CFD WITH OPENSOURCE SOFTWARE

A COURSE AT CHALMERS UNIVERSITY OF TECHNOLOGY
TAUGHT BY HÅKAN NILSSON

Project work:

Implementation of Transport Model into CavitatingFoam to simulate the Cavitation in Diesel Injector Nozzle

---

Developed for OpenFOAM-2.3.x

Author:
Barış BİÇER

Peer reviewed by:
Erik KRANE
Jelena ANDRIC

Disclaimer: This is a student project work, done as part of a course where OpenFOAM and some other OpenSource software are introduced to the students. Any reader should be aware that it might not be free of errors. Still, it might be useful for someone who would like learn some details similar to the ones presented in the report and in the accompanying files.

December, 2014

Contents

1. **Introduction** **3**

2. **Theoretical Background** **3**

3. **Description of CavitatingFoam** **4**

   3.1 Mathematical Equations…… 4

   3.2 Solver Members…… 6

4. **Description of interPhaseChangeFoam** **12**

   4.1 Mathematical Equations…….……... 12

   4.2 Solver Members……..…….. 13

   4.3 Cavitation Models……..…….……...17

      4.3.1 Kunz.C ...……... 18

5. **Implementation of Transport Equation Model into CavitatingFoam** **20**

   5.1 Implementation Procedure……...……..…… 20

      5.1.1 Modifications in TransportCavitatingFoam.C file …… 20

      5.1.2 Modifications in createField.H file ……... 22

      5.1.3 Copy and modification in phaseChangeTwoPhaseMixture folder…..... 24

      5.1.4 Modifications in Make/files and Make/options of main solver……25

      5.1.5 Modifications in UEqn.H file…… 26

   5.2 Test Case……..…….…..... 26

   5.3 Results……..…….…..... 28

6. **References** **30**

1. INTRODUCTION

This tutorial contains descriptions and explanations of the two cavitation solvers, such as cavitatingFoam and interPhaseChangeFoam included in OpenFOAM-2.3.x, and then elucidates the implementation of the transport equation model into cavitatingFoam, which is called as ”TransportCavitatingFoam”, to simulate the cavitation phenomena inside an injector nozzle.

In order to do this, first, the alpha transport equation is taken from interPhaseChangeFoam solver and then properly implemented into cavitatingFoam mainly by modifying the phase change and pressure equation instead of barotropic compressible model. Additionally, it is given detailed explanation for Kunz cavitation model [1], which is used to represent the cavitation phenomena in this report. Finally, to show the performance and applicability of the new solver, turbulent cavitating flow inside the enlarged rectangular nozzle is simulated, and the calculated results are verified through the experimental data.

2. THEORETICAL BACKGROUND

Cavitating flow in a nozzle of fuel injector for diesel engines has major importance due to its significant role inside the fuel spray atomization, which strongly affects the diesel engine performance and emissions. Hence, many researchers extensively carried out experiments to investigate the strong effects of cavitation in an injector nozzle on the fuel spray atomization. Bergwerk [2] made an early experiment about the cavitating flow in a small nozzle, and showed that cavitation results in large amplitude of disturbance, which leads to enhanced jet atomization. Nurick [3] performed cavitation visualizations in various type of nozzles with different sizes. Hiroyasu *et al*. [4] showed that liquid jet atomization is enhanced when cavitation expands from the inlet to the exit of the fuel nozzle. Chaves et al. [5] performed some measurements in small and cavitating nozzles with high injection pressure. Soteriou and Andrews [6] carried out analysis about the internal cavitating flow structure in a scaled-up nozzle and classified the incipient of cavitation into the three distinct locations, *i.e.*, a separated boundary layer inner region, a main stream flow, and an attached boundary layer inner region. Additionally, not only a visualization but also velocity measurement in a nozzle has been carried out to clarify the flow structure and the promotion mechanism of atomization by Sou et al. [7]. These observations showed that cavitation inception occurs as bubble clouds in a recirculation zone near the inlet of a nozzle, the formation of a long cavitation film

generates the development of cavitation zone almost to the exit (supercavitation), shedding of cavitation clouds accompanied by vortices finally results in a large deformation of the liquid jet.

Most of the experimental studies have been carried out using large scale transparent nozzles, which enable to facilitate visualization of cavitation structure [8, 9]. However, refraction of light at cylindrical side wall of the nozzles, so small size of the nozzles of about 0.1mm in diameter and 1mm in length, operating at very high injection pressure and velocity up to hundreds meters per second in the nozzles, and complex turbulent cavitating flow make the experimental visualizations and measurements with actual nozzle extremely difficult. Therefore, lots of cavitation models have been developed for many years in the literature to model and simulate the formation and development of cavitation inside injector nozzles [10-14].

3. DESCRIPTION of ”cavitatingFoam” solver

In order to model and simulate the cavitation phenomena using the OpenFOAM, there are some available solvers. OpenFOAM version 2.3.x is used. One of available solver is cavitatingFoam, which main codes are located in $FOAM_SOLVERS/multiphase/ cavitatingFoam. There exists also cavitatingDyMFoam solver, which is used to model the cavitation phenomena around the propellers using dynamic mesh property. The solver is described within the main cavitatingFoam.C code as follows:

”Transient cavitation code based on the homogeneous equilibrium model from which the compressibility of the liquid/vapour "mixture" is obtained. Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.”

3.1 Governing Equations

As it is explained above, cavitatingFoam solver is compressible and based on two-phase mixture approach with the barotropic equation of state, which describes the cavitation model based on the relations of the pressure and density as a closure equation, and defined as:

$$\frac{D\rho_m}{Dt} = \Psi \frac{DP}{Dt} \tag{1}$$

where ρ_m, t, and P denote the mixture density, the time and the pressure, respectively. The parameter Ψ refers to the compressibility of the mixture and corresponds to the inverse of the speed a of sound squared, i.e.

$$\Psi = \frac{1}{a^2} \tag{2}$$

This equation can be inserted directly in the continuity equation to formulate a pressure equation or integrated to obtain the pressure as a function of the density. The mass fraction of vapor in the fluid mixture is denoted by γ, which is calculated using the following relation:

$$\gamma = \frac{\rho_m - \rho_{l,sat}}{\rho_{v,sat} - \rho_{l,sat}} \tag{3}$$

where ρ_m, $\rho_{l,sat}$ and $\rho_{v,sat}$ are density of the mixture, liquid and vapor densities at saturation pressure, respectively. There is no cavitation if γ=0, whereas a cell is fully occupied by cavitation when γ=1. The mixture density ρ_m is calculated by taking into account the ratio of the vapor in the fluid. The mixture's equilibrium equation of state reads:

$$\rho_m = (1-\gamma)\rho_l^0 + (\gamma\Psi_v + (1-\gamma)\Psi_l)P_{sat} + \Psi_m(P - P_{sat}) \tag{4}$$

where, subscripts *l* and *v* stands for the liquid and vapor, respectively, and the subscript *m* stands for the mixture. The mixture compressibility Ψ_m can be modeled in several ways. Wallis linear model [15] used in the OpenFoam-2.3.x, based on vapor mass fraction, and given as follows:

$$\Psi_m = \gamma\Psi_v + (1-\gamma)\Psi_l \tag{5}$$

After determination of the vapor mass fraction, the local mixture viscosity μ_m can be computed as

$$\mu_m = \gamma\mu_v + (1-\gamma)\mu_l \tag{6}$$

The methodology of the cavitatingFoam solver starts by solving the continuity equation for ρ_m:

$$\frac{\partial\rho_m}{\partial t} + \nabla\bullet(\rho_m U) = 0 \tag{7}$$

where *U* denotes the mixture velocity. The calculated value of ρ_m is used to attain preliminary values for γ (in Eq. (3)) and Ψ (in Eq. (5)), and when solving the momentum equations:

$$\frac{\partial\rho_m U}{\partial t} + \nabla\bullet(\rho_m UU) = -\nabla P + \nabla\bullet\left[(\mu_{eff}(\nabla U + (\nabla U)^T)\right] \tag{8}$$

where μ_{eff} is the effective viscosity given by

$$\mu_{eff} = \mu + \mu_t \tag{9}$$

where μ_t represents the turbulence viscosity, which is modeled by one of the RANS turbulence models. In this test case, the RNG k-ε model is used. An iterative PIMPLE algorithm is employed to solve *P* and correct the velocity *U* to achieve continuity. Further details about the equations of the cavitatingFoam can be found in the previous studies [16, 17].

3.2 Solver members

The cavitatingFoam solver in the version of OpenFOAM-2.3.x consists of the following files/folders.

- alphavPsi.H
- cavitatingFoam.C
- continuityErrs.H
- CourantNo.H
- createFields.H
- pEqn.H
- readControls.H
- readThermodynamicsProperties.H
- rhoEqn.H
- setDeltaT.H
- setInitialDeltaT.H
- UEqn.H
- Make
    - files
    - options

The original source code of cavitatingFoam.C looks:

```
// ******************************cavitatingFoam.C********************************** //

#include "fvCFD.H"
#include "barotropicCompressibilityModel.H"
#include "incompressibleTwoPhaseMixture.H"
#include "turbulenceModel.H"
#include "pimpleControl.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

```
* * //
int main(int argc, char *argv[])
{
    #include “setRootCase.H”
    #include “createTime.H”
    #include “createMesh.H”
    #include “readThermodynamicProperties.H”
    #include “readControls.H”
    #include "createFields.H"
    #include "initContinuityErrs.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"

    pimpleControl pimple(mesh);

    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

    Info<< "\nStarting time loop\n"<< endl;

    while (runTime.run())
    {
        #include "readControls.H"
        #include "CourantNo.H"
        #include "setDeltaT.H"
        runTime++;
        Info<< "Time = " << runTime.timeName() << nl << endl;

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

            // --- Pressure corrector loop
            while (pimple.correct())
            {
                #include "pEqn.H"
            }
            if (pimple.turbCorr())
            {
                turbulence->correct();
            }
        }
        runTime.write();
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
```

```
    }
    Info<< "End\n" << endl;
    return 0;
}
// ************************************************************************* //
```

Before the time iteration starts, the solver reads the barotropic compressibility model, and thermodynamic properties, which are set under "$FOAM_TUTORIALS/multiphase/cavitatingFoam/constant/thermodynamicProperties/" as:

```
FoamFile
{
version     2.0;
format      ascii;
class       dictionary;
location    "constant";
object      thermodynamicProperties;
}
// * * * ** * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
barotropicCompressibilityModel linear;
psiv            psiv [ 0 -2 2 0 0 ] 5.6-06;
rholSat         rholSat [ 1 -3 0 0 0 ] 1000;
psil            psil [ 0 -2 2 0 0 ] 4.54e-07;
pSat            pSat [ 1 -1 -2 0 0 ] 2300;
rhoMin          rhoMin [ 1 -3 0 0 0 ] 0.001;
// ************************************************************************* //
```

The psiv and psil correspond to vapor and liquid compressibility, whereas rholSat and pSat show the liquid density at saturation and vapor saturation pressure, respectively. rhoMin represents min density, which is used to keep the density positive and can be set as 0.001.

When the time iteration loop starts, the solver first calculates the max. acoustics and normal Courant numbers, and then the new time step is adjusted according to the courant number value.

In the alphavPsi.H file, the calculation of the mass fraction of the vapor is carried out according to Eqn. (3), and then alpha liquid is calculated as shown below in code.

```
//*********************************alphavPsi.H*********************************** //

    alphav =
        max
        (
            min
            (
                (rho - rholSat)/(rhovSat - rholSat),
                scalar(1)
            ),
            scalar(0)
        );
    alphal = 1.0 - alphav;
    Info<< "max-min alphav: " << max(alphav).value()
        << " " << min(alphav).value() << endl;
    psiModel->correct();
    //Info<< "min a: " << 1.0/sqrt(max(psi)).value() << endl;
}
```

In the rhoEqn.H file, the continuity equation shown in Eqn. (7) is solved to compute the mixture density.

```
// ************************************rhoEqn.H************************************ //

    fvScalarMatrix rhoEqn
    (
        fvm::ddt(rho)
      + fvm::div(phi, rho)
    );
    rhoEqn.solve();
    rhoPhi = rhoEqn.flux();
    Info<< "max-min rho: " << max(rho).value()
        << " " << min(rho).value() << endl;
    rho == max(rho, rhoMin);
}
```

In the UEqn.H file, the momentum equation shown in Eqn. (8) is discretized, and solved to calculate an intermediate velocity field. First part is the LHS of the momentum equation with the viscosity equation from RHS, and the second part solves the LHS of the momentum equation to be equal to the discretized part of the pressure gradient.

```
// ************************************UEqn.H************************************** //
```

```
fvVectorMatrix UEqn
    (
        fvm::ddt(rho, U)
      + fvm::div(rhoPhi, U)
      + turbulence->divDevRhoReff(rho, U)
    );
    UEqn.relax();
    if (pimple.momentumPredictor())
    {
        solve(UEqn == -fvc::grad(p));
    }
    Info<< "max(U) " << max(mag(U)).value() << endl;
```

In the pEqn.H file, after calculation of the mass fluxes at the cells faces, the pressure equation is solved. After that, the mass fluxes at the cell faces are corrected by the determined number of the inner PIMPLE loops. After predicting the first pressure, the final mixture density is calculated according to Eqn. (5). After that, this mixture density is used to calculated pressure before momentum correction. Then, after performing the momentum corrector step on the basis of the new pressure field, the continuity error is calculated. This solving of the momentum equation will be repeated until reaching to the prescribed number of the outer corrector loop.

```
// ************************************pEqn.H************************************ //
{
   if (pimple.nCorrPIMPLE() == 1)
    {
        p =
        (
            rho
          - alphal*rhol0
          - ((alphav*psiv + alphal*psil) - psi)*pSat
        )/psi;
    }
    surfaceScalarField rhof("rhof", fvc::interpolate(rho));
    volScalarField rAU(1.0/UEqn.A());
    surfaceScalarField rhorAUf("rhorAUf", fvc::interpolate(rho*rAU));
    volVectorField HbyA("HbyA", U);
    HbyA = rAU*UEqn.H();
    phi = (fvc::interpolate(HbyA) & mesh.Sf())
         + rhorAUf*fvc::ddtCorr(U, phi);
    surfaceScalarField phiGradp(rhorAUf*mesh.magSf()*fvc::snGrad(p));
    phi -= phiGradp/rhof;
    while (pimple.correctNonOrthogonal())
    {
```

```
        fvScalarMatrix pEqn
        (
            fvm::ddt(psi, p)
          - (rhol0 + (psil - psiv)*pSat)*fvc::ddt(alphav) -
pSat*fvc::ddt(psi)
          + fvc::div(phi, rho)
          + fvc::div(phiGradp)
          - fvm::laplacian(rhorAUf, p)
        );
        pEqn.solve(mesh.solver(p.select(pimple.finalInnerIter())));
        if (pimple.finalNonOrthogonalIter())
        {
            phi += (phiGradp + pEqn.flux())/rhof;
        }
    }
    Info<< "Predicted p max-min : " << max(p).value()
        << " " << min(p).value() << endl;
    rho == max
    (
        psi*p
      + alphal*rhol0
      + ((alphav*psiv + alphal*psil) - psi)*pSat,
        rhoMin
    );
    #include "alphavPsi.H"
    p =
    (
        rho
      - alphal*rhol0
      - ((alphav*psiv + alphal*psil) - psi)*pSat
    )/psi;
    p.correctBoundaryConditions();
    Info<< "Phase-change corrected p max-min : " << max(p).value()
        << " " << min(p).value() << endl;
    // Correct velocity
    U = HbyA - rAU*fvc::grad(p);
    // Remove the swirl component of velocity for "wedge" cases
    if (pimple.dict().found("removeSwirl"))
    {
        label swirlCmpt(readLabel(pimple.dict().lookup("removeSwirl")));
        Info<< "Removing swirl component-" << swirlCmpt << " of U" << endl;
        U.field().replace(swirlCmpt, 0.0);
    }
    U.correctBoundaryConditions();
    Info<< "max(U) " << max(mag(U)).value() << endl;
}
```

The author has already used this solver, and applied it for the cavitation simulation inside rectangular nozzle. Therefore, the results will not be repeated again here. To see the applicability of the cavitatingFoam compared to the experimental data, refer to the previous papers of the author [17, 18].

4. DESCRIPTION of ”interPhaseChangeFoam” solver

Another solver to simulate the cavitation in OpenFOAM is interPhaseChangeFoam, which is located in the version of 2.3.x in $FOAM_SOLVERS/multiphase/interPhaseChangeFoam. The solver is described within the main interPhaseChangeFoam.C code, as follows

> *“Solver for 2 incompressible, isothermal immiscible fluids with phase-change (e.g. cavitation). Uses a VOF (volume of fluid) phase-fraction based interface capturing approach.*
>
> *The momentum and other fluid properties are of the "mixture" and a single momentum equation is solved.*
>
> *The set of phase-change models provided are designed to simulate cavitation but other mechanisms of phase-change are supported within this solver framework.*
>
> *Turbulence modelling is generic, i.e. laminar, RAS or LES may be selected.”*

4.1 Governing Equations

The interPhaseChangeFoam solver is based on the Volume of Fluid (VOF) model using mixture one-fluid approach. This solver is also using the same continuity Eq. (7) and momentum Eq. (8) equations, which are written for mixture, see section 3.1. The surface tension force (f_σ) is added into momentum equation Eq. (8), and it is calculated per unit volume via Continuum Surface Force (CSF) model [19] as

$$f_\sigma = \sigma \kappa \nabla \alpha \tag{10}$$

where σ and α display the surface tension coefficient and volume fraction of liquid. The curvature (κ) is defined as:

$$\kappa = -\nabla \left(\frac{\nabla \alpha}{|\nabla \alpha|} \right) \tag{11}$$

In OpenFOAM version 2.3.x, an improved version of VOF technique called “The Compressive Interface Capturing Scheme for Arbitrary Meshes (CICSAM)” [20] is

implemented, and used in the interPhaseChangeFoam solver. This is an explicit high resolution scheme, and can produce interface. In this model, an additional parameter "interface-compression velocity ($\mathbf{U}_c$)" in the surroundings of the interface is described to promote the interface resolution by steepening the gradient of the volume fraction function, which is represented in the transport equation for α as

$$\frac{\partial(\alpha\rho_l)}{\partial t} + \nabla\bullet(\alpha\rho_l\mathbf{U}) + \nabla\bullet\left[\alpha\mathbf{U}_c(1-\alpha)\right] = R_c - R_e \tag{12}$$

where R_c and R_e denote the rate of mass transfer for condensation and evaporation, respectively. The term in the square brackets is called artificial compression term. This term has a non-zero value only at the interface. The artificial compression describes the shrinkage of the phase-interphase towards a sharper one. This term does not affect the solution, and only defines the flow of α in the normal direction to the interface, which is proposed by Weller [21]

$$\mathbf{U}_c = \min\left[C_\alpha|\mathbf{U}|, \max(|\mathbf{U}|)\right]\frac{\nabla\alpha}{|\nabla\alpha|} \tag{13}$$

As indicated in the Eq. (13), the compression velocity is depending on maximum velocity at the interface, and it can be controlled by a constant parameter C_α, which is a scalar expression for limiting the artificial compression velocity. In the code, this parameter is named cAlpha, and can be found in \$FOAM_TUTORIALS/multiphase/interPhaseChangeFoam/cavitatingBullet/system/fvSolution in the "alpha.water.*" sub-dictionary for the version 2.3.x. If C_α=0, there is no compression, which means that the additional compression is simply skipped. If C_α=1, it introduces a conservative compression. If C_α>1, it stands for high compression.

The mixture density and viscosity, respectively, are computed in interPhaseChangeFoam solver, as follows

$$\rho_m = (1-\alpha)\rho_v + \alpha\rho_l \tag{14}$$

$$\mu_m = (1-\alpha)\mu_v + \alpha\mu_l \tag{15}$$

4.2 Solver members

The interPhaseChangeFoam solver in the version of OpenFOAM-2.3.x is located in

$FOAM_TUTORIALS/applications/solvers/multiphase/interPhaseChangeFoam and includes following file and folders.

- alphaEqn.H
- alphaEqnSubCycle.H
- createFields.H
- interPhaseChangeFoam.C
- UEqn.H
- pEqn.H
- phaseChangeTwoPhaseMixtures Folder
    - Kunz
        - Kunz.C
        - Kunz.H
    - Merkle
        - Merkle.C
        - Merkle.H
    - SchnerrSauer
        - SchnerrSauer C
        - SchnerrSauer.H
    - phaseChangeTwoPhaseMixture
        - newPhaseChangeTwoPhaseMixture.C
        - phaseChangeTwoPhaseMixture.C
        - phaseChangeTwoPhaseMixture.H
- Make
    - files
    - options

The main code of solver interPhaseChangeFoam.C in the version of OpenFOAM-2.3.x is as follows:

//**************************interPhaseChangeFoam.C************************ //

```
#include "CMULES.H"
#include "subCycle.H"
#include "interfaceProperties.H"
#include "phaseChangeTwoPhaseMixture.H"
```

```
#include "turbulenceModel.H"
#include "pimpleControl.H"
#include "fvIOoptionList.H"
#include "fixedFluxPressureFvPatchScalarField.H"
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
int main(int argc, char *argv[])
{
    #include "setRootCase.H"
    #include "createTime.H"
    #include "createMesh.H"
    #include "readGravitationalAcceleration.H"
    #include "initContinuityErrs.H"
    #include "createFields.H"
    #include "readTimeControls.H"
    pimpleControl pimple(mesh);
    #include "createPrghCorrTypes.H"
    #include "CourantNo.H"
    #include "setInitialDeltaT.H"
    // * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //
    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("0", dimMass/dimTime, 0)
            );
            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();
        Info<< "ExecutionTime = " << runTime.elapsedCpuTime() << " s"
            << "  ClockTime = " << runTime.elapsedClockTime() << " s"
            << nl << endl;
    }
    Info<< "End¥n" << endl;
    return 0;
}
// ************************************************************************* //
```

As it is seen in the above main code, there are two loops, which are α-loop (shown in the line: `#include "alphaEqnSubCycle.H"`), and pressure-loop (shown below the line: `//--- Pressure corrector loop`). When the time iteration loop starts, the solver first solves the correction of PIMPLE loop around of α-phase. After that, the courant number is calculated and the new time step is set according to value of courant number.

As seen in the "alphaEqnSubCycle.H" file below, there is a creation of surface scalar field phic for the calculation of mass flux in the beginning, and then, the solver seeks for the number of correctors to the loop around the α-equation and the number to sub-cycles. After that, two phase properties are calculated in the “alphaEqn.H” file, which is included in the "alphaEqnSubCycle.H" file. In the end, the mixture density is computed using weighted average of alpha field.

```
//*****************************alphaEqnSubCycle.H********************** //
    surfaceScalarField phic(mag(phi/mesh.magSf()));
    phic = min(interface.cAlpha()*phic, max(phic));
    volScalarField divU(fvc::div(phi));
    if (nAlphaSubCycles > 1)
    {
        dimensionedScalar totalDeltaT = runTime.deltaT();
        surfaceScalarField rhoPhiSum("rhoPhiSum", rhoPhi);
        for
        (
            subCycle<volScalarField> alphaSubCycle(alpha1,
nAlphaSubCycles);
```

```
            !(++alphaSubCycle).end();
        )
        {
            #include "alphaEqn.H"
            rhoPhiSum += (runTime.deltaT()/totalDeltaT)*rhoPhi;
        }
        rhoPhi = rhoPhiSum;
    }
    else
    {
        #include "alphaEqn.H"
    }
    rho == alpha1*rho1 + alpha2*rho2;
}
```

Further explanations about the another members of interPhaseChangeFoam solver can be found in the previous reports of this course [22, 23]

4.3 Cavitation Models

Cavitation models correspond to representation of the source terms which are indicated in the RHS of Eq. (12). For the interPhaseChangeFoam solver, it is already implemented three different cavitation models are already implemented. These are Merkle [24], Kunz [1], and SchnerrSauer [12]. The SchnerrSauer model is widely used and based on the bubble dynamics model using Rayleigh equation. The model assumptions and its drawbacks can be classified as follows:

- ✓ This model expresses the vapour fraction as a function of the radius of the bubbles, R, which is assumed to be the same for all the bubbles.
- ✓ It is also important to remind that Rayleigh theory was based on the balance of forces over spherical bubbles. It ignores bubble interactions, non-spherical bubble geometries, and local mass-momentum transfer around the interface.
- ✓ Another drawback of this method is that it requires estimation for the initial value of cavitation nuclei (n_0), bubble radius (R), and empirical constants for the condensation (C_c), and vaporization (C_v). The amount of these values affects the predicted cavity length and diameter.

Elaborated description and application of the SchnerrSauer model have already been given in the previous reports of this course [22, 23]. Hence, Kunz model will be described in detail

in this tutorial. The evaporation and condensation source terms in the RHS of Eq. (12) are according to Kunz model given as follows.

$$R_e = Cv \frac{\alpha \rho_v}{t_\infty (0.5 \rho_l \mathbf{U}_\infty{}^2)} \min[0, P - P_v] \tag{16}$$

$$R_c = Cc \frac{(1-\alpha)\alpha^2 \rho_v}{t_\infty} \tag{17}$$

respectively. U_∞, P_v, *Cc* and *Cv* show the mean stream velocity, vapor pressure (=2300 Pa), and empirical condensation and vaporization constants, which are set to 1000 in this project. Additionally, t_∞ is the mean flow time scale computed as L/U_∞, where L displays characteristics length scale (which is taken to be the nozzle length in this project).

4.3.1 Kunz.C

The main code of Kunz model in the version of OpenFOAM-2.3.x can be found under $FOAM_TUTORIALS/applications/solvers/multiphase/interPhaseChangeFoam/phaseChangeTwoPhaseMixures/Kunz/Kunz.C. Explanation of code is given part by part as follows.

```
// * * * * * * * * * * * * * * * * * * Constructors  * * * * * * * * * * * * * * * * * *//
Foam::phaseChangeTwoPhaseMixtures::Kunz::Kunz
(
    const volVectorField& U,
    const surfaceScalarField& phi
)
:
    phaseChangeTwoPhaseMixture(typeName, U, phi),
    UInf_(phaseChangeTwoPhaseMixtureCoeffs_.lookup("UInf")),
    tInf_(phaseChangeTwoPhaseMixtureCoeffs_.lookup("tInf")),
    Cc_(phaseChangeTwoPhaseMixtureCoeffs_.lookup("Cc")),
    Cv_(phaseChangeTwoPhaseMixtureCoeffs_.lookup("Cv")),
    p0_("0", pSat().dimensions(), 0.0),
    mcCoeff_(Cc_*rho2()/tInf_),
    mvCoeff_(Cv_*rho2()/(0.5*rho1()*sqr(UInf_)*tInf_))
{
    correct();
}
```

Description: ➔ In this section of code, initial constant parameters are read from the transportproperties file, which is located under the constant folder (within tutorial case), and corresponds to Eq. (17). It should be noted that these parameters are changeable related to the

models. Additionally, mcCoeff and mvCoeff are just intermediate coefficients, which are described here to calculate the multiplying parts in advance for the calculation, as shown in the RHS of Eq. (16) and Eq. (17).

```
Foam::phaseChangeTwoPhaseMixtures::Kunz::mDotAlphal() const
{
    const volScalarField& p =
alpha1_.db().lookupObject<volScalarField>("p");
volScalarField limitedAlpha1(min(max(alpha1_, scalar(0)), scalar(1)));
return Pair<tmp<volScalarField> >
(
        mcCoeff_*sqr(limitedAlpha1)
       *max(p - pSat(), p0_)/max(p - pSat(), 0.01*pSat()),
        mvCoeff_*min(p - pSat(), p0_)
    );
}

Foam::phaseChangeTwoPhaseMixtures::Kunz::mDotP() const
{
    const volScalarField& p =
alpha1_.db().lookupObject<volScalarField>("p");
    volScalarField limitedAlpha1(min(max(alpha1_, scalar(0)),
scalar(1)));
    return Pair<tmp<volScalarField> >
    (
        mcCoeff_*sqr(limitedAlpha1)*(1.0 - limitedAlpha1)
       *pos(p - pSat())/max(p - pSat(), 0.01*pSat()),
        (-mvCoeff_)*limitedAlpha1*neg(p - pSat())
    );
}
```

Description: ➔ All parameters used in the cavitation models are described inside $WM_PROJECT_DIR/applications/solvers/multiphase/interPhaseChangeFoam/phaseChangeTwoPhaseMixtures/phaseChangeTwoPhaseMixture/phaseChangeTwoPhaseMixture.H file. According to these definitions, "mDotAlphal()" and "mDotP()" are both volScalarField, and mDotAlphal() returns the mass condensation and vaporization rates as a coefficient to multiply "*(1 – alphal)*" for the condensation rate, and a coefficient to multiply "*alphal*" for the vaporisation rate. On the other hand, mDotP() returns the mass condensation and vaporization rates as an explicit term for the condensation rate, and a coefficient to multiply "(p-psat)" for the vaporization rate. Also, pressure is inserted into code by lookupObject. Explanations related to test-case of interPhaseChangeFoam tutorial and its another

applications can be found in the previous studies [18, 25, 26] and previous reports of this course [27].

5. IMPLEMENTATION OF TRANPORT EQUATION MODEL INTO cavitatingFoam

Before starting the implementation of transport alpha equation, and modification of the cavitatingFoam, it is advised to copy the original solver to a new location, and rename it to TransportCavitatingFoam.

```
mkdir -p $WM_PROJECT_USER_DIR/applications/solvers/multiphase
cd $WM_PROJECT_USER_DIR/applications/solvers/multiphase
cp -r $FOAM_SOLVERS/multiphase/cavitatingFoam .
mv cavitatingFoam TransportCavitatingFoam
cd TransportCavitatingFoam
mv cavitatingFoam.C TransportCavitatingFoam.C
```

In order to remove unused folder and the files coming from the previous compilation of cavitatingFoam, run:

```
rm -r cavitatingDyMFoam/
wclean
```

5.1 Implementation Procedure

As explained in chapter 3, the cavitatingFoam is compressible solver based on barotropic closure, which means that cavitation is modeled with barotropic compressibility model. Therefore, we should remove the barotropic and compressibility implementations from the cavitatingFoam solver, and then implement the incompressible transport alpha equation into that solver. Before explaining the modifications step by step, first copy the alphaEqn.H and alphaEqnSubCycle.H files from interPhaseChangeFoam solver into TransportCavitatingFoam folder, and remove the barotropic alpha equation.

```
cp -r $FOAM_SOLVERS/multiphase/interPhaseChangeFoam/alphaEqn.H .
cp -r $FOAM_SOLVERS/multiphase/interPhaseChangeFoam/alphaEqnSubCycle.H .
rm alphavPsi.H
```

5.1.1 Modifications in TransportCavitatingFoam.C file

Insert following lines at the beginning of the solver.

```
#include "CMULES.H"
#include "subCycle.H"
#include "phaseChangeTwoPhaseMixture.H"
#include "fvIOoptionList.H"
#include "readTimeControls.H"
#include "interfaceProperties.H"
```

```
#include "fixedFluxPressureFvPatchScalarField.H"
```

Remove the following lines that are related to barotropic compressibility model.

```
#include "incompressibleTwoPhaseMixture.H"
#include "readThermodynamicProperties.H"
#include "readControls.H"
#include "compressibleCourantNo.H"
```

Then insert the following lines below pimpleControl pimple(mesh);

```
#include "createPrghCorrTypes.H"
#include "correctPhi.H"
#include "CourantNo.H"
#include "setInitialDeltaT.H"
```

As seen above, the correctPhi.H file is added to correct the calculated mass flux. Therefore, copy it from interFoam solver into TransportCavitatingFoam folder.

```
cp -r $FOAM_SOLVERS/multiphase/interFoam/correctPhi.H .
```

Under the "Pressure-velocity PIMPLE corrector loop" implemented alpha equation should be solved. Therefore, between pimple loop and #include "UEqn.H" remove the following lines:

```
#include "rhoEqn.H"
#include "alphavPsi.H"
```

And then add the following lines:

```
#include "alphaControls.H"
surfaceScalarField rhoPhi
(
IOobject
(
   "rhoPhi",
   runTime.timeName(),
   mesh
),
mesh,
dimensionedScalar("0", dimMass/dimTime, 0)
);
if (pimple.firstIter() || alphaOuterCorrectors)
{
twoPhaseProperties->correct();
#include "alphaEqnSubCycle.H"
interface.correct();
}
```

Due to removing .H lines related to barotropic compressibility model from .C file, we should also delete them from the main folder of the solver:

```
rm readThermodynamicProperties.H
rm continuityErrs.H
rm readControls.H
rm CourantNo.H
rm setDeltaT.H
rm setInitialDelta.T
```

Finally, to watch the max/min pressure and velocity output data on screen while running the simulation, add the following lines inside the time loop.

```
Info<< "Max pressure = " << max(p).value() << endl;
Info<< "Min pressure = " << min(p).value() << endl;
Info<< "Max velocity = " << max(mag(U)).value() << endl <<endl;
Info<< "Max velocity = " << min(mag(U)).value() << endl <<endl;
```

5.1.2 Modifications inside createFields.H file

In the cavitatingFoam solver, P is calculated using a compressibility model. However, in the incompressible transport model, hydrostatic pressure definition is used. Therefore, first change the pressure notation as follow:

```
Info<< "Reading field p_rgh\n" << endl;
      volScalarField p_rgh
         (
             IOobject
             (
                 "p_rgh",
                 runTime.timeName(),
                 mesh,
                 IOobject::MUST_READ,
                 IOobject::AUTO_WRITE
             ),
             mesh
         );
```

After that, add the definition of the hydrostatic pressure at the end of the createFields.H file. Also, copy the pEqn.H file from interPhaseChangeFoam solver to TransportCavitatingFoam to replace the present one, which is calculated according to barotropic compressibility model.

```
    Info<< "Calculating field g.h\n" << endl;
    volScalarField gh("gh", g & mesh.C());
    surfaceScalarField ghf("ghf", g & mesh.Cf());
    volScalarField p
    (
        IOobject
        (
            "p",
            runTime.timeName(),
            mesh,
```

```
            IOobject::NO_READ,
            IOobject::AUTO_WRITE
        ),
        p_rgh + rho*gh
    );
    label pRefCell = 0;
    scalar pRefValue = 0.0;
    setRefCell
    (
        p,
        p_rgh,
        mesh.solutionDict().subDict("PIMPLE"),
        pRefCell,
        pRefValue
    if (p_rgh.needReference())
    {
        p += dimensionedScalar
        (
            "p",
            p.dimensions(),
            pRefValue - getRefCellValue(p, pRefCell)
        );
        p_rgh = p - rho*gh;
    }

    fv::IOoptionList fvOptions(mesh);
```

In the transport model, the mixture density "rho" will be calculated according to Eq. (14). Therefore, modify the corresponding line as follows:

```
    volScalarField rho
    (
         IOobject
         (
             "rho",
             runTime.timeName(),
             mesh,
             IOobject::READ_IF_PRESENT
         ),
         alpha1*rho1 + alpha2*rho2,
         alpha1.boundaryField().types()
     );
     rho.oldTime();
```

Remove the following lines, which create the face-flux field "phi" based on compressibility.

```
#include "createPhiv.H"
#include "compressibleCreatePhi.H"
```

Then, add following line, which creates and initializes the relative face-flux field phi.

```
#include "createPhi.H"
```

As barotropic closure with compressibility will not be used, delete the following lines:

```
    volScalarField& alphav(mixture.alpha1());
    alphav.oldTime();
    volScalarField& alphal(mixture.alpha2());
    Info<< "Creating compressibilityModel/n" << endl;
    autoPtr<barotropicCompressibilityModel> psiModel =
        barotropicCompressibilityModel::New
        (
            thermodynamicProperties,
            alphav
        );
    const volScalarField& psi = psiModel->psi();
    rho == max
    (
        psi*p
      + alphal*rhol0
      + ((alphav*psiv + alphal*psil) - psi)*pSat,
        rhoMin
    );
```

As a phase change TwoPhaseMixture will be used inside transport equation. Therefore, add the following lines instead of the code above.

```
    Info<< "Creating phaseChangeTwoPhaseMixture\n" << endl;
    autoPtr<phaseChangeTwoPhaseMixture> mixture =
    phaseChangeTwoPhaseMixture::New(U, phi);
    volScalarField& alpha1(mixture->alpha1());
    volScalarField& alpha2(mixture->alpha2());
    const dimensionedScalar& rho1 = mixture->rho1();
    const dimensionedScalar& rho2 = mixture->rho2();
    const dimensionedScalar& pSat = mixture->pSat();
    interfaceProperties interface(alpha1, U, mixture());
```

5.1.3 Copy and modification of phaseChangeTwoPhaseMixtures folder

As mentioned before, Kunz model [1] is used for the mass transfer modelling of cavitation. Therefore, to include this model inside of TransportCavitatingFoam solver, copy the folder of phaseChangeTwoPhaseMixtures from the interPhaseChangeFoam solver.

```
cp -r
$FOAM_SOLVERS/multiphase/interPhaseChangeFoam/phaseChangeTwoPhaseMixtures .
```

Then go to inside this folder, remove previous dependencies, and delete other models which are not used:

```
cd  phaseChangeTwoPhaseMixtures/
wclean
rm -r SchnerrSauer/
rm -r Merkle/
```

Go to the Make/files and make the following changes:

```
Kunz/Kunz.C

LIB = $(FOAM_USER_LIBBIN)/libphaseChangeTwoPhaseMixtures
```

After that, go to Make/options file, and add the following lines to first check the USER folder for the library files:

```
Kunz/Kunz.C

LIB_LIBS = \

    -L$(FOAM_USER_LIBBIN) \
```

Finally, compile the library:

```
wmake libso
```

If everything is OK, the new library will be created in the FOAM_USER_LIBBIN directory as libphaseChangeTwoPhaseMixtures.so. To check this run:

```
ls $FOAM_USER_LIBBIN
```

5.1.4 Modifications in Make/files and Make/options of main solver

First modify the *Make/files* as follows to write the executables of new solver as

```
$FOAM_USER_APPBIN:

    EXE = $(FOAM_USER_APPBIN)/TransportCavitatingFoam
```

After that, remove the following lines from Make/options, which are not used:

```
EXE_INC = \
-I$(LIB_SRC)/thermophysicalModels/barotropicCompressibilityModel/lnInclude

EXE_LIBS = \
 -lincompressibleTransportModels \
 -lbarotropicCompressibilityModel \
```

and add these lines:

```
EXE_INC = \
  -IphaseChangeTwoPhaseMixtures/lnInclude \
```

```
  -I$(LIB_SRC)/meshTools/lnInclude \
  -I$(LIB_SRC)/fvOptions/lnInclude \
  -I$(LIB_SRC)/sampling/lnInclude \

EXE_LIBS = \
 -lphaseChangeTwoPhaseMixtures \
 -lmeshTools \
 -lfvOptions \
 -lsampling \
```

5.1.5 Modification of UEqn.H file

As a different to present UEq.H file, add the following lines at the end:

```
UEqn.relax();

    if (pimple.momentumPredictor())
    {
        solve
        (
            UEqn
         ==
            fvc::reconstruct
            (
                (
            fvc::interpolate(interface.sigmaK())*fvc::snGrad(alpha1)
                - ghf*fvc::snGrad(rho)
                - fvc::snGrad(p_rgh)
               ) * mesh.magSf()
            )
        );
    }
```

Finally, the TransportCavitatingFoam is ready to be compiled:

```
wmake
```

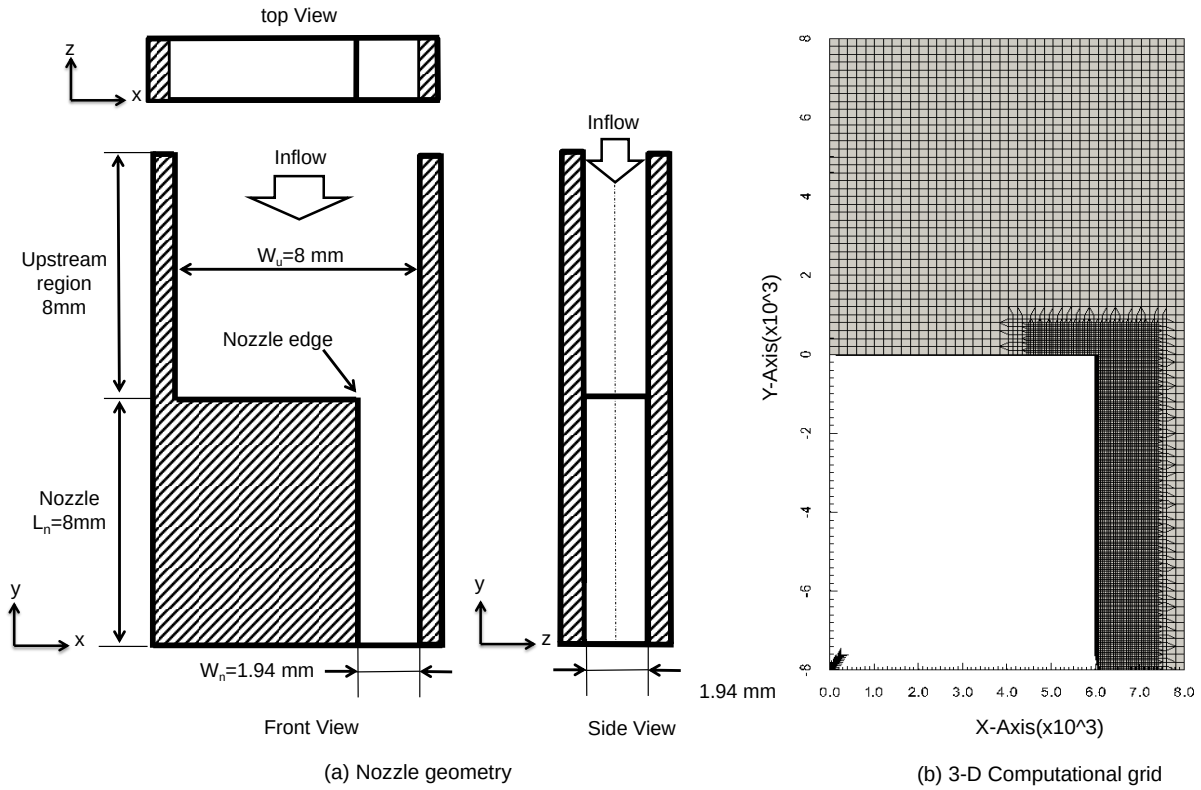
If all the modifications were done correctly, the new solver binary could be found in the FOAM_USER_APPBIN directory as named TransportCavitatingFoam. To check this run:

```
ls $FOAM_USER_APPBIN
```

5.2 Test Case

The cavitating turbulent flow inside a rectangular nozzle has been simulated using TransportCavitatingFoam new solver for the test case. A test case named *rectangular_nozzle_test_case*, which contains *0/, constant/* and *system/* folders, is already provided to users. Validation of the test case has been done using our previous experimental data [28] in terms of the cavitation profile in the nozzle.

Figure 1 shows nozzle geometry and 3D computational grid of the whole domain. A structured grid is used for the simulation with 73,100 hexahedral cells with minimum cell size 50 µm. The grid is created by OpenFOAM blockMesh utility with the refineMesh function.



(a) Nozzle geometry (b) 3-D Computational grid

Fig. 1. Nozzle geometry and 3-D computational grids

The inlet velocity is set to 3.2 m/s (which correspond to 0.22 MPa for the injection pressure), while the outlet pressure is fixed to the environment pressure of 0.10 MPa. The nozzle flow is considered as turbulent, since Reynolds number is higher than 20,000. Therefore, turbulence effects have been introduced using RANS methods such as RNG k-ε model. Additionally, Kunz cavitation model empirical constants *Cc* and *Cv* are set to 1000. Other initial settings can be found within the provided test-case.

An Intel Core i7-980X @ 3.33 GHz x 6 core (12 CPU), 12 GB RAM pc is used. The time step Δt and the maximum Courant number are set to 10^{-8} s and 0.1, respectively. The calculation takes approximately 1.5 days. Second order Gauss linear upwind scheme is used for the advection terms' discretization for velocity, whereas first order upwind scheme is used for the turbulent parameters due to stability problem. The water was used as a working fluid both in the experiment and numerical calculations.

Before running the code, go to system/conrolDict file and change the application name to:

```
application     TransportCavitatingFoam
```

then go to test-case directory and run the code:

```
TransportCavitatingFoam &>log
```

5.3 Results

Figure 2 top (a) shows the experimental results, while top (b), bottom (c) and (d) are the calculated results obtained using the new implemented solver "TransportCavitatingFoam" in terms of liquid volume fraction, pressure contours and velocity vectors.

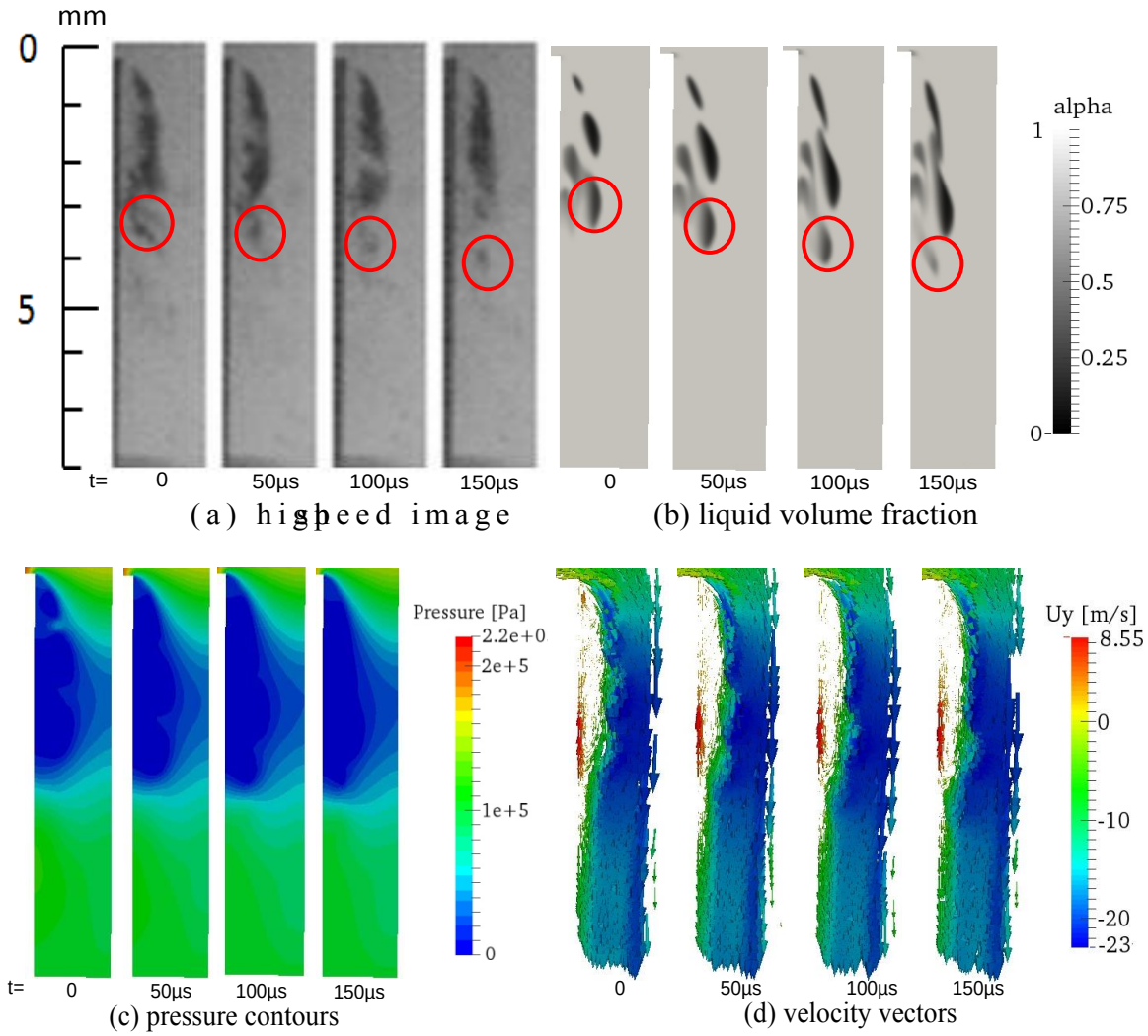


Fig. 2. Experimental and calculated results
(P_{in} = 0.22 MPa, results are shown at every 50 µs)

Figure 2 (a) is taken using a high-speed camera whose exposure time is 50 μs. It is seen that in Fig. 2 (a), a great number of nuclei grow in the low pressure regions within the recirculation zone and the vortices shedding from the reattachment point. Further information about the experimental study can be found in the previous author's publication [28]. Figure 2 (b) displays calculated liquid volume fraction and, as seen here, transient motion of cavitation, shedding of a vortex accompanied by clouds of bubbles, and the collapse of the bubbles are well simulated with the new implemented TransportCavitatingFoam solver. This phenomenon can also be observed in the provided movie.

6. REFERENCES

[1] R.F. Kunz, D.A. Boger, D.R. Stinebring, T.S. Chyczewski, J.W. Lindau, H.J. Gibeling, S. Venkateswaran, T.R. Govindan, A preconditioned Navier–Stokes method for two-phase flows with application to cavitation prediction, Computers & Fluids, 29 (2000) 849-875.

[2] W. Bergwerk, Flow pattern in diesel nozzle spray holes, Proceedings of the Institution of Mechanical Engineers, 173 (1959) 655-660.

[3] W. Nurick, Orifice cavitation and its effect on spray mixing, ASME Transactions Journal of Fluids Engineering, 98 (1976) 681-687.

[4] H. Hiroyasu, Break-up Length of a Liquid Jet and Internal Flow in a Nozzle, ICLASS-91, (1991) 275-282.

[5] H. Chaves, M. Knapp, A. Kubitzek, Experimental study of cavitation in the nozzle hole of diesel injectors using transparent nozzles, (1995).

[6] C. Soteriou, R.J. Andrews, Direct injection diesel sprays and the effect of cavitation and hydraulic flip on atomization, SAE paper 950080, (1995).

[7] A. Sou, S. Hosokawa, A. Tomiyama, Effects of cavitation in a nozzle on liquid jet atomization, International journal of heat and mass transfer, 50 (2007) 3575-3582.

[8] L. He, F. Ruiz, Effect of cavitation on flow and turbulence in plain orifices for high-speed atomization, 5 (1995) 569-584.

[9] M. Daikoku, H. Furudate, H. Noda, T. Inamura, Effect of cavitation in the two-dimensional nozzle on liquid breakup process, in: Proc. 9th ICLASS, 2003, pp. 1207.

[10] D.P. Schmidt, C. Rutland, M. Corradini, P. Roosen, O. Genge, Cavitation in two-dimensional asymmetric nozzles, SAE paper, (1999) 0518.

[11] W. Yuan, J. Sauer, G.H. Schnerr, Modeling and computation of unsteady cavitation flows in injection nozzles, Mécanique & industries, 2 (2001) 383-394.

[12] G. Schnerr, J. Sauer, Physical and numerical modeling of unsteady cavitation dynamics, in: Fourth International Conference on Multiphase Flow, New Orleans, USA, 2001.

[13] A.K. Singhal, M.M. Athavale, H. Li, Y. Jiang, Mathematical basis and validation of the full cavitation model, Transactions-American Society of Mechanical Engineers Journal of Fluids Engineering, 124 (2002) 617-624.

[14] P.J. Zwart, A.G. Gerber, T. Belamri, A two-phase flow model for predicting cavitation dynamics, in: Fifth International Conference on Multiphase Flow, Yokohama, Japan, 2004.

[15] G.B. Wallis, One-dimensional two-phase flow, McGraw-Hill New York, 1969.

[16] F.P. Kärrholm, H. Weller, N. Nordin, Modelling injector flow including cavitation effects for diesel applications, in, ASME, 2007.

[17] B. Bicer, A. Tanaka, T. Fukuda, A. Sou, Numerical Simulation of Cavitation Phenomena in Diesel Injector Nozzles, 16th Annual Conf. ILASS-ASIA, Nagasaki-JAPAN (2013), 58-65.

[18] B. Biçer, S. Akira, NUMERICAL SIMULATION OF TURBULENT CAVITATING FLOW IN DIESEL FUEL INJECTOR, Proceedings of the 3rd International Symposium of Maritime Sciences, Kobe, Japan, 3 (2014) 33-38.

[19] J. Brackbill, D.B. Kothe, C. Zemach, A continuum method for modeling surface tension, Journal of computational physics, 100 (1992) 335-354.

[20] O. Ubbink, Numerical prediction of two fluid systems with sharp interfaces, in, University of London UK, 1997.

[21] H. Weller, A new approach to VOF-based interface capturing methods for incompressible and compressible flow, OpenCFD Ltd., Report TR/HGW/04, (2008).

[22] N. Lu, Tutorial: Solve cavitating flow around a 2D hydrofoil using a user modified version of interPhaseChangeFoam, MSc/PhD course in CFD with OpenSource software, (2008).

[23] A. Asnaghi, interPhaseChangeFoam tutorial and PANS turbulence model, MSc/PhD course in CFD with OpenSource software, (2013).

[24] C.L. Merkle, J. Feng, P.E. Beulow, Computational modeling of the dynamics of sheet cavitation, 3rd International symposium on cavitation, Grenoble, France, 2 (1998) 47-54.

[25] C. Deimel, M. Günther, R. Skoda, Application of a pressure based CFD code with mass transfer model based on the Rayleigh equation for the numerical simulation of the cavitating flow around a hydrofoil with circular leading edge, in: EPJ Web of Conferences, EDP Sciences, 2014, pp. 02018.

[26] E. Roohi, A.P. Zahiri, M. Passandideh-Fard, Numerical simulation of cavitation around a two-dimensional hydrofoil using VOF method and LES turbulence model, Applied Mathematical Modelling, 37 (2013) 6469-6488.

[27] M. Andersen, A interPhaseChangeFoam tutorial, MSc/PhD course in CFD with OpenSource software, (2011).

[28] A. Sou, B. Biçer, A. Tomiyama, Numerical simulation of incipient cavitation flow in a nozzle of fuel injector, Computers & Fluids, 103 (2014) 42-48.