Cite as: Jansson, M.: Implementing a Zwart-Gerber-Belamri cavitation model. In Proceedings of CFD with OpenSource Software, 2018, Edited by Nilsson. H., `http://dx.doi.org/10.17196/OS_CFD#YEAR_2018`

CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Implementing a Zwart-Gerber-Belamri cavitation model

Developed for OpenFOAM-1806 and gnuplot 5.2

Author:
Marcus JANSSON
Linköping University
marcus.jansson@liu.se

Peer reviewed by:
Ebrahim GHAHRAMANI
Sandip WADEKAR

December 20, 2018

Learning outcomes

The reader will learn:

How to use it:

- How to adapt and run the throttle tutorial.

The theory of it:

- The theory of Schnerr-Sauer (SS) and Zwart-Gerber-Belamri (ZGB) cavitation models.

How it is implemented:

- How to implement the ZGB cavitation model in the phaseChangeTwoPhaseMixtures class.

How to modify it:

- How to modify the mass transfer rates.

Prerequisites

The reader is expected to know the following in order to get maximum benefit out of this report:

- Fundamentals of fluid mechanics and bubble dynamics e.g. book by Brennen [1] or Franc and Michel [2].
- Some understanding of cavitation modelling, e.g. Schnerr and Sauer [3] and Zwart et. al. [4]
- Run standard document tutorials like throttle tutorial and/or cavitatingBullet tutorial.
- Basic post-processing in paraFoam have been used to visualize the results. However, paraFoam is not in the scope of this tutorial.

Contents

Chapter 1

Introduction

This tutorial describes how to add the Zwart-Gerber-Belamri (ZGB) cavitation model to the phaseChangeTwoPhaseMixtures class used by interPhaseChangeFoam solver. Both the ZGB model and the default Schnerr-Sauer (SS) model are described in this chapter. The solver is described briefly in Chapter 2, followed by a simple test case with the default SS cavitation model. In Section 2.2, the ZGB cavitation model is implemented and the same case is solved with this additional model.

1.1 Bubble dynamics and cavitation models

The dynamics of a single bubble subjected to a pressure field can be described by

$$\frac{p_b - p_\infty}{\rho_l} = R\frac{d^2R}{dt^2} + \frac{3}{2}\left(\frac{dR}{dt}\right)^2 + \frac{4\nu_l}{R}\frac{dR}{dt} + \frac{2S}{\rho_l R} \tag{1.1}$$

which is known as the Rayleigh-Plesset equation [1]. $R = R(t)$ is the bubble radius, S is the surface tension coefficient, p_b is the bubble pressure and p_∞ is the far-field pressure. ρ_l and ν_l is the liquid density and liquid kinematic viscosity respectively. The Rayleigh-Plesset equation assumes that bubbles are spherical and symmetric and that thermal effects are negligible. By further neglecting higher order terms and the effects of surface tension and viscosity, eq. (1.1) can be simplified to

$$\frac{dR}{dt} = \sqrt{\frac{2}{3}\frac{p_b - p}{\rho_l}} \tag{1.2}$$

Due to evaporation and condensation, mass transfer occur between the liquid- and vapor phase. This mass transfer is governed by a transport equation for the liquid fraction

$$\frac{\partial}{\partial t}\left(\alpha_l \rho_l\right) + \nabla \cdot \left(\alpha_l \rho_l \vec{v}\right) = \dot{m} \tag{1.3}$$

where ρ_l is the density of the liquid phase and $\dot{m}$ is the mass transfer rate. In interPhaseChangeFoam, which later will be described, the mass transfer is decomposed to vaporization- and condensation terms

$$\dot{m} = \alpha_l \dot{m}_\alpha^- + \left(1 - \alpha_l\right)\dot{m}_\alpha^+ = \alpha_l\left(\dot{m}_\alpha^- - \dot{m}_\alpha^+\right) + \dot{m}_\alpha^+ \tag{1.4}$$

where $\dot{m}_\alpha^-$ and $\dot{m}_\alpha^+$ are the destruction of liquid from evaporation and creation of liquid from condensation, respectively. α_l is the liquid volume fraction.

Finally, eq. (1.3) is implemented in terms of volume fraction and decomposed volume transfer rates, $\dot{V}^-$ and $\dot{V}^+$,

$$\frac{\partial \alpha_l}{\partial t} + \nabla \cdot (\alpha_l \vec{v}) = \left(\nabla \cdot \vec{v} + \dot{V}^- - \dot{V}^+ \right) \alpha_l + \dot{V}^+ \tag{1.5}$$

where the volume transfer rate, $\dot{V}$, and velocity divergence, $\nabla \cdot \vec{v}$,

$$\dot{V} = \left(\frac{1}{\rho_l} - \alpha_l \left(\frac{1}{\rho_l} - \frac{1}{\rho_v} \right) \right) \dot{m} \tag{1.6}$$

$$\nabla \cdot \vec{v} = \left(\frac{1}{\rho_l} - \frac{1}{\rho_v} \right) \dot{m} \tag{1.7}$$

have been used. The divergence of the velocity field, eq. (1.7), also appears on the RHS of the pressure correction equation. In this case, the mass transfer can be decomposed to

$$\dot{m}_p = \frac{\dot{m}}{(p - p_v)} = \dot{m}_p^- - m_p^+ \tag{1.8}$$

where p_v is a threshold pressure (the saturation pressure for the liquid). For a full description of the discretized equations and the decomposition of mass transfer rates, refer to Asnaghi [5].

1.1.1 Schnerr-Sauer cavitation model

Schnerr and Sauer [3] derived an expression for the net mass change rate from the generalized vapor transport equation. This model is implemented in OpenFoam through the mass source terms $\dot{m}_\alpha^-$ and $\dot{m}_\alpha^+$

$$\dot{m}_\alpha^- = C_v \left(1 + \alpha_{nuc} - \alpha_l\right) \frac{3\rho_v \rho_l}{\rho_m R_B} \sqrt{\frac{2}{3\rho_l} \frac{1}{|p - p_v|}} min\left(p - p_v, 0\right) \tag{1.9}$$

$$\dot{m}_\alpha^+ = C_c \alpha_l \frac{3\rho_v \rho_l}{\rho_m R_B} \sqrt{\frac{2}{3\rho_l} \frac{1}{|p - p_v|}} max\left(p - p_v, 0\right) \tag{1.10}$$

which follow the decomposition in eq. (1.4). ρ_v is the vapor density. ρ_m is the mixture density, a volume weighted average of the liquid and vapor density. α_{nuc} is a numerical nucleation term that is introduced to initialize cavitation. Without the presence of this term, eq. (1.9) equals zero for all pressures in a pure liquid and cavitation would never occur. The relation between nucleation site volume fraction, α_{nuc}, and bubble radius, R_B, is described according to

$$\alpha_{nuc} = \frac{\frac{1}{6} n_b \pi d_{nuc}^3}{1 + \frac{1}{6} n_b \pi d_{nuc}^3} \tag{1.11}$$

$$R_B = \left(\frac{3}{4\pi n_b} \frac{1 + \alpha_{nuc} - \alpha_l}{\alpha_l} \right)^{\frac{1}{3}} \tag{1.12}$$

where n_b is the bubble number density and d_{nuc} is the nucleation site diameter. It can be seen that the mass transfer rates for evaporation and condensation are differentiated by the coefficients C_v and C_c, respectively. The default value is however 1 for both coefficients.

The phase change terms in the pressure correction equation are defined similarly, but according to the decomposition in eq. (1.8),

$$\dot{m}_p^- = -C_v \alpha_l \left(1 + \alpha_{nuc} - \alpha_l\right) \frac{3\rho_v \rho_l}{\rho_m R_B} \sqrt{\frac{2}{3\rho_l} \frac{1}{|p - p_v|}} neg\left(p - p_v\right) \tag{1.13}$$

$$\dot{m}_p^+ = C_c \alpha_l \left(1 - \alpha_l\right) \frac{3\rho_v \rho_l}{\rho_m R_B} \sqrt{\frac{2}{3\rho_l} \frac{1}{|p - p_v|}} pos\left(p - p_v\right) \tag{1.14}$$

where the neg function returns -1 when the input is negative and zero otherwise, and the pos function returns 1 when the input is positive and zero otherwise.

1.1.2 Zwart-Gerber-Belamri cavitation model

In the ZGB cavitation model, the mass transfer rates are given as

$$\dot{m}_\alpha^- = -C_v r_{nuc} \frac{3\rho_v}{R_B} \sqrt{\frac{2}{3\rho_l} \frac{1}{|p - p_v|}} min\left(p - p_v, 0\right) \tag{1.15}$$

$$\dot{m}_\alpha^+ = C_c \frac{3\rho_v}{R_B} \sqrt{\frac{2}{3\rho_l} \frac{1}{|p - p_v|}} max\left(p - p_v, 0\right) \tag{1.16}$$

where r_{nuc} is the nucleation site volume fraction. Like the SS model, it is based on the simplified Rayleigh-Plesset equation but assumes a constant nucleation site volume. Thus, both the nucleation site volume fraction, r_{nuc}, and nucleation site radius, R_B, are model constants. According to Zwart et. al. [4], these constants are $r_{nuc} = 5.0 * 10^{-4}$ and $R_B = 1.0 * 10^{-6}$ m by default. Following the same decomposition as in the SS model, the phase change terms in the pressure correction equation are

$$\dot{m}_p^- = -C_v r_{nuc} \alpha_l \frac{3\rho_v}{R_B} \sqrt{\frac{2}{3} \frac{|p - p_v|}{\rho_l}} neg\left(p - p_v\right) \tag{1.17}$$

$$\dot{m}_p^+ = C_c \left(1 - \alpha_l\right) \frac{3\rho_v}{R_B} \sqrt{\frac{2}{3} \frac{|p - p_v|}{\rho_l}} pos\left(p - p_v\right) \tag{1.18}$$

Chapter 2

interPhaseChangeFoam solver

The `interPhaseChangeFoam` solver is a fully transient solver. Turbulence is generic, i.e. it can be solved using either laminar, RAS or LES. It is based on the PIMPLE pressure correction. It solves for two isothermal, incompressible fluids using a Volume of Fluids (VoF) approach where momentum equations are solved for the mixture. `interPhaseChangeFoam` supports mass transfer through cavitation and the available models are Merkle, Kunz, and Schnerr-Sauer. The Schnerr-Sauer model is implemented as described in Section 1.1.1. Note that `interPhaseChangeFoam` solves for the liquid fraction, which will affect how the mass transfer rate is implemented.

```
interPhaseChangeFoam
├── Allwclean
├── Allwmake
├── alphaControls.H
├── alphaEqn.H
├── alphaEqnSubCycle.H
├── createFields.H
├── interPhaseChangeDyMFoam/
├── interPhaseChangeFoam.C
├── Make/
├── pEqn.H
├── phaseChangeTwoPhaseMixtures/
│   ├── Kunz/
│   ├── lnInclude/
│   ├── Make/
│   ├── Merkle/
│   ├── phaseChangeTwoPhaseMixture/
│   └── SchnerrSauer/
└── UEqn.H
```

The `interPhaseChangeFoam` directory contains both `Allwmake` and `Allwclean`. You can also find a solver for dynamic meshing, `interPhaseChangeDyMFoam`, which will not be considered in this tutorial. There are equations for p, U and alpha, which will be described later. One interesting thing is that there is a class named `phaseChangeTwoPhaseMixtures` with its own `Make` directory. This class is located in the solver, which is not the more common approach in OpenFOAM. We will come back to both this class and the equations, but first take a look in `interPhaseChangeFoam.C`.

```
    interPhaseChangeFoam.C
    ...
    Info<< "\nStarting time loop\n" << endl;

    while (runTime.run())
    {
        #include "readTimeControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"

        runTime++;

        Info<< "Time = " << runTime.timeName() << nl << endl;

        // --- Pressure-velocity PIMPLE corrector loop
        while (pimple.loop())
        {
            #include "alphaControls.H"

            surfaceScalarField rhoPhi
            (
                IOobject
                (
                    "rhoPhi",
                    runTime.timeName(),
                    mesh
                ),
                mesh,
                dimensionedScalar(dimMass/dimTime, Zero)
            );

            mixture->correct();

            #include "alphaEqnSubCycle.H"
            interface.correct();

            #include "UEqn.H"

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }

            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }

        runTime.write();

        runTime.printExecutionTime(Info);
    }

    Info<< "End\n" << endl;

    return 0;
}
```

As mentioned, interPhaseChangeFoam is a PIMPLE solver. In the outer loop, before the pressure correction, the interfaces are corrected by calling `alphaEqnSubCycle.H`. and solving the (liquid) phase transport equation. The phase equation is found in `alphaEqn.H`. On the LHS we have, in order, the transient term $\frac{\partial \alpha}{\partial t}$ solved with first order Euler Scheme, the convective term $\nabla \cdot (\alpha \vec{v})$ solved with upwind flux, and the convective source term $\nabla \cdot (\alpha_l \vec{v})$ (on the RHS in eq. (1.5)). The other source terms can be seen on the right hand side according to eq. (1.5). As indicated by the names of the functions, `vDotvAlphal` and `vDotcAlphal` are volume source terms.

```
alphaEqn.H
...
Pair<tmp<volScalarField>> vDotAlphal =
    mixture->vDotAlphal();
const volScalarField& vDotcAlphal = vDotAlphal[0]();
const volScalarField& vDotvAlphal = vDotAlphal[1]();
const volScalarField vDotvmcAlphal(vDotvAlphal - vDotcAlphal);
...
fvScalarMatrix alpha1Eqn
(
    fv::EulerDdtScheme<scalar>(mesh).fvmDdt(alpha1)
  + fv::gaussConvectionScheme<scalar>
    (
        mesh,
        phi,
        upwind<scalar>(mesh, phi)
    ).fvmDiv(phi, alpha1)
  - fvm::Sp(divU, alpha1)
 ==
    fvm::Sp(vDotvmcAlphal, alpha1)
  + vDotcAlphal
);
```

The volume source terms `vDotvAlphal` (vaporization) and `vDotcAlphal` (condensation) are components of the pair called `vDotAlphal`. This is found in the `phaseChangeTwoPhaseMixtures` class, which is a part of the `interPhaseChageFoam` solver.

```
phaseChangeTwoPhaseMixture.C
...
Foam::Pair<Foam::tmp<Foam::volScalarField>>
Foam::phaseChangeTwoPhaseMixture::vDotAlphal() const
{
    volScalarField alphalCoeff(1.0/rho1() - alpha1_*(1.0/rho1() - 1.0/rho2()));
    Pair<tmp<volScalarField>> mDotAlphal = this->mDotAlphal();

    return Pair<tmp<volScalarField>>
    (
        alphalCoeff*mDotAlphal[0],
        alphalCoeff*mDotAlphal[1]
    );
}

Foam::Pair<Foam::tmp<Foam::volScalarField>>
Foam::phaseChangeTwoPhaseMixture::vDotP() const
{
    dimensionedScalar pCoeff(1.0/rho1() - 1.0/rho2());
    Pair<tmp<volScalarField>> mDotP = this->mDotP();

    return Pair<tmp<volScalarField>>(pCoeff*mDotP[0], pCoeff*mDotP[1]);
}
```

The volume source terms `vDotAlphal` are related to the mass source terms `mDotAlphal` with the coefficient `alphaCoeff`, corresponding to eq. (1.6). The mass source terms are also part of the `phaseChangeTwoPhaseMixtures` class, but specific for each individual cavitation model e.g. Schnerr-Sauer which is found in `SchnerrSauer.C`.

```
SchnerrSauerr.C
...
Foam::tmp<Foam::volScalarField>
Foam::phaseChangeTwoPhaseMixtures::SchnerrSauer::pCoeff
(
 const volScalarField& p
) const
{
volScalarField limitedAlpha1(min(max(alpha1_, scalar(0)), scalar(1)));
volScalarField rho
(
    limitedAlpha1*rho1() + (scalar(1) - limitedAlpha1)*rho2()
);

return
    (3*rho1()*rho2())*sqrt(2/(3*rho1()))
   *rRb(limitedAlpha1)/(rho*sqrt(mag(p - pSat()) + 0.01*pSat()));
}

Foam::Pair<Foam::tmp<Foam::volScalarField>>
Foam::phaseChangeTwoPhaseMixtures::SchnerrSauer::mDotAlphal() const
{
    const volScalarField& p = alpha1_.db().lookupObject<volScalarField>("p");
    volScalarField pCoeff(this->pCoeff(p));

    volScalarField limitedAlpha1(min(max(alpha1_, scalar(0)), scalar(1)));

    return Pair<tmp<volScalarField>>
    (
        Cc_*limitedAlpha1*pCoeff*max(p - pSat(), p0_),

        Cv_*(1.0 + alphaNuc() - limitedAlpha1)*pCoeff*min(p - pSat(), p0_)
    );
}
```

`SchnerrSauer.C` returns a pair `mDotAplhal` with the mass transfer rates. The member functions `pCoeff` and `mDotAlphal` together make the mass source terms seen in eq. (1.9)-(1.10). `limitedAlpha1` is just a bounded liquid fraction, since this value should be between 0 and 1. `0.01*pSat()` is a small addition that prevents zero in the denominator. `rRb` is the reversed bubble radius $\frac{1}{R_B}$ according to eq. (1.12). In `mDotAlphal` we can see `alphaNuc`, mentioned earlier in eq. (1.11), which prevents zero value in the denominator. The volume fraction of nucleation sites, `alphaNuc`, is a function of the bubble number density, `n`, and the initial bubble diameter, `dNuc`, which are both model constants. `rRb` and `alphaNuc` are also specified in `SchnerrSauer.C`, but are excluded here for brevity.

2.1 throttle tutorial

This section describe how to pre-process, run and post-process a case involving incompressible flow through a cavitating jet. The throttle geometry consists of two volumes connected through a narrow section, shown in Figure 2.1. At the left side is a pressure inlet, and the right side a pressure outlet. If the pressure difference between inlet and outlet is large enough, the change in dynamic pressure in the narrow section will induce cavitation.

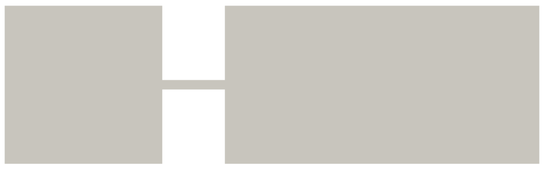


Figure 2.1: Geometry of the throttle tutorial case.

The following steps describe how to copy the throttle tutorial to the run directory and how to modify it to solve with `interPhaseChangeFoam`. Start by copying and renaming the tutorial files. Clean up some files that are not used.

```
cp -r $FOAM_TUTORIALS/multiphase/cavitatingFoam/RAS/throttle \
$FOAM_RUN/throttleSchnerr
cd $FOAM_RUN/throttleSchnerr
rm All*
rm system/topo*
rm system/refineMeshDict
```

To run the tutorial with `interPhaseChangeFoam`, some changes are needed in initial- and boundary conditions and in the `system` directory.

2.1.1 Boundary and initial conditions

Boundary and initial conditions are found in the `\0` directory where the following variables are found

```
alpha.vapour                         p
k                                    rho
nut                                  U
omega
```

The `interPhaseChangeFoam` solver uses the dynamic pressure and the liquid fraction. It can be seen from the boundary conditions that `cavitatingFoam`, which is the default solver for this tutorial, solves for the vapor volume fraction instead. Rename the initial conditions for the pressure and replace the initial conditions for the phase fraction. Reduce the outlet pressure to make sure that cavitation occurs.

```
mv 0/p 0/p_rgh
sed -i s/"uniform 100e5;"/"uniform 50e5;"/g 0/p_rgh
rm 0/alpha.vapour
cp -r $FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet/0.orig/alpha.water \
0/alpha.water
sed -i s/"inletOutlet"/"fixedValue"/g 0/alpha.water
sed -i s/"inletValue"/"value"/g 0/alpha.water
sed -i s/"bullet"/"frontBack"/g 0/alpha.water
sed -i s/"zeroGradient"/"empty"/g 0/alpha.water
sed -i s/"symmetry"/"zeroGradient"/g 0/alpha.water
```

2.1.2 Transport properties

Replace `/constant/transportProperties` and remove `/constant/thermodynamicProperties` (since `interPhaseChangeFoam` solves for isothermal fluids). Add the gravity `g` which is used to subtract the hydrostatic pressure. Leave the default k-ω SST turbulence model.

```
rm constant/thermodynamicProperties
rm constant/transportProperties
cp -r $FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet\
/constant/transportProperties constant/.
cp -r $FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet/constant/g \
constant/.
```

2.1.3 Discretization schemes and solution control

As mentioned, `interPhaseChangeFoam` solves for the liquid fraction and not the vapor fraction. Solver settings for the liquid fraction, and dynamic pressure, are found in `system/fvSolution` and `system/fvSchemes`. Change `solvers` in `system/fvSolution` to

```
solvers
{
    "alpha.water.*"
    {
        cAlpha          0;
        nAlphaCorr      2;
        nAlphaSubCycles 1;

        MULESCorr       yes;
        nLimiterIter    5;

        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-8;
        relTol          0;
        maxIter         10;
    };

    "U.*"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-6;
        relTol          0;
    };

    p_rgh
    {
        solver          GAMG;
        tolerance       1e-8;
        relTol          0.1;

        smoother        DICGaussSeidel;



        maxIter         50;
    };

    p_rghFinal
    {
        solver          PCG;
        preconditioner
        {
            preconditioner  GAMG;

            tolerance       1e-6;
            relTol          0;

            nVcycles        2;

            smoother        DICGaussSeidel;

        };
        tolerance       1e-7;
        relTol          0;
        maxIter         50;
    };

    "pcorr.*"
    {
        $p_rgh;
        relTol          0;
    };

    Phi
    {
        $p_rgh;
        relTol          0;
    };

    "(U|k|omega)"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0.1;
    }

    "(U|k|omega)Final"
    {
        solver          smoothSolver;
        smoother        symGaussSeidel;
        tolerance       1e-08;
        relTol          0;
    }
}
```

and add after `solvers`

```
potentialFlow
{
    nNonOrthogonalCorrectors   3;
}
```

In `/system/fvSchemes`, add the following `divSchemes`.

```
div(phi,alpha)      Gauss vanLeer;
div(phirb,alpha)    Gauss linear;
```

2.1.4 Run and post-process

Create a `blockMesh` and run `interPhaseChangeFoam`.

```
blockMesh >& log_blockMesh&
interPhaseChangeFoam >& log_run&
```

At the final time step, the velocity shown in Figure 2.2a has got stabilized and the jet takes a symmetric shape. From the water fraction shown in Figure 2.2b, it can be seen that the throttle cavitates.

Add a `singleGraph` file to the `system` directory to extract pressure values along a center line through the domain.

```
touch system/singleGraph
```

Change the content of `singleGraph` to:

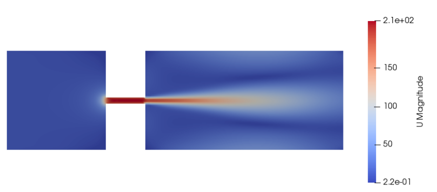
```
singleGraph
{
    start   (0.0 0.0 0.0);
    end     (0.017 0.0 0.0);
    fields  (p_rgh U);

    #includeEtc "caseDicts/postProcessing/graphs/sampleDict.cfg"

    setConfig
    {
        axis    distance;
    }

    // Must be last entry
    #includeEtc "caseDicts/postProcessing/graphs/graph.cfg"
}
```

Run the post processing with the `singleGraph` function.

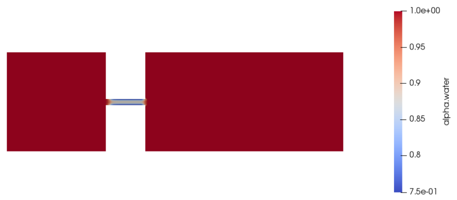
```
postProcess -func singleGraph >& log_postProcess&
```

Plot the pressure using gnuplot. It can be seen in Figure 2.3 that the pressure in the throttle reaches the vapor pressure, which explains the cavitation inception.

```
gnuplot
set style data linespoints
plot "postProcessing/singleGraph/0.0019/line_p_rgh.xy"
```



(a) Velocity



(b) Water volume fraction

Figure 2.2: Last time step for throttle tutorial solved with interPhaseChangeFoam and Schnerr-Sauer cavitation model.

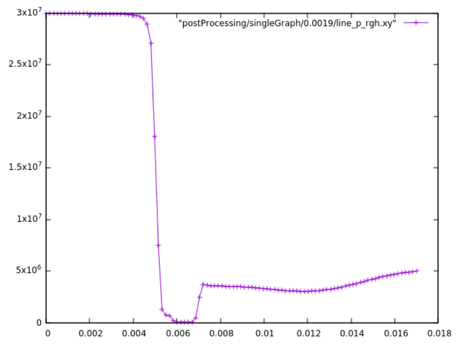


Figure 2.3: Pressure along a centre line.

2.2 ZGB cavitation model

Since the ZGB cavitation model is not available by default, it has to be added to the `phaseChangeTwoPhaseMixtures` class.

2.2.1 Implementation and compilation

Start by copying the existing cavitation models in `phaseChangeTwoPhaseMixtures` to your user directory. Copy the SS cavitation model (which we will use as a starting point) and rename the files.

```
cp -r $FOAM_APP/solvers/multiphase/interPhaseChangeFoam/phaseChangeTwoPhaseMixtures \
$WM_PROJECT_USER_DIR/src/phaseChangeTwoPhaseMixtures
cd $WM_PROJECT_USER_DIR/src/phaseChangeTwoPhaseMixtures
cp -r SchnerrSauer Zwart
cd Zwart
mv SchnerrSauer.C Zwart.C
mv SchnerrSauer.H Zwart.H
```

Replace each 'SchnerrSauer' with 'Zwart' in `Zwart.C` and `Zwart.H`.

```
sed -i s/"SchnerrSauer"/"Zwart"/g Zwart.*
```

Make sure that `Make/files` ends with the following two lines:

```
...
Zwart/Zwart.C

LIB = $(FOAM_USER_LIBBIN)/libphaseChangeTwoPhaseMixtures
```

This ensures that the Zwart model is compiled and the binaries are put in the user directory, to not interfere with the original installation. Try using `wmake` to see that the model compiles. Note that it is still identical to the SS cavitation model. When the compilation is successful, the user library `phaseChangeTwoPhaseMixtures` will be used as default instead of the library in the original installation. You can check which library will be used by `interPhaseChangeFoam`.

```
ldd `which interPhaseChangeFoam` | grep phaseChange
```

It can be seen that the `phaseChangeTwoPhaseMixtures` library is read from the user directory.

```
libphaseChangeTwoPhaseMixtures.so =>/home/marcjans/OpenFOAM/marcjans-v1806/ \
platforms/linux64GccDPInt32Opt/lib/libphaseChangeTwoPhaseMixtures.so \
(0x00007fca9277d000)
```

When comparing eq. (1.9)-(1.10) to eq. (1.15)-(1.16), it can be seen that the differences are minor. The ZGB model only considers the vapor density while the SS model also includes the fraction between liquid- and mixture density. For small vapor fractions, however, this fraction will be close to unity. Furthermore, there is a difference in the contribution of the phase volume fractions. Since the SS model in OpenFOAM uses coefficients for evaporation and condensation, this is similar to the ZGB model. The default values are different though, where ZGB uses $C_v = 50$ and $C_c = 0.01$.

As mentioned, the ZGB model uses constant values for nucleation site volume fraction, r_{nuc}, and nucleation site radius, R_B. Start by removing the private member data for `n_`, `dNuc_`, `rRb_` and `alphaNu_c` in `Zwart.H`. Add the following new member data between `Cv_` and `p0_`.

```
//- Nucleation site volume
dimensionedScalar rNuc_;

//- Nucleation site radius
dimensionedScalar Rb_;
```

Continue with `Zwart.C`. Remove the constructors for `n_` and `dNuc_`. Add in the constructor, between `Cv_` and `p0_`:

```
rNuc_("rNuc", dimless, phaseChangeTwoPhaseMixtureCoeffs_),
Rb_("Rb", dimLength, phaseChangeTwoPhaseMixtureCoeffs_),
```

Remove the entire member functions `rRb` and `alphaNuc`. Replace the member functions `pCoeff` and `mDotAlpha` so the mass transfer rates correspond to eqs. (1.15)-(1.16). Replace `mDotP` according to eqs. (1.17)-(1.18).

```
Foam::tmp<Foam::volScalarField>
Foam::phaseChangeTwoPhaseMixtures::Zwart::pCoeff
(
    const volScalarField& p
) const
{
    return
        (3*rho2())*sqrt(2/(3*rho1()))
        /(Rb_*sqrt(mag(p - pSat()) + 0.01*pSat()));
}
```

```
Foam::Pair<Foam::tmp<Foam::volScalarField> >
Foam::phaseChangeTwoPhaseMixtures::Zwart::mDotAlphal() const
{
    const volScalarField& p = alpha1_.db().lookupObject<volScalarField>("p");
    volScalarField limitedAlpha1(min(max(alpha1_, scalar(0)), scalar(1)));

    volScalarField pCoeff(this->pCoeff(p));

    return Pair<tmp<volScalarField> >
    (
        Cc_*pCoeff*max(p - pSat(), p0_),

        Cv_*rNuc_*pCoeff*min(p - pSat(), p0_)
    );
}

Foam::Pair<Foam::tmp<Foam::volScalarField> >
Foam::phaseChangeTwoPhaseMixtures::Zwart::mDotP() const
{
    const volScalarField& p = alpha1_.db().lookupObject<volScalarField>("p");
    volScalarField pCoeff(this->pCoeff(p));

    volScalarField limitedAlpha1(min(max(alpha1_, scalar(0)), scalar(1)));

    return Pair<tmp<volScalarField> >
    (
        Cc_*(1.0 - limitedAlpha1)*pos(p - pSat())*pCoeff,

        (-Cv_)*rNuc_*limitedAlpha1*neg(p - pSat())*pCoeff
    );
}
```

Finally, remove `n_` and `dNuc_` from the `read` function and add `rNuc_` and `Rb_`. Compile the class again using `wmake`.

```
bool Foam::phaseChangeTwoPhaseMixtures::Zwart::read()
{
    if (phaseChangeTwoPhaseMixture::read())
    {
        phaseChangeTwoPhaseMixtureCoeffs_ = optionalSubDict(type() + "Coeffs");

    phaseChangeTwoPhaseMixtureCoeffs_.lookup("Cc") >> Cc_;
        phaseChangeTwoPhaseMixtureCoeffs_.lookup("Cv") >> Cv_;
        phaseChangeTwoPhaseMixtureCoeffs_.lookup("rNuc") >> rNuc_;
        phaseChangeTwoPhaseMixtureCoeffs_.lookup("Rb") >> Rb_;

        return true;
    }
    else
    {
        return false;
    }
}
```

2.2.2 Pre- and post processing

Go to the run directory and copy the throttle tutorial. Remove the old time directories.

```
run
cp -r throttleSchnerr throttleZwart
cd throttleZwart
rm -r 0.*
rm -r postProcessing
```

In the `constant/transportProperties`, change `phaseChangeTwoPhaseMixture` from `SchnerrSauer` to `Zwart` to use the new cavitation model. Add the new model coefficients after the section with `SchnerrSauerCoeffs`.

```
ZwartCoeffs
{
    Cc      0.01;
    Cv      50;
    rNuc    5.0e-04;
    Rb      1.0e-06;
}
```

Note the additional coefficients `rNuc` and `Rb` and that the values for `Cc` and `Cv` are changed. Solve the case, now with the new ZGB cavitation model.

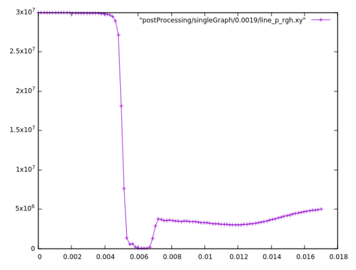
```
interPhaseChangeFoam >& log_run&
```

Run the post processing.

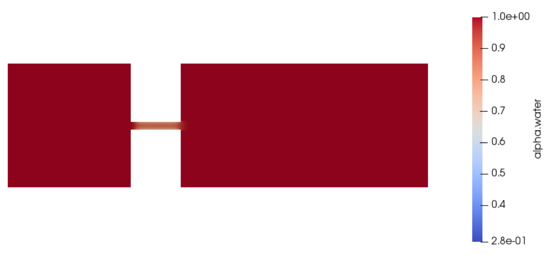
```
postProcess -func singleGraph >& log_postProcess&
```

Plot the pressure using gnuplot. Both the pressure distribution and the vapor fraction shown in Figure 2.4 are similar to the previous results. It should be noted that the models are sensitive to their various parameters and have to be tuned for each specific case. Validation of the models and their results are not in the scope of this tutorial.

```
gnuplot
set style data linespoints
plot "postProcessing/singleGraph/0.0019/line_p_rgh.xy"
```



(a) Pressure



(b) Water fraction

Figure 2.4: Last time step for throttle tutorial solved with interPhaseChangeFoam and Zwart et. al. cavitation model.

Bibliography

[1] C. E. Brennen. *Cavitation and Bubble Dynamics.* Oxford University Press, 1995.

[2] J-P. Franc and J-M. Michel. *Fundamentals of Cavitation.* Kluwer Academic Publishers, 2010.

[3] G. H. Shnerr and J. Sauer. Physical and numerical modeling of unsteady cavitation dynamics. In *Fourth International Conference on Multiphase Flow*, New Orleans, USA, 2001.

[4] P. J. Zwart, A. G. Gerber, and T Belamri. A two-phase flow model for predicting cavitation dynamics. In *Proceedings of the International Conference on Multiphase Flow (ICMF 04)*, Yokohama, Japan, 2004.

[5] Abolfazl Asnaghi. *Developing computational methods for detailed assessment of cavitation on marine propellers.* PhD thesis, Department of Shipping and Marine Technology, Chalmers University of Technology, 2015.

Study questions

1. How do you check which libraries are used by a specific solver, e.g. interPhaseChangeFoam?
2. What are the main differences between the Schnerr-Sauer model and the Zwart-Gerber-Belamri model?
3. What is the role of α_{nuc}?
4. Why do you need to change the phase fraction boundary conditions for the throttle, cf. cavitatingBullet? (disregard the different patch names)