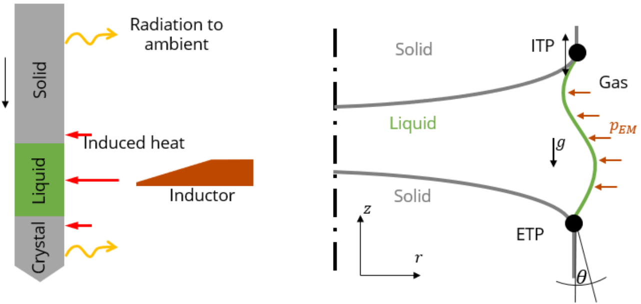
Free surface shape calculation using the interfaceTrackingFvMesh class and considering external pressure and fixed contact angles

Iason Tsiapkinis

Leibniz-Institute for Crystal Growth
Berlin, Germany

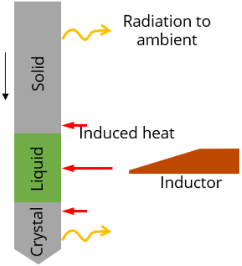
January 16, 2023

Motivation

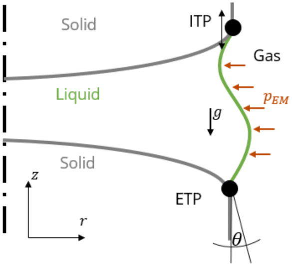


Left: Sketch of Floating Zone (FZ) process. Right: Free surface deformation.

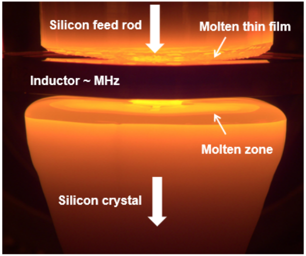
Motivation: Floating Zone (FZ) growth of silicon crystals



Sketch FZ process



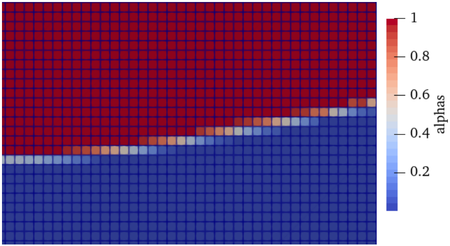
Free surface deformation.



FZ growth process at 1500°C

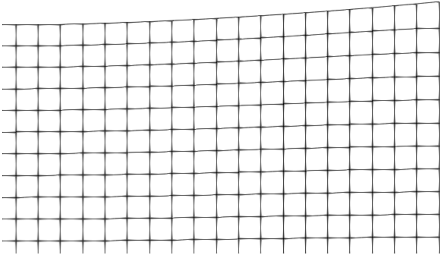
- High density ratios between solid and liquid
- High surface tension
- Contact conditions at external and internal triple points (ETP, ITP)

Multiphase methods



Interface-Capturing-Technique

- Volume-of-Fluids, Level-Set, etc.
- Diffusive interface ($\alpha = [0, 1]$)



Interface-Tracking-Technique

- Sharp interface by boundary-fitted moving mesh
- Interface = Computational boundary

Learning Outcomes

How to use it:

- How to use the `interfaceTrackingFvMesh` class and the accompanying `freeSurface` boundary conditions.
- How to set up a test case including free surface mesh deformation using `interfaceTrackingFvMesh`.
- How to set up contact conditions at the liquid-gas-solid interface.

The theory of it:

- The theory behind the interface tracking method for finite volumes.
- The theory behind the methods of the Finite-Area-Method utilized in this class.
- The numerical technique for a contact angle constrained by an adapted pressure boundary condition.

Learning Outcomes

How it is implemented:

- How the interface tracking class `interfaceTrackingFvMesh` is designed and implemented in OpenFOAM .
- How the `freeSurfaceVelocity` and `freeSurfacePressure` boundary conditions are implemented and applied.
- The differences between the `interTrackFoam` solver in foam-extend and the `interfaceTrackingFvMesh` class.

How to modify it:

- How to modify the class to calculate, write and use additional variables
- How to write out additional surface data.
- How to modify the implementation of the contact angle condition to make it more strict
- How to implement the contact angle condition inside the pressure boundary condition

Interface Tracking Method

Mathematical model and implementation follow the descriptions in

- S. Muzaferija and M. Perić, "Computation of free-surface flows using the finite-volume-method and moving grids", *Numerical Heat Transfer, Part B: Fundamentals*, vol. 32, pp. 369-384, 1997
- Z. Tuković and H. Jasak, "A moving mesh finite volume interface tracking method for surface tension dominated interfacial fluid flow", *Computers & Fluids*, vol. 55, pp. 70-84, 2012

Interface tracking is implemented

- in OpenFOAM as a dynamic mesh library `interfaceTrackingFvMesh`
- in foam-extend as a solver `interTrackFoam`

Governing equations

Incompressible and isothermal Navier-Sokes equations for a Newtonian fluid:

$$\begin{aligned} \nabla \cdot \boldsymbol{u} &= 0 \\ \rho \left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{u} \right) &= \nabla \cdot \boldsymbol{\tau} \end{aligned}$$

where ρ is the fluid density and $\boldsymbol{u}$ is the fluid velocity.

Stress tensor:

$$\boldsymbol{\tau} = \eta \left(\nabla \boldsymbol{u} + (\nabla \boldsymbol{u})^T \right) - p\boldsymbol{I}$$

where

- ρ is the fluid density
- $\boldsymbol{u}$ is the fluid velocity.
- η: dynamic viscosity of the fluid
- p: dynamic pressure obtained by subtracting the hydrostatic pressure $\rho \boldsymbol{g} \cdot \boldsymbol{x}$ from the absolute pressure.

Kinematic Condition at the free surface

- The *kinematic condition* requires the free surface to not "break"
- No mass flux through the phase boundaries for immiscible fluids
- Velocity must be continuous across the interface

$$[\![\boldsymbol{u}]\!] = 0.$$

where $[\![\cdot]\!]$ of a quantity ψ is defined as

$$[\![\psi]\!] = \psi^{+} - \psi^{-}.$$

- $+$: "outer" fluid
- $-$: "inner" fluid

Dynamic Condition at the free surface

- The *dynamic condition* is derived from the *momentum balance* at the surface
- Forces acting on a fluid are in an equilibrium
- Jump of stress tensor:

$$[\![\boldsymbol{n}\boldsymbol{\tau}]\!] = \bar{\nabla}\sigma - \sigma\kappa\boldsymbol{n},$$

- σ: surface tension (can vary with concentrations and/or temperature)
- κ: mean surface curvature (in $1/m$)

$$\kappa = -\bar{\nabla} \cdot \boldsymbol{n}.$$

- Surface gradient operator

$$\bar{\nabla} = (\boldsymbol{I} - \boldsymbol{n}\boldsymbol{n}) \cdot \nabla = \nabla - \boldsymbol{n}\frac{\partial}{\partial n}$$

Dynamic Condition at the free surface

Jump of stress tensor:

$$[\![\boldsymbol{n}\boldsymbol{\tau}]\!] = \bar{\nabla}\sigma - \sigma\kappa\boldsymbol{n},$$

Balance in **normal direction** yields

$$[\![p]\!] = -2[\![\eta]\!] \left(\bar{\nabla} \cdot \boldsymbol{u}\right) + \sigma\kappa.$$

$\rightarrow$ Jump in dynamic pressure due to the surface divergence of the velocity $\boldsymbol{u}$ and the local curvature κ.

Balance in **tangential direction** yields

$$[\![\boldsymbol{n} \cdot \eta\nabla u]\!] = -[\![\eta]\!](\bar{\nabla} \cdot \boldsymbol{u})\boldsymbol{n} - [\![\eta]\!]\bar{\nabla}\boldsymbol{u} \cdot \boldsymbol{n} - \bar{\nabla}\sigma.$$

$\rightarrow$ Jump in the normal gradient of the velocity by the surface gradient of its normal component, its surface divergence, and the change of surface tension along the curvature.

Dynamic Condition at the free surface

Balance in **normal direction** yields

$$[\![p]\!] = -2[\![\eta]\!]\left(\bar{\nabla}\cdot\boldsymbol{u}\right) + \sigma\kappa.$$

Balance in **tangential direction** yields

$$[\![\boldsymbol{n}\cdot\eta\nabla u]\!] = -[\![\eta]\!](\bar{\nabla}\cdot\boldsymbol{u})\boldsymbol{n} - [\![\eta]\!]\bar{\nabla}\boldsymbol{u}\cdot\boldsymbol{n} - \bar{\nabla}\sigma.$$

Let us assume that we do not want to include the external fluid "+" and $\eta^{+} \ll \eta^{-}$:

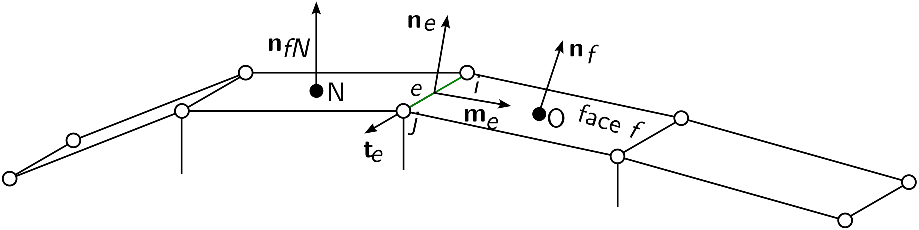
Boundary conditions at the free surface

$$\begin{aligned} p &= p_{\mathrm{a}} - \rho\boldsymbol{g}\cdot\boldsymbol{x}_f - 2\eta\left(\bar{\nabla}\cdot\boldsymbol{u}\right) - \sigma\kappa \\ \boldsymbol{n}\cdot\eta\nabla u &= -\left(\bar{\nabla}\cdot\boldsymbol{u}\right)\boldsymbol{n} - \bar{\nabla}\boldsymbol{u}\cdot\boldsymbol{n} + \frac{1}{\eta}\bar{\nabla}\sigma \end{aligned}$$

Finite-Area-Method (FAM)

- FAM to calculate surface divergence $\bar{\nabla} \cdot \boldsymbol{u}$, surface gradient $\bar{\nabla} \boldsymbol{u}$ as well as the curvature $\kappa = \bar{\nabla} \cdot \boldsymbol{n}$
- FAM is similar to FVM **but**:

FVM	**FAM**
cells	→ faces
faces	→ edges
volume integrals	→ surface intergrals
surface integrals	→ line intergrals



Surface area mesh with a one-dimensional curvature. Normal vector $\boldsymbol{n}_e$ is calculated using the normal vectors of neighboring faces.

Interface tracking procedure

Mesh flux through faces is **not** zero after applying the boundary conditions
→ mesh has to be moved according to surface velocity:

$$\boldsymbol{v}_f = \boldsymbol{u}_f.$$

$$\dot{V}_f = \boldsymbol{S}_f \cdot \boldsymbol{v}_f = \boldsymbol{S}_f \cdot \boldsymbol{u}_f = \frac{m_f}{\rho_f}.$$

Volume flux correction:

$$\dot{V}_f' = \frac{m_f}{\rho_f} - \dot{V}_f.$$

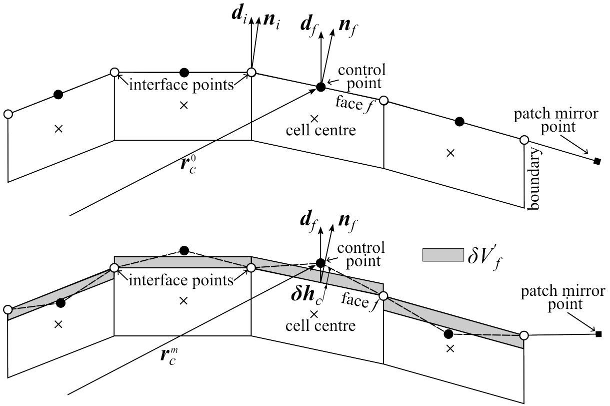
Absolute change in cell volume (i.e. the volume change needed in this time step):

$$\delta V_f' = C_{\mathrm{ddt}} \dot{V}_f' \Delta t,$$

- $\boldsymbol{u}_f$: fluid velocity at a face f
- $\boldsymbol{v}_f$: mesh velocity at a face f
- $\boldsymbol{S}_f$: Surface area vector at face f
- m_f: mass flux through face f
- $\dot{V}_f$: Volume change of cell with face f

Interface tracking procedure

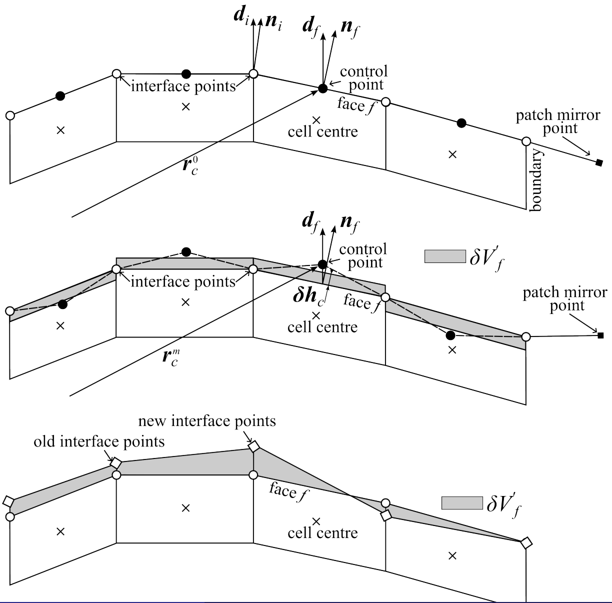
- Using control points instead of mesh points to ensure a smooth surface
- At the start: control points are at face centers
- $\delta V_f'$: The volume needed to be swept by face f



$$\delta h_f' = \frac{\delta V_f'}{S_f \boldsymbol{n}_f \cdot \boldsymbol{d}_f}.$$

$\boldsymbol{d}_f$: motion direction vector

Interface tracking procedure



- $$\delta h'_f = \frac{\delta V'_f}{S_f \boldsymbol{n}_f \cdot \boldsymbol{d}_f}.$$
- $\boldsymbol{d}_f$: motion direction vector
- least-squares-fit of interface points

Algorithm

This is done inside one outer correction when the dynamic mesh motion is called:

1. Start dynamic mesh motion
   1. Update values
   2. Interface Tracking procedure
      1. Calculate the cell volume change correction $\delta \dot{V}'_f$ to account for current relative flux through the interface cells
      2. Calculate the required displacement $\delta \dot{h}'_f$
      3. Define and move the `controlPoints`
      4. Calculate `patchMirrorPoints` mirroring `controlPoints` for boundary mesh points with neighboring face normal vectors
      5. Do a least-squares plane fit for each mesh point using `controlPoints`
      6. calculate the displacement of free surface mesh points
   3. Displacement of the inner mesh with mesh motion solver.
2. Update the pressure and velocity boundary conditions with FAM discretization methods
3. Assemble and solve the discretised momentum equation on the new interface shape
4. Do at least 2 PISO iteration loops
5. Calculate new mass fluxes through the faces at the interfaces and restart the outer loop at point 1 if the number of outer correction loops is not reached.

interfaceTrackingFvMesh Library

- $FOAM_SCR/dynamicFaMesh/interfaceTrackingFvMesh
- Inherits from dynamicMotionSolverFvMesh
- The calculated displacements of the interface points are used for the dynamic mesh motion problem
- FAM is implemented separately in $FOAM_SCR/finiteArea/include/faCFD.H

interfaceTrackingFvMesh Library

$FOAM_SCR/dynamicFaMesh/interfaceTrackingFvMesh

```
├── boundaryProcessorFaPatchPoints.H
├── freeSurfacePointDisplacement.C
├── functionObjects
│   ├── pointHistory
│   │   ├── pointHistory.C
│   │   └── pointHistory.H
│   └──writeFreeSurface
│       ├── writeFreeSurface.C
│       └── writeFreeSurface.H
├── fvPatchFields
│   ├── freeSurfacePressure
│   │   ├── freeSurfacePressureFvPatchScalarField.C
│   │   └── freeSurfacePressureFvPatchScalarField.H
│   └──freeSurfaceVelocity
│       ├── freeSurfaceVelocityFvPatchVectorField.C
│       └── freeSurfaceVelocityFvPatchVectorField.H
├── interfaceTrackingFvMesh.C
├── interfaceTrackingFvMesh.H
├── solveBulkSurfactant.H
└── surfactantProperties.H
```

Boundary condition freeSurfacePressure

Pressure boundary condition at the free surface

$$\mathrm{p} = \mathrm{p_a} - \rho \boldsymbol{g} \cdot \boldsymbol{x}_f - 2\eta \left(\bar{\nabla} \cdot \boldsymbol{u} \right) - \sigma \kappa$$

freeSurfacePressureFvPatchScalarField.C - updateCoeffs()

```
    interfaceTrackingFvMesh& itm =
        refCast<interfaceTrackingFvMesh>
        (
            const_cast<dynamicFvMesh&>
            (
                mesh.lookupObject<dynamicFvMesh>("fvSolution")
            )
        );

    operator==
    (
        pa_ + itm.freeSurfacePressureJump()
    );

    fixedValueFvPatchScalarField::updateCoeffs();
```

Boundary condition `freeSurfacePressure`

Pressure boundary condition at the free surface

$$\mathrm{p} = \mathrm{p_a} - \rho \boldsymbol{g} \cdot \boldsymbol{x}_f - 2\eta \left(\bar{\nabla} \cdot \boldsymbol{u} \right) - \sigma\kappa$$

`interfaceTrackingFvMesh.C`: `freeSurfacePressureJump()`

```
Foam::tmp<scalarField>
Foam::interfaceTrackingFvMesh::freeSurfacePressureJump()
{
    auto tPressureJump = tmp<scalarField>::New(aMesh().nFaces(), Zero);
    auto& pressureJump = tPressureJump.ref();

    const scalarField& K = aMesh().faceCurvatures().internalField();

    const uniformDimensionedVectorField& g =
        meshObjects::gravity::New(mesh().time());

    const turbulenceModel& turbulence =
        mesh().lookupObject<turbulenceModel>("turbulenceProperties");

    scalarField nu(turbulence.nuEff(fsPatchIndex()));
```

Boundary condition freeSurfacePressure

Pressure boundary condition at the free surface

$$\mathrm{p} = \mathrm{p_a} - \rho \boldsymbol{g} \cdot \boldsymbol{x}_f - 2\eta \left(\bar{\nabla} \cdot \boldsymbol{u} \right) - \sigma \kappa$$

interfaceTrackingFvMesh.C: freeSurfacePressureJump()

```
    pressureJump =
      - (g.value() & mesh().Cf().boundaryField()[fsPatchIndex()])
      + 2.0*nu*freeSurfaceSnGradUn();

    if (pureFreeSurface())
    {
        pressureJump -= sigma().value()*K;
    }
    else
    {
        pressureJump -= surfaceTension().internalField()*K;
    }

    return tPressureJump;
}
```

Boundary condition `freeSurfacePressure`

Pressure boundary condition at the free surface

$$\mathrm{p} = \mathrm{p_a} - \rho \boldsymbol{g} \cdot \boldsymbol{x}_f - 2\eta \left(\bar{\nabla} \cdot \boldsymbol{u} \right) - \sigma \kappa$$

`interfaceTrackingFvMesh.C`: `freeSurfaceSnGradUn()`

```
Foam::interfaceTrackingFvMesh::freeSurfaceSnGradUn()
{
    auto tSnGradUn = tmp<scalarField>::New(aMesh().nFaces(), Zero);
    auto& SnGradUn = tSnGradUn.ref();

    areaScalarField divUs
    (
        fac::div(Us())
      - aMesh().faceCurvatures()*(aMesh().faceAreaNormals()&Us())
    );

    SnGradUn = -divUs.internalField();

    return tSnGradUn;
}
```

Boundary condition freeSurfaceVelocity

Velocity boundary condition at the free surface

$$\boldsymbol{n} \cdot \eta \nabla u = -\left(\bar{\nabla} \cdot \boldsymbol{u}\right) \boldsymbol{n} - \bar{\nabla} \boldsymbol{u} \cdot \boldsymbol{n} + \frac{1}{\eta} \bar{\nabla} \sigma$$

freeSurfaceVelocityFvPatchVectorField.C - updateCoeffs()

```
    const fvMesh& mesh = patch().boundaryMesh().mesh();

    interfaceTrackingFvMesh& itm =
        refCast<interfaceTrackingFvMesh>
        (
            const_cast<dynamicFvMesh&>
            (
                mesh.lookupObject<dynamicFvMesh>("fvSolution")
            )
        );

    gradient() = itm.freeSurfaceSnGradU();

    fixedGradientFvPatchVectorField::updateCoeffs();
}
```

Boundary condition `freeSurfaceVelocity`

Velocity boundary condition at the free surface

$$\boldsymbol{n} \cdot \eta \nabla u = -\left(\bar{\nabla} \cdot \boldsymbol{u}\right) \boldsymbol{n} - \bar{\nabla} \boldsymbol{u} \cdot \boldsymbol{n} + \frac{1}{\eta} \bar{\nabla} \sigma$$

`interfaceTrackingFvMesh.C`: `freeSurfaceSnGradU`

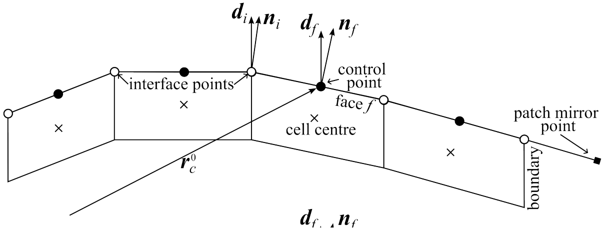
```
    SnGradU =
        tangentialSurfaceTensionForce/(nu + SMALL)
      - nA*divUs.internalField()
      - (gradUs.internalField()&nA);
```

- The term `tangentialSurfaceTensionForce` is 0 for pure surfaces or calculated in `surfaceTensionGrad`
- `divUs` is calculated from as previously
- Surface gradient of `Us` is calculated as only the tangential part of the gradient on the surface:

$$\bar{\nabla} = (\boldsymbol{I} - \boldsymbol{nn}) \cdot \nabla = \nabla - \boldsymbol{n} \frac{\partial}{\partial n},$$

Contact angle condition with patchMirrorPoints

- Using control points instead of mesh points to ensure a smooth surface
- At the start: control points are at face centers
- `patchMirrorPoints` are mirrored `controlPoints`
- calculated in `pointDisplacement()` function in `freeSurfacePointDisplacement.C`
- lines 119-197



$$\delta h_f' = \frac{\delta V_f'}{S_f \boldsymbol{n}_f \cdot \boldsymbol{d}_f}.$$

$\boldsymbol{d}_f$: motion direction vector

Contact angle condition with patchMirrorPoints

Get normal vector of neighboring wall:

```
        vectorField N
        (
            aMesh().boundary()[patchI].ngbPolyPatchFaceNormals()
        );
```

N is rotated with Rodriguez formula as

$$\boldsymbol{N}^{\mathrm{rot}} = \boldsymbol{N}\cos\theta + \boldsymbol{e}_r\left(\boldsymbol{e}_r\cdot\boldsymbol{N}\right)\left(1-\cos\theta\right) + \left(\boldsymbol{e}_r\times\boldsymbol{N}\right),$$

with $\theta = 90$ - contactAngle

Contact angle condition with patchMirrorPoints

```
        // Correct N according to specified contact angle
        if (contactAnglePtr_)
        {
            label ngbPolyPatchID =
                aMesh().boundary()[patchI].ngbPolyPatchIndex();

            if (ngbPolyPatchID != -1)
            {
                if
```

This is only done if the contactAnglePtr_ exists, the neighboring fv-Patch is of type wall, and the contactAngle boundary is of type calculated. The location $\boldsymbol{r}^{\mathrm{PMP}}$ of the patchMirrorPoints is then calculated with

$$\boldsymbol{r}^{\mathrm{PMP}} = \mathbf{r}^{\mathrm{eC}} + \left(\boldsymbol{I} - 2\boldsymbol{N}^{\mathrm{rot}}\boldsymbol{N}^{\mathrm{rot}}\right) \cdot \boldsymbol{\delta},$$

where $\mathbf{r}^{\mathrm{eC}}$ is the center of the edge, and $\boldsymbol{\delta}$ is the vector pointing from the edge center to the inner controlPoint. As can be seen from this formulation, the contact angle condition is applied in an indirect way by setting the patchMirrorPoints.

Contact angle condition with patchMirrorPoints

0/contactAngle

```
dimensions      [0 0 0 0 0 0 0];

internalField   uniform 0;

boundaryField
{
    left
    {
        type            calculated;
        value           uniform 70;
    }

    right
    {
        type            calculated;
        value           uniform 70;
    }
```

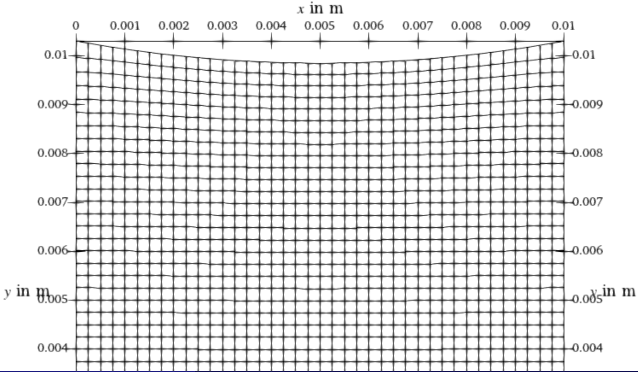
interfaceTrackingFvMesh::update()

```
        velocityMotionSolver& vMotion =
            refCast<velocityMotionSolver>
            (
                const_cast<motionSolver&>(motion())
            );

        pointVectorField& pointMotionU = vMotion.pointMotionU();
        pointMotionU.primitiveFieldRef() = Zero;

        fixedValuePointPatchVectorField& fsPatchPointMeshU =
            refCast<fixedValuePointPatchVectorField>
            (
                const_cast<pointPatchVectorField&>
                (
                    pointMotionU.boundaryField()[fsPatchIndex()]
                )
            );

        fsPatchPointMeshU ==
            displacement/mesh().time().deltaT().value();

        dynamicMotionSolverFvMesh::update();
```

Tutorial case

```
cp -r $FOAM_TUTORIALS/incompressible/pimpleFoam \\
      /laminar/contactAngleCavity .
./Allrun
```

- cavity with free surface at the top
- dynamic mesh motion with `interFaceTrackingFvMesh`
- fixed contact angles at the free surface boundaries



Case file structure

```
├── 0.orig
│   ├── U
│   ├── contactAngle
│   ├── p
│   └── pointMotionU
├── Allclean
├── Allrun
├── constant
│   ├── dynamicMeshDict
│   ├── g
│   ├── transportProperties
│   └── turbulenceProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── decomposeParDict
    ├── faMeshDefinition
    ├── faSchemes
    ├── faSolution
    ├── fvSchemes
    └── fvSolution
```

faMeshDefinition file

- Definition of Finite-Area-Mesh
- Similar to `constant/polyMesh/boundary`
- `polyMeshPatches`: name of the `polyPatch` that is converted to the Finite-Area-Mesh
- `boundaryDictionary`:

system/faMeshDefinition

```
    left
    {
        type                patch;
        neighbourPolyPatch  left;
    }

    right
    {
        type                patch;
        neighbourPolyPatch  right;
    }
```

dynamicMeshDict file

dynamicMeshDict

```
motionSolverLibs (fvMotionSolvers interfaceTrackingFvMesh);

dynamicFvMesh   interfaceTrackingFvMesh;

motionSolver    velocityLaplacian;

diffusivity     uniform; //onTimeChange inverseDistance 1(top);


// Free surface data

fsPatchName top;

fixedFreeSurfacePatches ( );

pointNormalsCorrectionPatches ();
// pointNormalsCorrectionPatches ( left right );

normalMotionDir false;

motionDir (0 1 0);
```

dynamicMeshDict file

- fsPatchName: Name of free surface polyPatch
- fixedFreeSurfacePatches: Names of fixed free surfaces. Here a free surface patch can be specified to not move
- pointNormalCorrectionPatches: Free surface patches for which point normals must be corrected
- pointNormalCorrectionPatches: Free surface patches where a wave should not be reflected
- normalMotionDir:
  true: motion is in point normal direction
  false: motion is in direction of motionDir

Additional entries are

- pureFreeSurface of type boolean. This is false by default. If set to true the surface tension is dependent on the surfactant concentrations. The properties can be set in the surfactantProperties dictionary

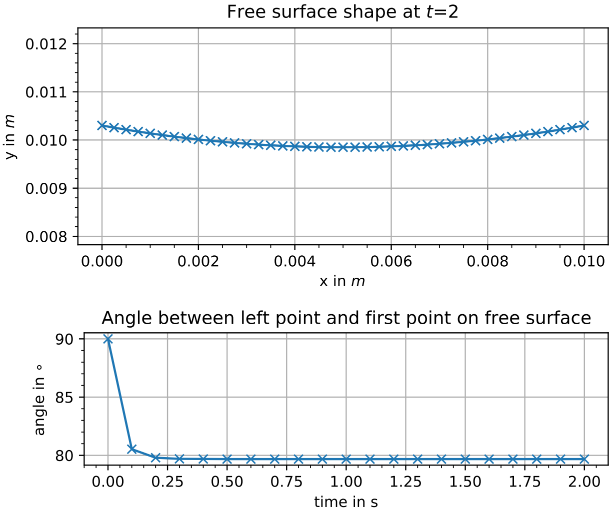
faMeshDefinition file

- Definition of Finite-Area-Mesh
- Similar to constant/polyMesh/boundary
- polyMeshPatches: name of the polyPatch that is converted to the Finite-Area-Mesh
- boundaryDictionary:

system/faMeshDefinition

```
    left
    {
        type                patch;
        neighbourPolyPatch  left;
    }

    right
    {
        type                patch;
        neighbourPolyPatch  right;
    }
```

Results



→ Resulting angle of 80° is to high in the **tutorial case of this library**

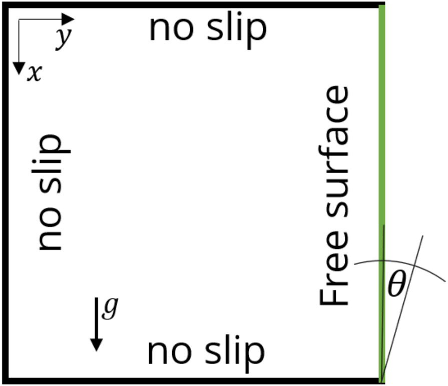
How to modify it:

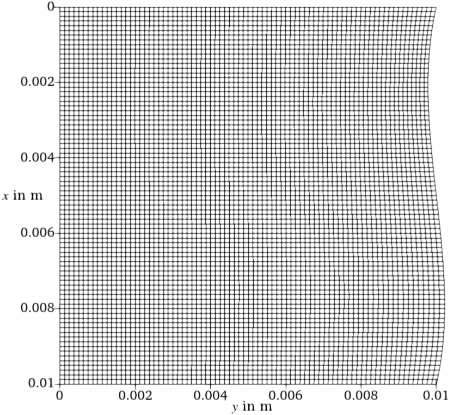
How to modify it:

- **How to modify the class to calculate, write and use additional variables**
- How to write out additional surface data.
- How to modify the implementation of the contact angle condition to make it more strict
- How to implement the contact angle condition inside the pressure boundary condition

New case setup

New case setup to resemble FZ-process with silicon fluid
$(\rho = 2580\frac{\mathrm{kg}}{\mathrm{m}^3},\ \nu = 3.333 \cdot 10^{-7}\frac{\mathrm{m}^2}{\mathrm{s}},\ \sigma = 0.88\frac{\mathrm{N}}{\mathrm{m}},\ \theta = 11°)$





Calculating `faceAngles`

The actual `contactAngle` is never calculated!

- fields on the free surface are handled with pointers and accessed through `public` member functions

Add private member data:

- `mutable areaScalarField* faceAnglesPtr_;`
- `vector verticalDir_;`

Add private member functions for creating the pointer and updating the values:

- `void makeFaceAngles() const;`
- `void updateFaceAngles()`

Add public member functions for accessing:

- `const areaScalarField& faceAngles() const;`
- `areaScalarField& faceAngles();`

Calculating faceAngles

interfaceTrackingFvMesh.C - faceAngles() member function

```
Foam::areaScalarField& Foam::myInterfaceTrackingFvMesh::faceAngles()
{
    if (!faceAnglesPtr_)
    {
        makeFaceAngles();
    }

    return *faceAnglesPtr_;
}


const Foam::areaScalarField& Foam::myInterfaceTrackingFvMesh::faceAngles() const
{
    if (!faceAnglesPtr_)
    {
        makeFaceAngles();
    }

    return *faceAnglesPtr_;
}
```

Calculating faceAngles

interfaceTrackingFvMesh.C - makeFaceAngles() member function

```
void Foam::myInterfaceTrackingFvMesh::makeFaceAngles() const
{
```

```
    faceAnglesPtr_ = new areaScalarField
    (
        IOobject
        (
            "faceAngles",
            mesh().time().timeName(),
            mesh(),
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        aMesh(),
        dimensionedScalar(Zero)
    );
}
```

Calculating faceAngles

interfaceTrackingFvMesh.C - updateFaceAngles() member function

```
void Foam::myInterfaceTrackingFvMesh::updateFaceAngles()
{
    // Calculate local angle of face
    const vectorField& Nf = aMesh().faceAreaNormals().internalField();

    forAll(faceAngles(), faceI)
    {
        faceAngles()[faceI] =  90 - radToDeg(acos((Nf[faceI]&verticalDir_)));
        faceAngles()[faceI] *= sign(aMesh().faceCurvatures()[faceI]);
    }
}
```

How to modify it:

How to modify it:

- How to modify the class to calculate, write and use additional variables
- **How to write out additional surface data.**
- How to modify the implementation of the contact angle condition to make it more strict
- How to implement the contact angle condition inside the pressure boundary condition

Write out surface data

Some `areaFields` are written to the time directory but ate not available in ParaView

- ParaView can not handle the FAM fields and mesh
- There is a `functionObject writeFreeSurface`, but that only writes the location of the `controlPoints`
- Idea 1: rewrite the `areaFields` as `volFields` but only on the `freeSurface` cells
- Idea 2: write out `areaFields` as `vtk` together with the surface mesh

Write out surface data

interfaceTrackingFvMesh.C - writeVTK() member function

```
void Foam::myInterfaceTrackingFvMesh::writeVTK() const
{
    // GenericPatchGeoFieldsWriter<uindirectPrimitivePatch>
    vtk::GenericPatchGeoFieldsWriter<uindirectPrimitivePatch> writer
    (
        aMesh().patch(),
        vtk::formatType::LEGACY_ASCII,
        mesh().time().timePath()/"freeSurface",
        false // serial only
    );
    writer.writeGeometry();
    writer.beginCellData(4);
    writer.write(Us());
    writer.write(fsNetPhi());
    writer.write(aMesh().faceCurvatures());
    writer.write(faceAngles());
}
```

Write out surface data

Add a call to this function in the function `writeData` in file `functionObjects/writeFreesurface/writeFreesurface.C`

```
itm.writeVTK();
```

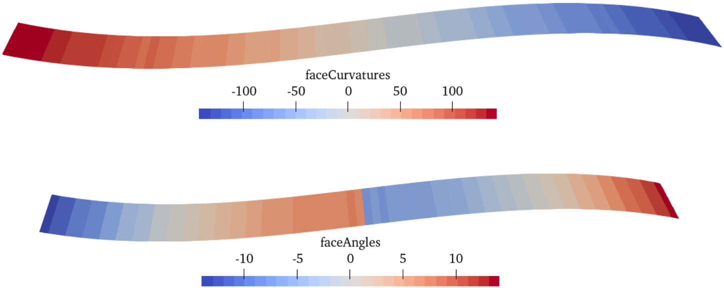
Using the `functionObject`:

```
writeFreeSurface
{
    type                    writeFreeSurface;
}
```

→ `freeSurface.vtk` is written in every `writeTime` with fields and mesh

Write out surface data

→ `freeSurface.vtk` is written in every `writeTime` with fields and mesh



How to modify it:

How to modify it:

- How to modify the class to calculate, write and use additional variables
- How to write out additional surface data.
- **How to modify the implementation of the contact angle condition to make it more strict**
- How to implement the contact angle condition inside the pressure boundary condition

Expanded `contactAngle` BCs

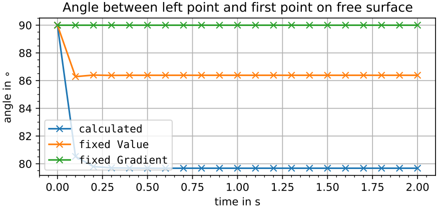
- `contactAngle` is not correct
- `contactAngle` has to be specified for all boundaries, i.e. no "free" condition possible

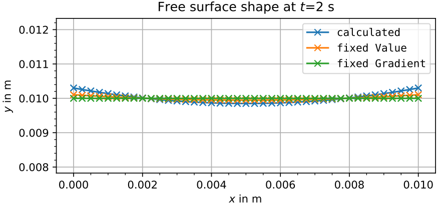
Adapted code in function `pointDisplacement()` in `freeSurfacePointDisplacement.C` to include additional "boundary conditions" for the `contactAngle`

- `type calculated`: uses the default calculation
- `type fixedValue`: uses a more "rigid" calculation
- `type zeroGradient`: no contact angle calculation is done

Expanded `contactAngle` BCs

- `type calculated`: uses the default calculation
- `type fixedValue`: uses a more "rigid" calculation
- `type zeroGradient`: no contact angle calculation is done





How to modify it:

How to modify it:

- How to modify the class to calculate, write and use additional variables
- How to write out additional surface data.
- **How to modify the implementation of the contact angle condition to make it more strict**
- How to implement the contact angle condition inside the pressure boundary condition

Contact angle boundary condition

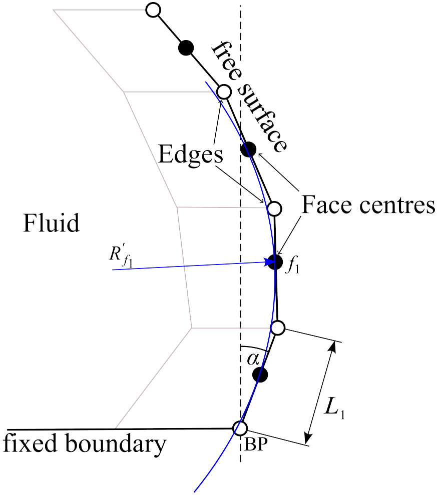
One-dimensional curvature:

$$\kappa_f = \frac{1}{R'_f},$$

$\rightarrow$ pressure boundary condition at each face:

$$p_f = p_\mathrm{a} - \rho \boldsymbol{g} \cdot \boldsymbol{x}_f - 2\eta \left(\bar{\nabla} \cdot \boldsymbol{u}_f \right) - \sigma \frac{1}{R'_f}.$$

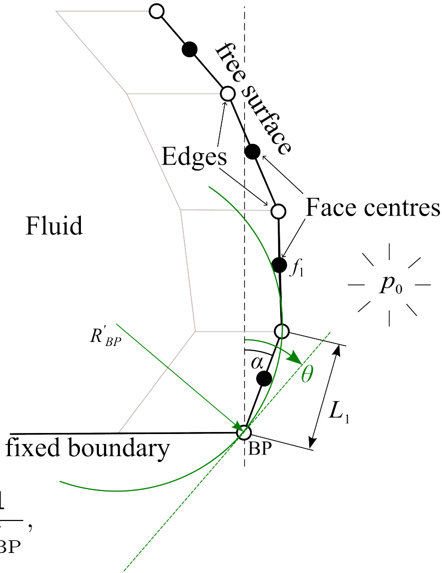
With p_a being some pressure level



Contact angle boundary condition

- Idea from Young-Laplace equation, where gauge pressure defines contact angle boundary condition
- Curvature at the boundary point BP can be expressed through a circle
- tangent at BP makes the angle θ with the vertical and the circle goes through both vertices of the boundary face

$$\frac{1}{R'_{\mathrm{BP}}} = \frac{2\sin\left(\theta - \alpha_1\right)}{L_1}$$



$$p_{\mathrm{BP}} = p_0 - \rho \boldsymbol{g} \cdot \boldsymbol{x}_{\mathrm{BP}} - 2\eta\left(\bar{\nabla} \cdot \boldsymbol{u}_{\mathrm{BP}}\right) - \sigma \frac{1}{R'_{\mathrm{BP}}},$$

Implementation of new BC

- Create a new BC `freeSurfaceContactAngle` condition by copying `fvPatchFields/freeSurfacePressure` and changing the file names and BC name
- Add member data for the patch and the contact angle
- Add calculation for gauge pressure

Implementation of new BC

Additional member data:

freeSurfaceContactAnglePressureFvPatchScalarField.H

```
class freeSurfaceContactAnglePressureFvPatchScalarField
:
    public fixedValueFvPatchScalarField
{
protected:

    // Protected data

        //- Ambient pressure
        scalarField pa_;

        //- Name of fa boundary for contact angle condition
        word contactAnglePatch_;

        //- Desired contact angle
        scalar contactAngle_;
```

updateCoeffs() in freeSurfaceContactAnglePressureFvPatchScalarField.C

```
void Foam::freeSurfaceContactAnglePressureFvPatchScalarField::updateCoeffs()
{
```

```
    myInterfaceTrackingFvMesh& itm =
        refCast<myInterfaceTrackingFvMesh>
        (
            const_cast<dynamicFvMesh&>
            (
                mesh.lookupObject<dynamicFvMesh>("fvSolution")
            )
        );
```

```
const label& patchI = itm.aMesh().boundary().findPatchID(contactAnglePatch_);
```

updateCoeffs() in freeSurfaceContactAnglePressureFvPatchScalarField.C

```
    const labelList peFaces =
                labelList::subList
                (
                    itm.aMesh().edgeOwner(),
                    itm.aMesh().boundary()[patchI].faPatch::size(),
                    itm.aMesh().boundary()[patchI].start()
                );

    const scalar currentAngleDiff(90-contactAngle_-itm.faceAngles()[peFaces[0]])
     ;
```

```
    const pointField& points = itm.aMesh().patch().localPoints();
    const edgeList& edges = itm.aMesh().patch().edges();
    const labelList& pEdges = itm.aMesh().boundary()[patchI]; // the one edge
    vectorField peCentres(pEdges.size(), Zero);
    forAll(peCentres, edgeI)
    {
        peCentres[edgeI] =
            edges[pEdges[edgeI]].centre(points);
    }
```

$$\frac{1}{R'_{\mathrm{BP}}} = \frac{2\sin\left(\theta - \alpha_1\right)}{L_1}$$

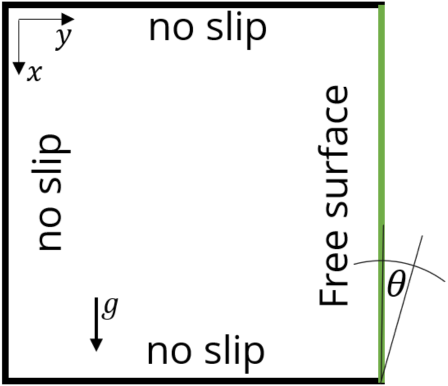
$$p_{\mathrm{BP}} = p_0 - \rho \boldsymbol{g} \cdot \boldsymbol{x}_{\mathrm{BP}} - 2\eta\left(\bar{\nabla} \cdot \boldsymbol{u}_{\mathrm{BP}}\right) - \sigma\frac{1}{R'_{\mathrm{BP}}},$$

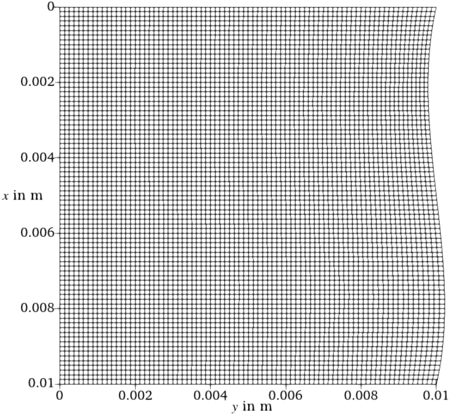
`updateCoeffs()` in `freeSurfaceContactAnglePressureFvPatchScalarField.C`

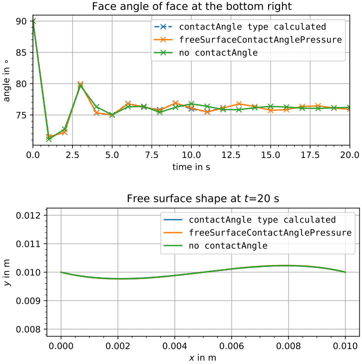
```
    scalarField pressureForce(pressureJump.size(),
                2/Le[0]*sin(currentAngleDiff*constant::mathematical::pi/180));

    if (itm.pureFreeSurface())
    {
        pressureJump -= itm.sigma().value()*pressureForce;
    }
    else
    {
        pressureJump -= itm.surfaceTension().internalField()*pressureForce;
    }
```

New case setup

New case setup to resemble FZ-process with silicon fluid
($\rho = 2580 \frac{\text{kg}}{\text{m}^3}$, $\nu = 3.333 \cdot 10^{-7} \frac{\text{m}^2}{\text{s}}$, $\sigma = 0.88 \frac{\text{N}}{\text{m}}$, $\theta = 11°$)







`contactAngleColumn` case with a desired contact angle of 79° and for the three boundary condition types:

1. `type calculated` boundary condition for the `contactAngle`, i.e., with the default code
2. `freeSurfaceContactAnglePressure` boundary condition for the pressure
3. no `contactAngle` file specified.

Conclusion

- Implementation of interface tracking as a library in OpenFOAM is very flexible and includes surface tension gradients due to concentration changes
- Free surface mesh motion enables a sharp interface for external pressure
- Implementation of contact angle boundary condition in `interfaceTrackingFvMesh` is done via the `controlPoints` and not the actual surface shape
- There is no convergence check for the net mass flux

- Surface data can be written to `vtk` files together with the mesh
- `faceAngles` are calculated and can be used by the boundary conditions
- Boundary condition with gauge pressure does not work as intended and the theory has to be re-evaluated