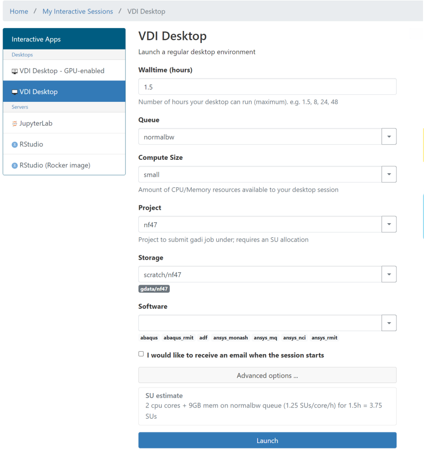
Regular waves interaction with a floating structure

Dr Darrin Stephens

Laminar2 Turbulent Pty Ltd

Initial steps

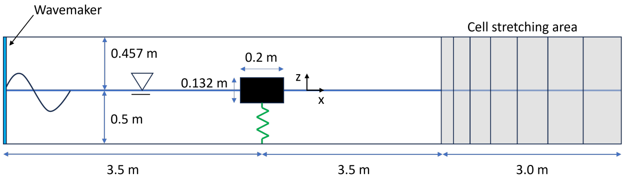
- Sign into ARE portal. https://are.nci.org.au/
- Launch a VDI Desktop session:
  - Walltime = 1.5 hours
  - Queue = normalbw
  - Compute Size = small
  - Project = nf47
  - Storage = scratch/nf47
- Copy files
  - cd /scratch/nf47/<username>
  - mkdir OpenFOAM_tut
  - cd OpenFOAM_tut
  - cp -R /scratch/nf47/HPCD-CFD-OpenFOAM/* .



Outline

The problem

- We will be simulating a floating object in a numerical wave tank.
- The object is tethered to the floor of the tank via a spring.
- A wave maker is used to generate regular Stokes II waves.
- For simplicity, the wave tank and object are treated as 2D.



Problem description

- Momentum and continuity equations:

$$\frac{\partial(\rho\,\mathbf{U})}{\partial t} + \nabla\cdot(\rho\,\mathbf{U}\,\mathbf{U}) = -\nabla p + \rho\,\mathbf{g} + \nabla\cdot(\mu\nabla\mathbf{U}) - \mathbf{F}_s$$

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

$\mathbf{g}$ is gravitational acceleration and $\mathbf{F}_s$ is surface tension force given by

$$\mathbf{F}_s = \sigma\kappa\,\mathbf{n}$$

$$n = \frac{\nabla\alpha_1}{|\nabla\alpha_1|}, \quad \kappa = \nabla\cdot\mathbf{n}$$

- A modified pressure field is used.

$$p_{rgh} = p - \rho\mathbf{g}\cdot\mathbf{x}$$

- The buoyancy terms on the right-hand side of the momentum equation can be expressed as

$$-\nabla p + \rho\mathbf{g} = -\nabla p_{rgh} - \mathbf{g}\cdot\mathbf{x}\nabla\rho$$

Problem description

- Volume of fluid in a cell is specified through the field α_1 through an expression:

$$\forall_f = \alpha_1 \forall_{cell}$$

- If the cell is completely filled with fluid, then $\alpha_1 = 1$.
- Vice versa, the cell is filled with the void if $\alpha_1 = 0$.
- Volume of fluid is defined through a transport equation:

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

Problem description

- The compression of the surface is achieved by using the artificial compression term:

$$\nabla \cdot (\alpha_1 (1 - \alpha_1) \mathbf{U}_r)$$

so that the VOF equation is

$$\frac{\partial \alpha_1}{\partial t} + \nabla \cdot (\alpha_1 \mathbf{U}) + \nabla \cdot (\alpha_1 (1 - \alpha_1) \mathbf{U}_r) = 0$$

- The density is computed from the linear interpolation

$$\rho = \alpha_1 \rho_l + (1 - \alpha_1) \rho_g$$

Running the tutorial

- We are using OpenFOAM v2212, so load the appropriate module.
- A script, `Allrun` has been provided that executes all the required steps. Use `-p` as an argument to the script for parallel execution.
- A PBS batch script is also provided that will submit to a queue. This script takes care of loading the module.
- Suggest submitting the simulation now so it runs while we walk through the simulation setup.
- Approximate running time is 15 minutes.

Case overview

```
Case Folder
├── 0.orig
│   ├── alpha.water
│   ├── pointDisplacement
│   ├── p_rgh
│   └── U
├── Allclean
├── Allrun
├── constant
│   ├── dynamicMeshDict
│   ├── g
│   ├── transportProperties
│   ├── turbulenceProperties
│   └── waveProperties
└── system
    ├── blockMeshDict
    ├── controlDict
    ├── decomposeParDict
    ├── fvSchemes
    ├── fvSolution
    ├── setFieldsDict
    └── topoSetDict
```

Meshing

- For meshing, we use `blockMesh`.
  Script:`runApplication blockMesh`
- `blockMesh` is a multi-block mesh generator.
- The mesh is generated from a dictionary file named `./system/blockMeshDict`.
- `blockMesh` generates high-quality meshes and it is the tool to use for very simple geometries.
- The effort and time required to set up the dictionary increases significantly as geometry complexity increases.

Meshing: blockMeshDict

- To generate a mesh with `blockMesh`, the vertices of each block, block connectivity and number of cells in each direction need to be defined.

```
vertices
(
    (-3.5 -0.1 -0.5)
    ( 6.5 -0.1 -0.5)
    ( 6.5  0.1 -0.5)
    (-3.5  0.1 -0.5)
    (-3.5 -0.1  0.457)
    ( 6.5 -0.1  0.457)
    ( 6.5  0.1  0.457)
    (-3.5  0.1  0.457)
);
blocks
(
  hex (0 1 2 3 4 5 6 7) (746 1 87)
  simpleGrading
  (
    (
      (7 700 1)
      (3 46 20)
    )
    1
    1
  )
);
```

Meshing: blockMeshDict cont'd

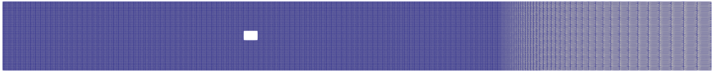
- Faces are allocated to boundary patches.

```
boundary
(
  stationaryWalls{
    type wall;
    faces ((0 3 2 1));
  }
  inlet{
    type patch;
    faces ((0 4 7 3));
  }
  outlet{
    type patch;
    faces ((2 6 5 1));
  }
  atmosphere{
    type patch;
    faces ((4 5 6 7));
  }
  sides{
    type empty;
    faces ((1 5 4 0) (3 7 6 2));
  }
  floatingObject{
    type wall;
    faces ();
  }
);
```

- The region of the mesh that represents the floating object is marked using `topoSet`.
  Script: `runApplication topoSet`
- `topoSet` is controlled with a dictionary file named `./system/topoSetDict`.
- The floating object is subtracted from the mesh using `subsetMesh`.
  Script: `runApplication subsetMesh -overwrite c0 -patch floatingObject`

Meshing: topoSetDict


```
actions
(
    {
        name    c0;
        type    cellSet;
        action  new;
        source  boxToCell;
        sourceInfo
        {
            box (-0.1 -0.1 -0.0786) (0.1 0.1 0.0534);
        }
    }
    {
        name    c0;
        type    cellSet;
        action  invert;
    }
);
```



Fluid properties

- Fluid properties are specified in the file `./constant/transportProperties`:

```
phases (water air);
water
{
    transportModel  Newtonian;
    nu              1e-06;
    rho             1000;
}
air
{
    transportModel  Newtonian;
    nu              1.48e-05;
    rho             1;
}
sigma           0;
```

Gravitational constant and simulation type

- The gravitational constant is specified in the file `./constant/g`:

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

- The selection of the simulation type is specified in the file `./constant/turbulenceProperties`:

```
simulationType  laminar;
```

Wave Model

- The wave conditions are input via the file `./constant/waveProperties`.
- We will be using regular waves with a Stokes II formulation.
- Wave height 0.12 m.
- Wave period 2.0 s.
- The wave inlet condition is ramped from flat water over a 2-second duration.
- The inlet patch uses one paddle.
- Wave absorption is enabled at the inlet to absorb any reflections.
- Absorption at the outlet uses shallow water theory.

Wave Model: WaveProperties

```
inlet
{
    alpha            alpha.water;
    waveModel        StokesII;
    nPaddle          1;
    waveHeight       0.12;
    waveAngle        0.0;
    rampTime         2.0;
    activeAbsorption yes;
    wavePeriod       2.0;
}

outlet
{
    alpha            alpha.water;
    waveModel        shallowWaterAbsorption;
    nPaddle          1;
}
```

Dynamic Mesh Model

- Deformation and morphing of the mesh are defined in the dictionary `./constant/dynamicMeshDict`.
- The first line of this dictionary selects the class to handle the mesh motion and topology changes.

```
dynamicFvMesh dynamicMotionSolverFvMesh;
```

- Then we import the library containing the motion solver.

```
motionSolverLibs ("libsixDoFRigidBodyMotion.so");
```

- Selection of the motion solver.

```
motionSolver sixDoFRigidBodyMotion;
```

Dynamic Mesh Model: dynamicMeshDict

- Definition of the floating object and the zone within the mesh moves as a rigid body, the zone where the mesh is morphed and the zone with no morphing.
- Ridgid body properties: mass, moment of inertia.
- Solver selection, `Newmark` 2nd order time-integrator.

```
sixDoFRigidBodyMotionCoeffs
{
    patches          (floatingObject);
    innerDistance    0.05;
    outerDistance    0.35;
    centreOfMass     (0.0 0.0 -0.016);
    mass            3.148;
    momentOfInertia ( 0.015 0.015 0.021 );
    report           on;
    accelerationRelaxation 0.7;
    solver
    {
        type Newmark;
    }
}
```

Dynamic Mesh Model: dynamicMeshDict

- Definition of constraints on the motion of the floating object.
- Definition of restraints on the motion of the floating object.

```
sixDoFRigidBodyMotionCoeffs
{
    constraints
    {
        fixedAxis
        {
            sixDoFRigidBodyMotionConstraint axis;
            axis  (0 1 0);
        }
    }
    restraints
    {
        verticalSpring
        {
            sixDoFRigidBodyMotionRestraint  linearSpring;
            anchor                          (0.0 0.0 -0.5);
            refAttachmentPt                 (0.0 0.0 -0.082);
            stiffness                       150;
            damping                         0;
            restLength                      0.4;
            gravityVector                   (0 0 -9.81);
            thickness                       0.025;
        }
    }
}
```

```
dimensions      [0 1 -1 0 0 0 0];
internalField   uniform (0 0 0)
boundaryField
{
  stationaryWalls
  {
    type             noSlip;
  }
  inlet
  {
    type             waveVelocity;
    value            uniform (0 0 0);
  }
  outlet
  {
    type             waveVelocity;
    value            uniform (0 0 0);
  }
  atmosphere
  {
    type             pressureInletOutletVelocity;
    value            uniform (0 0 0);
  }
  sides
  {
    type             empty;
  }
  floatingObject
  {
    type             movingWallVelocity;
    value            uniform (0 0 0);
  }
```

Boundary conditions: Buoyant pressure (p_{rgh})

```
dimensions      [1 -1 -2 0 0 0 0];
internalField   uniform 0;
boundaryField
{
  stationaryWalls
  {
    type            fixedFluxPressure;
  }
  inlet
  {
    type            fixedFluxPressure;
  }
  outlet
  {
    type            fixedFluxPressure;
  }
  atmosphere
  {
    type            totalPressure;
    p0              uniform 0;
    value           uniform 0;
  }
  sides
  {
    type            empty;
  }
  floatingObject
  {
    type            fixedFluxPressure;
  }
}
```

Boundary conditions: pointDispacement


```
dimensions      [0 1 0 0 0 0 0];
internalField   uniform (0 0 0);
boundaryField
{
    stationaryWalls
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    inlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    outlet
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    atmosphere
    {
        type            fixedValue;
        value           uniform (0 0 0);
    }
    sides
    {
        type            empty;
    }
    floatingObject
    {
        type            calculated;
    }
```

Boundary conditions: Volume fraction (α_1)


```
dimensions      [0 0 0 0 0 0 0];
internalField   uniform 0;
boundaryField
{
    stationaryWalls
    {
        type            zeroGradient;
    }
    inlet
    {
        type            waveAlpha;
        value           uniform 0;
    }
    outlet
    {
        type            zeroGradient;
    }
    atmosphere
    {
        type            inletOutlet;
        inletValue      uniform 0;
        value           uniform 0;
    }
    sides
    {
        type            empty;
    }
    floatingObject
    {
        type            zeroGradient;
    }
}
```

Initial conditions: Volume fraction (α_1)

- It is very important to set the initial conditions to prescribe the initial state of the water.
- In order to initialise the volume fraction field to simulate a flat free surface at the time $t = 0$, The `setFields` utility is used.
- The `setFields` utility is responsible for the spatial distribution of the α_1 field by setting the desired value at the particular locations.
- For the water part of the domain, the value of the α_1 field is set to 1 and everywhere else is set to 0.
- `setFields` utility has the ability to specify desired regions in the form of boxes specified with two opposite nodes.

Initial conditions: Volume fraction (α_1)

- The `setFields` utility is control by the `./system/setFieldsDict` dictionary.
- `setFieldsDict` entries:

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

regions
(
    boxToCell
    {
        box (-100 -100 -100) (100 100 0.0);
        fieldValues ( volScalarFieldValue alpha.water 1 );
    }
);
```

- Script: `runApplication setFields`

Numerical settings: fvSchemes

- Numerical schemes for the different terms are contained in the dictionary `./system/fvSchemes`.

```
ddtSchemes
{
    default          CrankNicolson 0.9;
}
gradSchemes
{
    default          Gauss linear;
    grad(U)          cellLimited Gauss linear 1;
}
divSchemes
{
    div(rhoPhi,U)    Gauss vanLeerV;
    div(phi,alpha)   Gauss vanLeer;
    div(phirb,alpha) Gauss linear;
    div(((rho*nuEff)*dev2(T(grad(U))))) Gauss linear;
}
laplacianSchemes
{
    default          Gauss linear corrected;
}
interpolationSchemes
{
    default          linear;
}
snGradSchemes
{
    default          corrected;
}
```

Numerical settings: fvSolution

- interFoam solver uses volume fraction specific solver settings:

```
solvers
{
    "alpha.water.*"
    {
        nAlphaCorr          2;
        nAlphaSubCycles     1;
        cAlpha              1;
        MULESCorr           yes;
        nLimiterIter        5;
        alphaApplyPrevCorr  yes;
        solver              smoothSolver;
        smoother            symGaussSeidel;
        tolerance           1e-8;
        relTol              0;
}
```

- nAlphaSubCycles: Number of sub-cycles over the volume fraction equation in a given time step.
- cAlpha: Level of compression.
- MULESCorr implicit MULES

Numerical settings: fvSolution

- `interFoam` solver uses the PIMPLE algorithm:

```
PIMPLE
{
    momentumPredictor         no;
    nOuterCorrectors          1;
    nCorrectors               3;
    nNonOrthogonalCorrectors 1;
    correctPhi                yes;
}
```

- `momentumPredictor`: Don't solve the momentum equation. Just use it as a linearised predictor.
- `nOuterCorrectors`: PISO mode.

Solving: controlDict

- Simulation parameters are defined in the dictionary `./system/controlDict`.

```
application       interFoam;
startFrom         latestTime;
startTime         0;
stopAt            endTime;
endTime           8;
deltaT            0.001;
writeControl      adjustableRunTime;
writeInterval     0.05;
purgeWrite        0;
writeFormat       binary;
writePrecision    6;
writeCompression  no;
timeFormat        general;
timePrecision     6;
runTimeModifiable no;
adjustTimeStep    yes;
maxCo             0.5;
maxAlphaCo        0.5;
maxDeltaT         0.01;
```

Solving: Execution

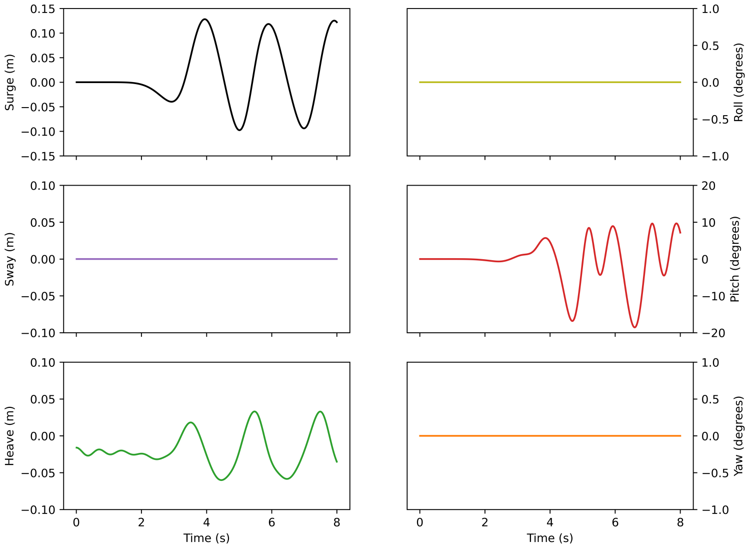
- To run in parallel, we first decompose the mesh into a number of pieces matching the number of processors.
  Script: `runApplication decomposePar`
- Execution of the solver.
  Script: `runParallel interFoam`

Post-processing: Function Objects

- Some post-processing can be done within the solver at run time using function objects.
- These are found at the bottom of the `controlDict`

```
functions
{
  sixDoF_History
  {
    type          sixDoFRigidBodyState;
    libs          ("libsixDoFRigidBodyState.so");
    angleFormat   degrees;
    writeControl  timeStep;
    writeInterval 5;
  }
}
```

Post-processing: SixDof history



Post-processing: ParaView

- Visual post-processing and be performed using ParaView
- You will need to load the module to use ParaView.