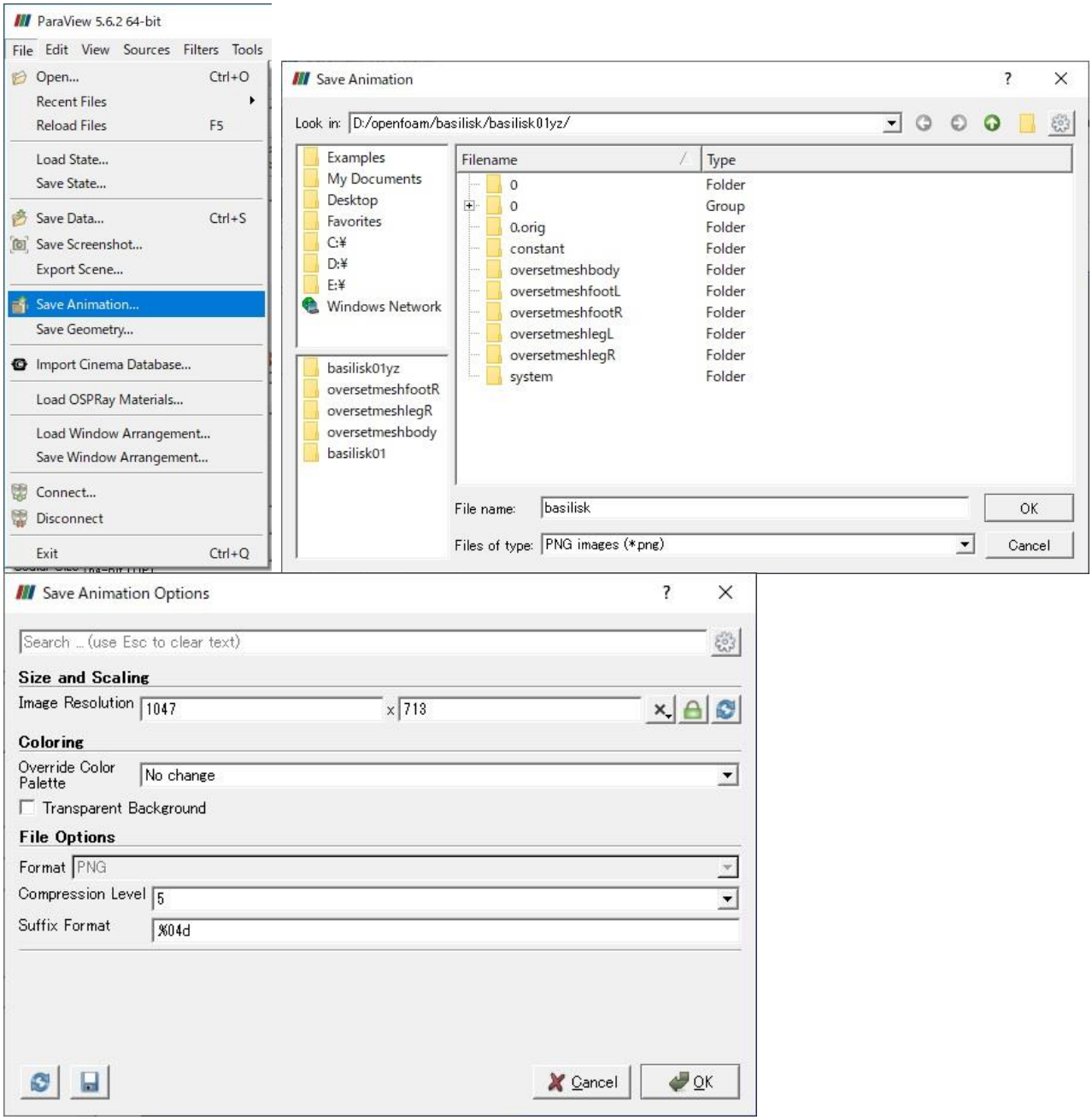
左側にある「Mesh Regions」を選ぶラジオボタンにおいて，”robobody”，”legR”, ”legL”, “footR”, “footL”を選択し，「basilisk.foam」を選択しなおして非表示にし，「Appy」すると，水面とバシリスクのみが表示される．



「Time:」において，29 ステップ目（0.15s 後）を表示させると，以下のようになる．バシリスクの色は，この時点では圧力になっている．水面下の圧力が高いため，空気中の圧力は相対的に低い色，青になる．



動画は，「File」の「Save Animation」を選択し，適当なファイル名「basilisk」をつけ，解像度を設定することで時系列の静止画として保存できる．この後，別ソフトでつなげて動画にする．



水面反力など，時系列データの取得などは後日．