# How To Use Free-Surface Flows in OpenFOAM v. 4.1

**Instructor**: Victoria Korchagova, ISP RAS

**Training level**: Intermediate

**Session type**: Lecture with examples

**Software stack**: OpenFOAM v.4.1

***www.unicfd.ru***

# Plan of training course

1. Introduction
2. Key points of training course
3. interFoam solver: how it works
   - Governing equations
   - Volume of Fluid method
   - Solution process
   - Boundary conditions
4. Basic case: Spillway tutorial
   - Problem statement
   - Physical properties setup
   - Mesh generation
   - Boundary conditions setup
   - Numerical schemes and time advancement settings. Running.
   - Results
5. Additional case: RT tutorial
   - Problem statement
   - Settings in OpenFOAM. Running.
   - Results. Comparison with linear theory.
6. Conclusions and discussion

# Introduction

## Applicability

- printing;
- engines;
- ecological cases;
- dams, spillways;
- etc.

## Complexities

- large deformations of the interface;
- creation of different subregions (droplets, bubbles. . . );
- solution should:
  - be stable;
  - have small diffusivity in the interface region;
  - satisfy to conservation laws;
  - be correct in different scales;
  - require not so much resources for computations.

# Key points of training course

- Look how the solver for free-surface flows works
- Study how to set boundary conditions in different versions of OpenFOAM
- Look to all stages of modeling of free-surface flows in OpenFOAM v. 4.1: from mesh generation to post-processing

Boundary conditions are critically important in the successful modeling.
There are strong differences between OpenFoam 2.2.x and 2.3.0+.

---

The main tool: an interFoam solver in OF v. 4.1.
We will study it with Spillway tutorial: the turbulent flow of fluid

# Part I
# Theoretical part

---

interFoam: how it works

# Structure of theoretical part

1. Governing equations
2. Volume of Fluid method
3. Block scheme of interFoam
4. Pressure-velocity coupling
5. Boundary conditions (most common types):
   - walls and inlets;
   - outlets and open boundaries;
   - planes of symmetry.

# Governing equations for incompressible flow

- Continuity equation:

$$\nabla \cdot \mathbf{U} = 0;$$

- Navier — Stokes equations:

$$\frac{\partial(\rho \mathbf{U})}{\partial t} + \nabla \cdot (\rho \mathbf{U} \otimes \mathbf{U}) = -\nabla p + \nabla \cdot \hat{\tau} + \rho \mathbf{g},$$

  where $\hat{\tau} = \mu(\nabla \mathbf{U}^T + \nabla \mathbf{U})$ is the viscous stress tensor;

- boundary conditions on the interface:

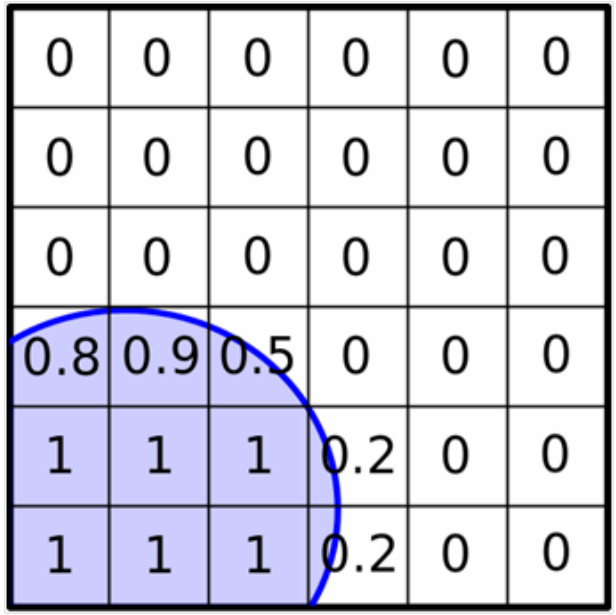
$$[-p\mathbf{I} + \hat{\tau}] \cdot \mathbf{n} = \sigma \kappa \mathbf{n}, \quad [\mathbf{U}] = 0;$$

- initial and boundary conditions on the flow region boundaries (different types — walls, inlets, outlets, open boundaries and other).

# Volume of Fluid method

Add a volume fraction function:

$$\alpha_1 = \begin{cases} 1 \text{ if cell full of liquid;} \\ 0 \text{ if cell full of gas;} \\ (0;1) \text{ if cell is placed on the interface;} \end{cases}$$




For two phases

$$\alpha_1 + \alpha_2 = 1.$$

Solve a transport equation for this function:

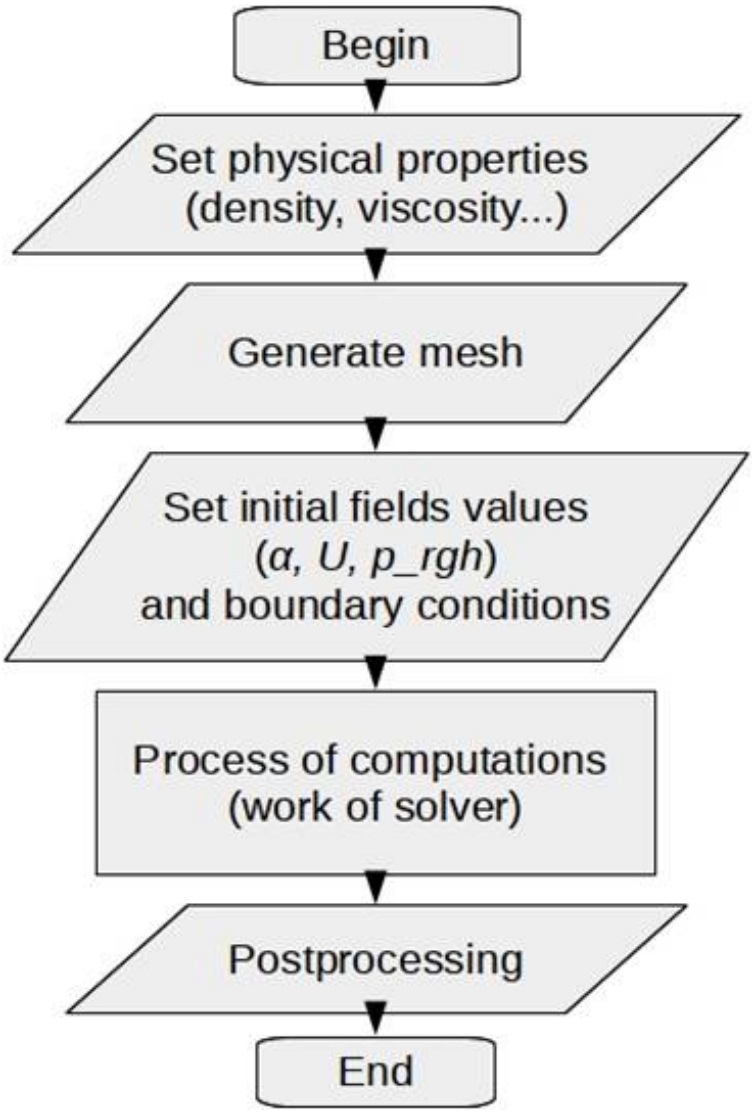
$$\frac{\partial \alpha_1}{\partial t} + \nabla \cdot (\mathbf{U}\alpha_1) + \nabla \cdot (\mathbf{U}_R \alpha_1 \alpha_2) = 0,$$

where $\mathbf{U} = \alpha_1 \mathbf{U}_1 + \alpha_2 \mathbf{U}_2$ — velocity of mixture;
$\mathbf{U}_R = \mathbf{U}_1 - \mathbf{U}_2$ — velocity field suitable to compress the interface.

---

Approximation: **Finite Volume Method**

# Solution process




Set manually on files in the folder of case

Use utilities blockMesh, snappyHexMesh or third-party software

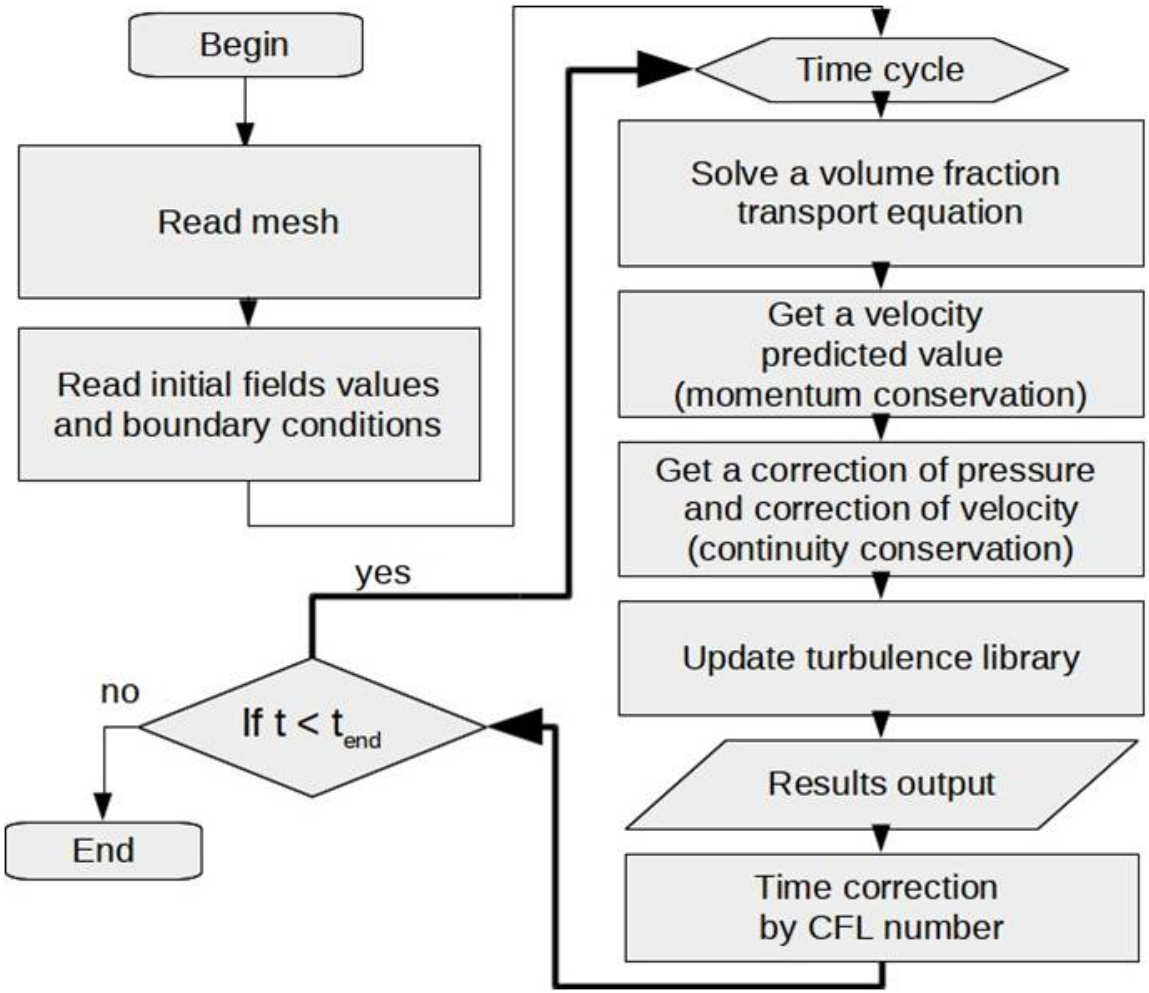
Set manually, use setFields utility

interFoam solver

View your results in Paraview

# interFoam solver

## Block scheme




see `alphaEqn.H`[1] ←

see `UEqn.H` ←

see `pEqn.H` ←

Location: `OpenFoam-4.1/applications/solvers/multiphase/interFoam`

[1] see slide 8

# interFoam solver

## Modified Navier — Stokes equations

Surface tension forces are approximated as an additional term $\mathbf{F}_\sigma$ in Navier — Stokes equation[2]:

$$\frac{\partial(\rho\mathbf{U})}{\partial t} + \nabla\cdot(\rho\mathbf{U}\otimes\mathbf{U}) = -\nabla p + \nabla\cdot\hat{\tau} + \rho\mathbf{g} + \mathbf{F}_\sigma.$$

Use modified pressure:

$$p^* = p - \rho(\mathbf{g}\cdot\mathbf{r}).$$

Its gradient:

$$\nabla p^* = \nabla p - (\nabla\rho)(\mathbf{g}\cdot\mathbf{r}) - \rho\mathbf{g}.$$

Approximation of surface tension forces — by volume fraction gradient:

$$\mathbf{F}_\sigma \approx \sigma\kappa\nabla\alpha_1.$$

[2] see slide 7

# interFoam solver I

## Pressure-velocity coupling

Use velocity as sum of prediction and correction parts:

$$\mathbf{U} = \mathbf{U}^* + \mathbf{U}'. \tag{1}$$

**Semi-discrete form** of momentum equation:

$$A\mathbf{U} = H - \nabla p^* - (\nabla \rho)(\mathbf{g} \cdot \mathbf{r}) + \mathbf{F}_\sigma.$$

Here $A$ is the diagonal part of initial matrix system,
$H$ is the non-diagonal part of matrix + r.h.s. without diagonal part of initial matrix system, pressure gradient and terms for gravity and surface tension.

We can write comparing with the velocity splitting (1):

$$\mathbf{U}^* = A^{-1}H;$$

$$\boxed{\mathbf{U}' = -A^{-1}\nabla p^* - A^{-1}(\nabla \rho)(\mathbf{g} \cdot \mathbf{r}) + A^{-1}\mathbf{F}_\sigma.}$$

# interFoam solver II

## Pressure-velocity coupling

Continuity equation in the discrete form:

$$\int_V \nabla \cdot \mathbf{U}\, dV = \int_S \mathbf{U} \cdot \mathbf{n}\, dS \approx \sum_f \underbrace{\mathbf{U}_f \cdot \mathbf{S}_f}_{\varphi_f} = 0, \tag{2}$$

consequently,

$$\boxed{\sum_f \varphi_f = 0.} \tag{3}$$

Let's calculate fluxes through one face $f$:

$$\underbrace{\mathbf{U}_f \cdot \mathbf{S}_f}_{\varphi} = \underbrace{\mathbf{U}_f^* \cdot \mathbf{S}_f}_{\varphi_{H/A}} - \underbrace{(A^{-1})_f}_{D_p} (\nabla p^*)_f \cdot \mathbf{S}_f - \\ - \underbrace{(A^{-1}(\nabla \rho)(\mathbf{g} \cdot \mathbf{r}))_f \cdot \mathbf{S}_f + (A^{-1}\mathbf{F}_\sigma)_f}_{\varphi_g}. \tag{4}$$

# interFoam solver III

## Pressure-velocity coupling

Create **a pressure equation** which is derived from continuity equation (3):

$$\sum_f (\varphi_{H/A} + \varphi_g) = \sum_f D_p (\nabla p^*)_f \cdot \mathbf{S}_f.$$

**Pressure gradient**:

$$(\nabla p^*)_f = \frac{\varphi_{H/A} + \varphi_g - \varphi}{D_p \cdot \mathbf{S}_f}. \tag{5}$$

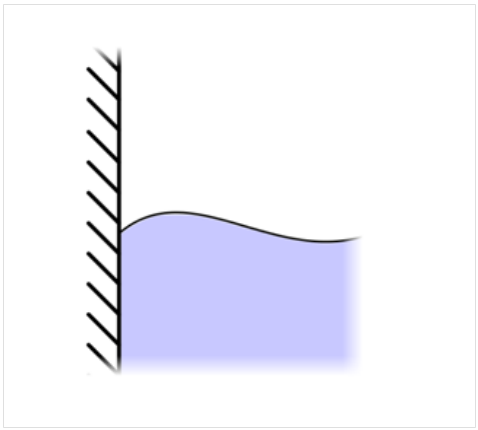
In the source code:

```
phiHbyA = phiHbyA + phig;
snGrad(p_rgh) = (phiHbyA - phi) / (rAUf * Sf).
```

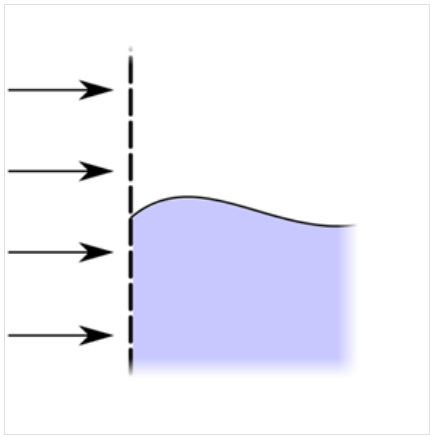
# interFoam solver

## Types of boundary conditions

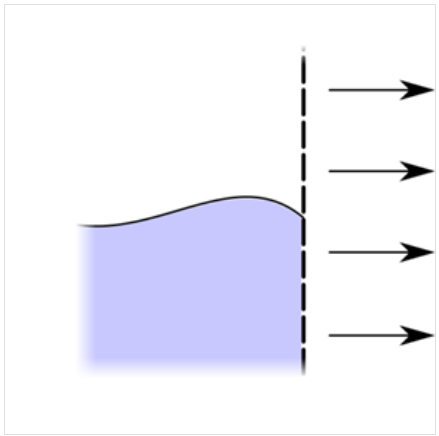
**Wall**



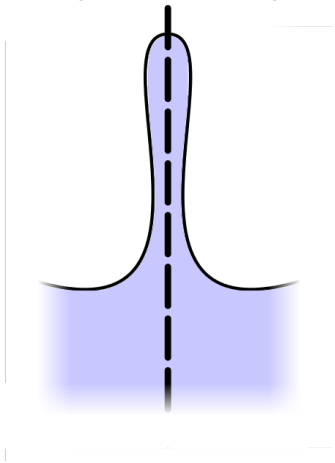
**Inlet**



**Outlet**



**Symmetry**



### Note

For incompressible fluids it is important to calculate correctly the pressure gradient. Absolute value of pressure is calculated up to a constant.
So, it is enough to know the pressure reference value only in one point.

**Location**: `OpenFoam-4.1/src/finiteVolume/fields/fvPatchFields`

# interFoam solver I

## Walls & inlets

Boundary condition for **volume fraction**:

$$\alpha = 0 \text{ or } \alpha = 1 \ (\texttt{fixedValue}) \text{ — for inlets;}$$
$$\nabla\alpha = 0 \ (\texttt{zeroGradient}) \text{ — for walls.}$$

Boundary condition for **velocity**:

$$\mathbf{U} = \{U_x; U_y; U_z\} \ (\texttt{fixedValue}).$$

### Note

Capillary effects are neglected; to account for these effects — special boundary conditions for volume fraction must be imposed.

Boundary conditions for **pressure** are critically important here.
Let's consider the correct boundary condition and changes in OpenFOAM v.2.3.0+.

# interFoam solver II

## Walls & inlets

On boundaries for walls and inlets:

$$\mathbf{U} = \{U_x; U_y; U_z\}, \quad \mathbf{U}^* = \{U_x; U_y; U_z\} \Rightarrow \mathbf{U}' = \{0; 0; 0\}.$$

According to BC for volume fraction:

$$\mathbf{F}_\sigma \approx \sigma\kappa\nabla\alpha_1 = 0 \text{ on the boundary.}$$

So, according to expression for volumetric flux (4):

$$-D_p(\nabla p^*)_f \cdot \mathbf{S}_f - (A^{-1}(\nabla\rho)(\mathbf{g}\cdot\mathbf{r}))_f \cdot \mathbf{S}_f) = 0,$$

and, therefore, we can derive boundary condition for pressure:

$$\boxed{(\nabla p^*)_f = -((\nabla\rho)(\mathbf{g}\cdot\mathbf{r}))_f.}$$

# interFoam solver III

## Walls & inlets

### Implementation in OpenFOAM

In OpenFOAM v.2.2.x — `buoyantPressure`: this expression is in the source code of boundary condition.

In OpenFOAM v.2.3.0+ — `fixedFluxPressure`: boundary condition is satisfied automatically by pressure gradient (5), calculating in the solver's source code.

# interFoam solver I

## Outlets

Boundary condition for **volume fraction**:

$$\nabla \alpha = 0 \ (\texttt{zeroGradient}).$$

Boundary condition for **velocity**:

$$\nabla \mathbf{U} = 0 \ (\texttt{zeroGradient}).$$

Boundary condition for **pressure**:

- reference level of pressure — if there are no pressure boundary conditions in any another boundary (typically means “atmosphere”):

$$p_p^* = p_0 - \frac{U^2}{2} \ (\texttt{totalPressure}),$$

where $p_0$ — total pressure, $U$ — velocity magnitude.

# interFoam solver II

## Outlets

In the source code we use:

```
p = p0 - 0.5*(1 - pos(phi))*magSqr(U).
```

Here `pos()` is the boolean function which equals to 1 when the flow leaves the domain, i.e. `phi>0`.

- `zeroGradient` — if you have some another boundary with derived reference level of pressure.

---

In case of **symmetry** just use `slip` boundary condition for all variables.

**Note**

Calculations of patch boundary field on the symmetry planes are performed using the Householder projection on the patch.

# Part II
# Practical part

---

## Basic case: Spillway tutorial

# Stages of simulation

1. Geometry: make STL-surface to draw the dam (in SALOME).
2. Liquid/gas properties: write density, kinematic viscosity ...
3. Mesh generation:
   - blockMesh: create a basic mesh box;
   - snappyHexMesh: refine mesh near the dam surface;
   - extrudeMesh: make a 2D-mesh for fast calculations.
4. Boundary conditions: define it with files in `0.org` folder.
5. Set fields: set initial liquid phase volume fraction.
6. Numerical settings:
   - describe interpolation of terms;
   - describe solvers for SLAE;
   - setup turbulence models.
7. Time settings: set the end time, CFL-number ...
8. Running: an interFoam command.
9. Post-processing: open file `<filename>.foam` in Paraview.

# Problem statement

## Input data

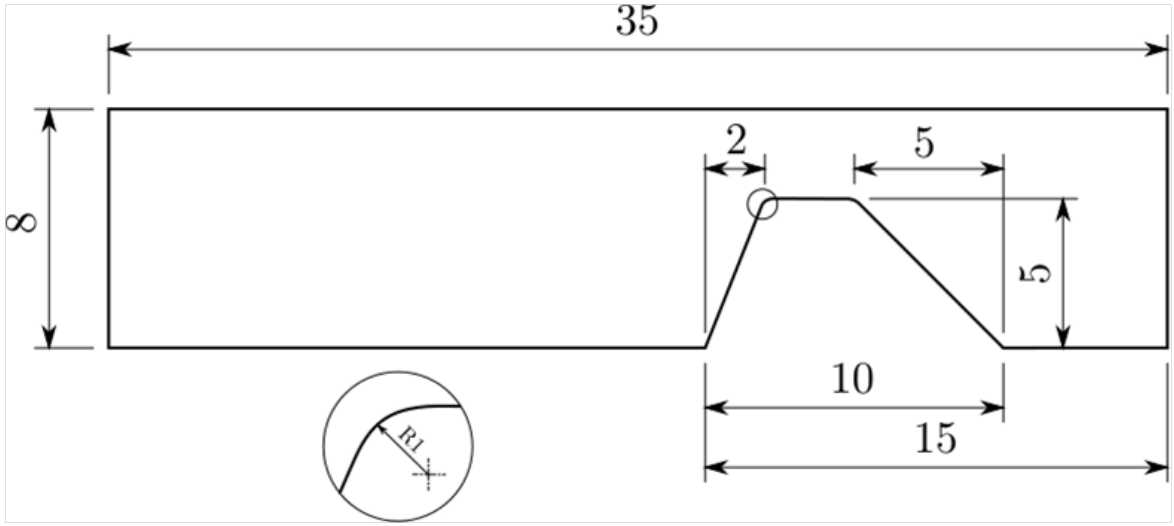
Water:
**density**: 1000 kg/m$^3$;
**dynamic viscosity**:
$10^{-3}$ Pa· s.

Air:
**density**: 1 kg/m$^3$;
**dynamic viscosity**:
$1.48 \cdot 10^{-5}$ Pa· s.

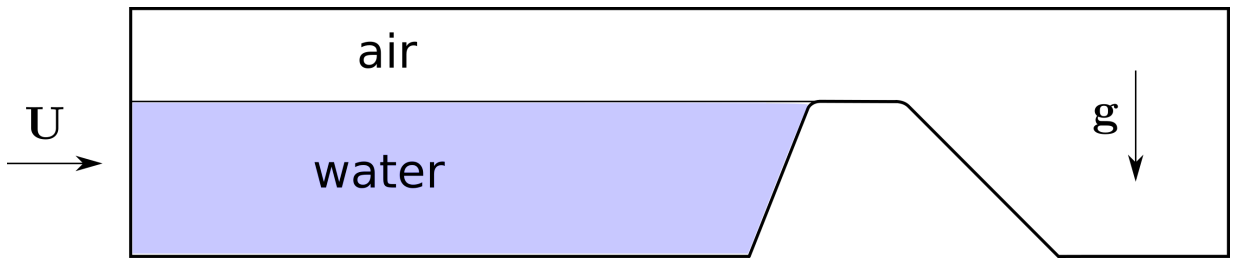
Surface tension coefficient: 0.07 N/m.

Inlet velocity: 0.6 m/s.

**Geometry**




**Scheme**



# Physical properties I

**See:** folder `\constant`

File **transportProperties**:

- Set phases:

```
phases (water air);
```

- Set density and kinematic viscosity for each phase:

```
water
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ] 1e-06;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1000;
}

air
{
    transportModel  Newtonian;
    nu              nu [ 0 2 -1 0 0 0 0 ] 1.48e-05;
    rho             rho [ 1 -3 0 0 0 0 0 ] 1;
}
```

# Physical properties II

- Set surface tension:

```
sigma           sigma [ 1 0 -2 0 0 0 0 ] 0.07;
```

File **g**: set the value and direction of gravity

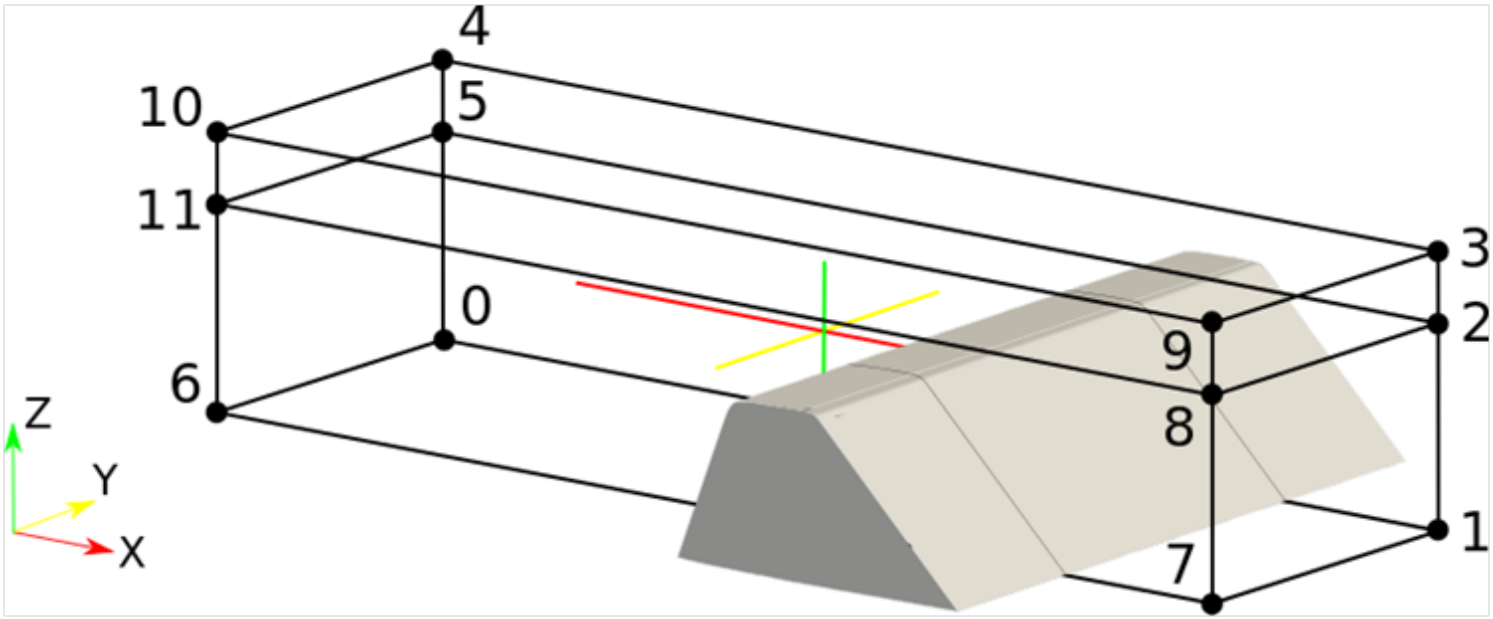
```
dimensions      [0 1 -2 0 0 0 0];
value           (0 0 -9.81);
```

File **turbulenceProperties**: set the turbulence model

```
simulationType  RAS;

RAS
{
    RASModel        kOmegaSST;

    turbulence      on;

    printCoeffs     on;
}
```

# blockMesh I

Create basic coarse mesh in the flow region and mark the boundaries.
**See**: `constant/polyMesh/blockMeshDict`
**Command**: `blockMesh`




**Exercise**: check results in Paraview.
You should see the rectangular region with the coarse mesh

# blockMesh II

- Set scale:

```
convertToMeters 1;
```

- Set vertices:

```
vertices
(
    (-20 -0.45 0 )
    ( 15 -0.45 0 )
    ( 15 -0.45 6 )
    ( 15 -0.45 8 )
    (-20 -0.45 8 )
    (-20 -0.45 6 )
    (-20 -0.55 0 )
    ( 15 -0.55 0 )
    ( 15 -0.55 6 )
    ( 15 -0.55 8 )
    (-20 -0.55 8 )
    (-20 -0.55 6 )
);
```

- Create two boxes:

```
blocks
(
    hex (0 1 2 5 6 7 8 11) (70 12 1) simpleGrading (1 1 1)
    hex (5 2 3 4 11 8 9 10) (70 4 1) simpleGrading (1 1 1)
);
```

# blockMesh III

- Define boundaries:

```
boundary
(
    inletAir
    {
        type patch;
        faces
        (
            (4 5 11 10)
        );
    }

    inletWater
    {
        type patch;
        faces
        (
            (5 0 6 11)
        );
    }
```

```
    outlet
    {
        type patch;
        faces
        (
            (1 2 8 7)
            (2 3 9 8)
        );
    }

    atmosphere
    {
        type patch;
        faces
        (
            (3 4 10 9)
        );
    }

    bottomWall
    {
        type wall;
        faces
        (
            (0 1 7 6)
        );
    }
```

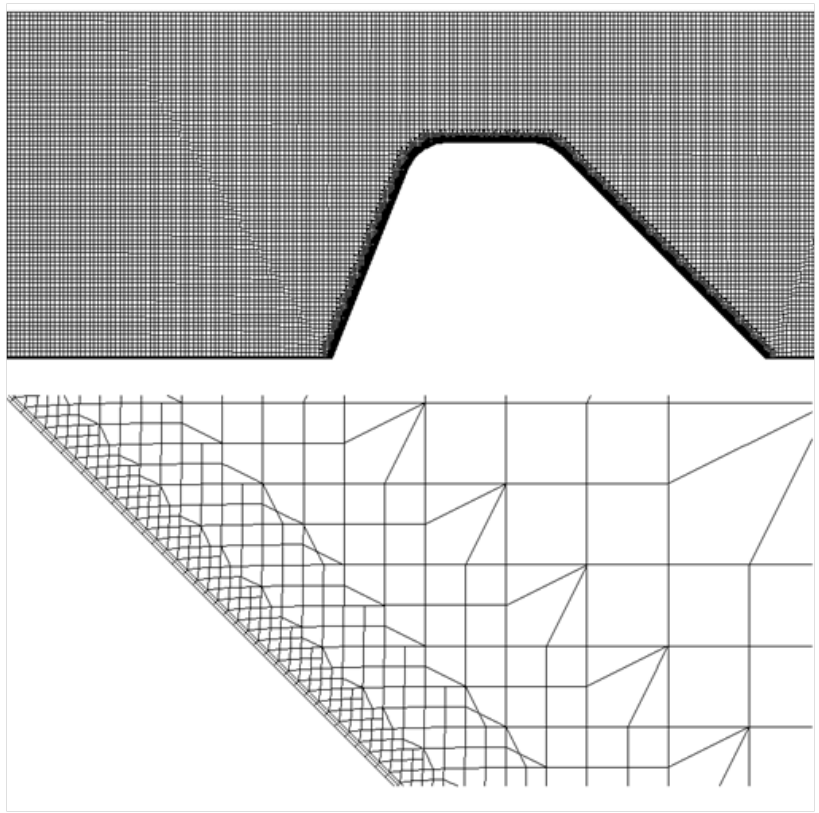
```
    front
    {
        type empty;
        faces
        (
            (1 0 5 2)
            (2 5 4 3)
        );
    }

    back
    {
        type empty;
        faces
        (
            (6 7 8 11)
            (11 8 9 10)
        );
    }

);
```

# snappyHexMesh I

**See**: `constant/system/snappyHexMeshDict`
**Command**: `snappyHexMesh -overwrite`

**Stage 1.** Refine mesh near the surface of the dam.
Three stages of meshing:

- refinement (see `CastellatedMeshControl` section);
- smoothing (see `SnapControls` section);
- set of layers (see `addLayersControls` section).



Use STL-surface to do it (see `constant/triSurface`):

```
dam.stl
{
    type triSurfaceMesh;
    name dam;
}
```

# snappyHexMesh II

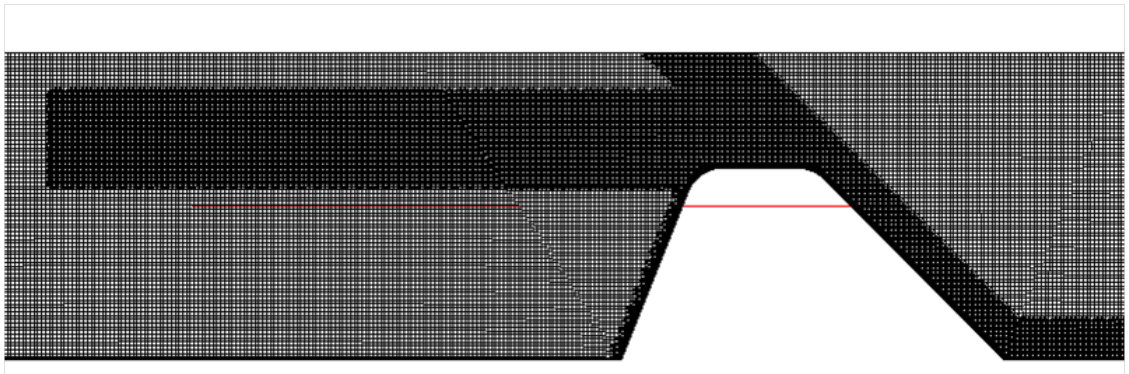
**Stage 2.** Add two refinement regions where the surface of water will flow. Use two boxes and plane to do it:

```
surface
{
    type searchableBox;
    min ( -15  -1  4.5 );
    max (   4   0  7   );
}
```

```
aroundDam
{
    type          searchablePlane;
    planeType     pointAndNormal;

    pointAndNormalDict
    {
        basePoint     ( 7.5 -0.5 2.5 );
        normalVector  ( 1    0   1   );
    };
}
```

```
outlet
{
    type searchableBox;
    min (  10  -1  0 );
    max (  15   0  1 );
}
```

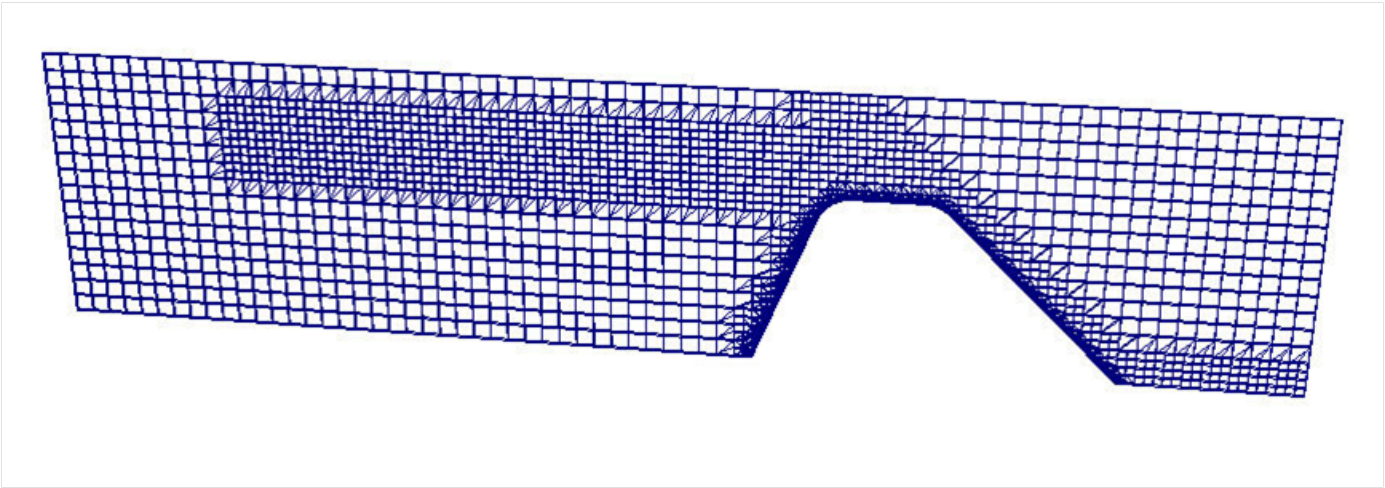
**Exercise**: try to run snappyHexMesh with different studies of remeshing. Watch differences.



# extrudeMesh

**See**: `constant/system/extrudeMeshDict`
**Command**: `extrudeMesh`

`blockMesh` and `snappyHexMesh` made 3D-mesh.
`extrudeMesh` creates a 2D-mesh from the surface (in our case — from patch).



**Exercise**: run checkMesh utility before running extrudeMesh and after it. Compare numbers of cells.

# Boundary conditions

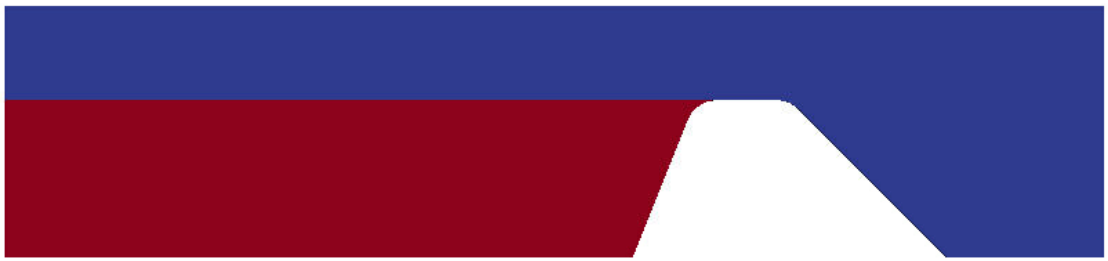
**See**: folder `0.org/`. Change needed files and copy it to folder `0/`.

| Name | $\alpha$ | $k$ | $\omega$ | $p^*$ | $\mathbf{U}$ |
|---|---|---|---|---|---|
| **inletAir** | fixedValue 0 | fixedValue 2.16e-4 | fixedValue 0.1470 | fixedFlux Pressure | fixedValue (0 0 0) |
| **inletWater** | fixedValue 1 | fixedValue 2.16e-4 | fixedValue 0.1470 | fixedFlux Pressure | fixedValue (0.6 0 0) |
| **outlet** | zero Gradient | zero Gradient | zero Gradient | fixedFlux Pressure | zero Gradient |
| **walls** | kqRWall Function | omegaWall Function | zero Gradient | fixedFlux Pressure | fixedValue (0 0 0) |
| **atmosphere** | inletOutlet | inletOutlet 2.16e-4 | inletOutlet 0.1470 | total Pressure | pressure Inlet Outlet Velocity |
| **front,back, defaultFaces** | empty | empty | empty | empty | empty |

# setFields

Set an initial distribution of fields (`alpha.water`) in regions.
Files in `0/` folder were modified.

```
defaultFieldValues
(
    volScalarFieldValue alpha.water 0
);

regions
(
    boxToCell
    {
        box (-20 -1 0) (3 1 5);
        fieldValues
        (
            volScalarFieldValue alpha.water 1
        );
    }
);
```



# Numerical schemes and time settings. Running simulation

See `system/controlDict` to create time settings:

- time interval,
- CFL number,
- write interval,
- time precision.

Settings for numerical schemes (use default settings): see `system/fvSchemes` and `system/fvSolution`.

Start application by `interFoam` command.

## Scripts

Sequence of all commands is placed into script file **./Allrun**.
Clean results: **./Allclean**.

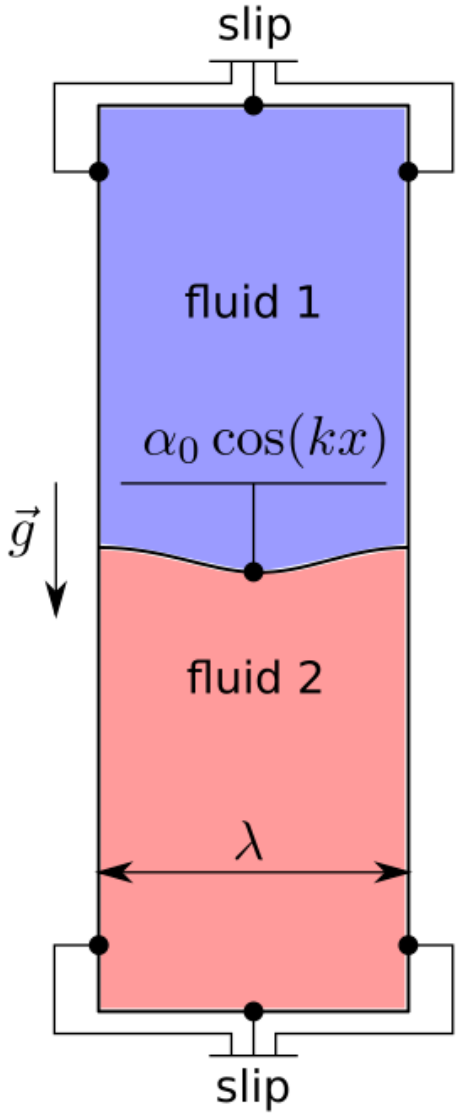
# Results

# Part II
# Practical part

---

Additional case: Rayleigh — Taylor instability

# Problem statement




## Input data

Fluid 1:
**density**: 1.255 kg/m$^3$;
**dynamic viscosity**: $3.13 \cdot 10^{-3}$ Pa·s.

Fluid 2:
**density**: 0.032 kg/m$^3$;
**dynamic viscosity**: $3.13 \cdot 10^{-3}$ Pa·s.

Surface tension coefficient: 0.01 N/m.

Interface form: $\alpha_0 = 0.05$ m, $k = 2\pi$.
Wave length: $\lambda = 1$ m.

# Linear theory

## Instability conditions

1. Initial perturbation: $\alpha > 0$, $\alpha \ll \lambda$.
2. Surface tension coefficient: $\sigma < \sigma_c, \sigma_c = \dfrac{\Delta\rho g}{k^2}$.
3. Dynamic viscosities: $\mu_1 = \mu_2 = \mu$.

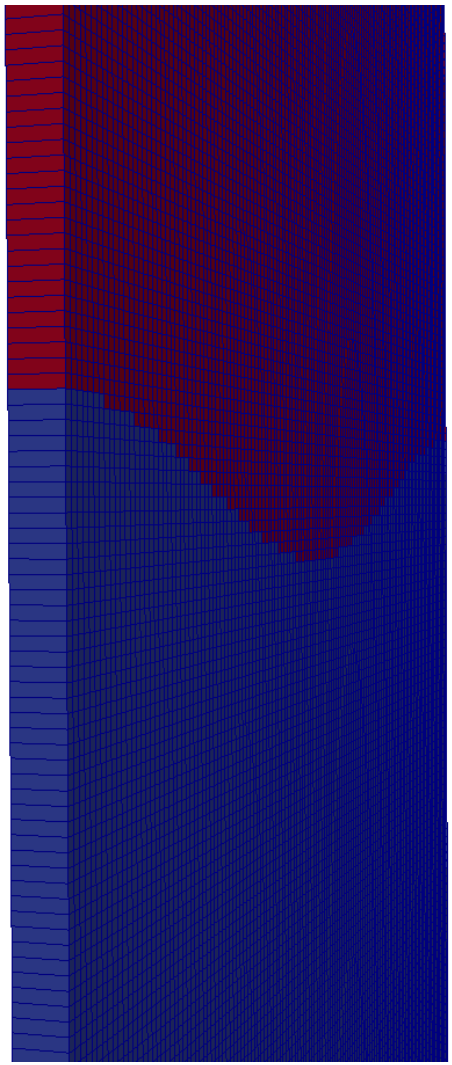
## Law for growth of amplitude instabitily

$$\alpha(t) = \alpha_0 \cosh(\Gamma t),$$

where $\Gamma$ — growth rate (1/s), $A$ — Atwood number (non-dimensional):

$$\Gamma = \sqrt{kg\left(A - \frac{k^2\sigma}{g(\rho_1 + \rho_2)}\right)}; \quad A = \frac{\rho_2 - \rho_1}{\rho_2 + \rho_1}.$$

# OpenFOAM case



## General settings

**Mesh**: one block, uniform mesh, no refinement.
**Boundary conditions**: `slip` for all variables.
**Transport properties**: set density and kinematic viscosity for two fluids.
**Turbulence properties**: laminar flow.
**Numerical settings**: standard `interFoam` settings.

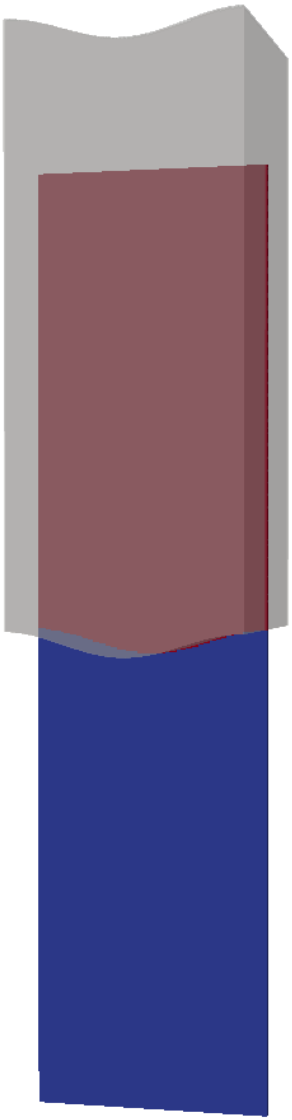
## How to set up the interface

Different ways to set up the initial perturbation:

1. use STL surface in standard `setFields` utility;
2. use `funkySetFields` utility from `swak4Foam` library.

# OpenFOAM case

## setFields

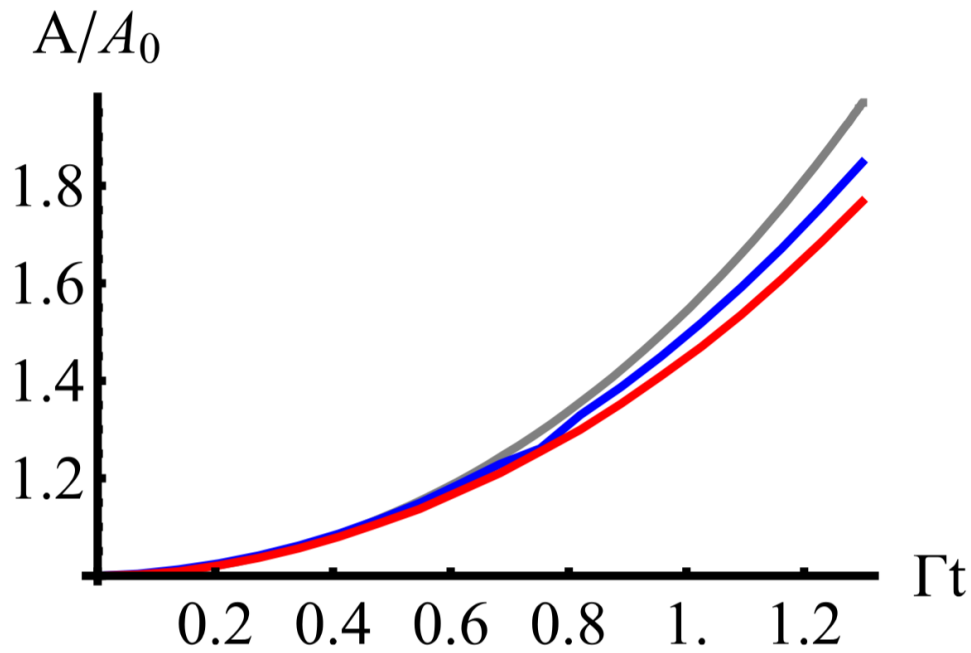


### Source code for STL in setFields

```
regions
(
    surfaceToCell
    {
        file            "liquid.stl";
        useSurfaceOrientation false;  // use closed surface inside/outside
                                      // test (ignores includeCut,
                                      // outsidePoints)
        outsidePoints   ((1e-4 0 -1.04));    // definition of outside
        includeCut      true;                // cells cut by surface
        includeInside   true;                // cells not on outside of surf
        includeOutside  false;               // cells on outside of surf
        nearDistance    -1;                  // cells with centre near surf
                                             // (set to -1 if not used)
        curvature       -100;                // cells within nearDistance
                                             // and near surf curvature
                                             // (set to -100 if not used)
        fieldValues
        (
            volScalarFieldValue alpha.liquid 1
            volVectorFieldValue U           (0 0 0)
        );
    }
);
```

STL surface should be placed in the case folder.

# Results




Gray color — linear theory

Blue color — Gerris[3]

Red color — OpenFOAM

[3] open-source code for free-surface flows, see **http://gfs.sourceforge.net/wiki/index.php/Main_Page**

# Summary

- We looked how `interFoam` works (look in the source code).
- We learned how to set boundary conditions for free-surface flows.
- We studied how to solve cases for free-surface flows step-by-step on the basic example — Spillway tutorial.
- We get the first experience in linear theory of hydrodynamic instabilities and run the additional case — RT instability

Let's talk about training track.
Questions?