6-DOF VOF-solver without Damping in OpenFOAM

Erik Ekedahl

February 15, 2009

Abstract

Implementation of a Volume of Fluid solver coupled with mesh motion in six degrees of freedom. A set up with a cubical box submerged in a calm surface was studied in order to determine the stability of the scheme. A description of the algorithm and the code as well as a disscusion on the result and recommendations for further improvements are included.

Contents

1 Introduction

In many engineering problems it is interesting to study the fluid-solid interaction. Considering marine applications where for example a ship is moving at a given velocity. This sort of simulation will already, without mesh motion, cause the surface to break and spray close to the hull. Adding mesh motion to this not only requires more computer power but also adds more complexity to the surface reconstructions. Resolving these issues give way to studies of ship induced waves and more.

This chapter covers the implementation of a six-DoF solver to couple with *interDyMFoam*. Here a floating box is considered as a simplification of more complex structures. The focus of this first study of a Volume of Fluid solver coupled with mesh motion in six degrees of freedom is to present some simple basic cases.

Case A : leaving the water surface completely undisturbed and leaving the box to float. Here the water surface and the horisontal plane cutting through the center of gravity are aligned.

Case B : a wave is used to set the box moving.

Case C : same set up as in case B but with a finer mesh.

2 Theoretical preamble

2.1 Volume of Fluid Method

The Volume of Fluid method (VoF) is a two-phase surface compression method that solves the Navier-Stokes equations. In the VoF-method the two phases are considered as a single phase with a volume fraction between 0 and 1, as described in (**??**).

The interface is not a sharp surface but rather a region where further refinements are made than in the regions with either $\gamma = 0$ or $\gamma = 1$, this makes the method efficient. The velocity field is estimated from the previous values and a PISO algorithm is used to solve an extra transport equation for the volume fraction after which the velocity field is updated. [gui()]

2.2 Under-relaxation

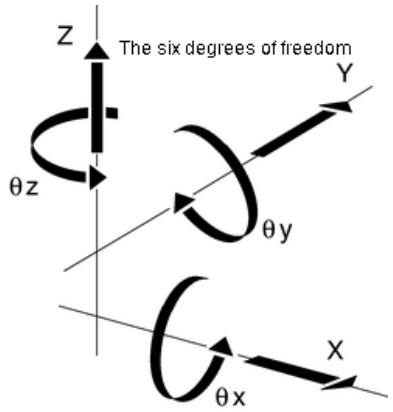
Under-relaxation is an efficient way to stabilize numerical schemes and improving convergence. By controlling to what extent the solver uses the values previous calculated to determine the new ones. An under relaxation factor of 0.7 can be described mathematically, let α be the under-relaxation factor, then

$$\phi_{new} = \alpha\phi_{new} + (1 - \alpha)\phi_{old} \tag{1}$$

where ϕ_{old} is the old value and ϕ_{new} is the new value just calculated in the solver. [R.M. Barron(2003)]

2.3 Degrees of Freedom, Forces and Moments

The term six degrees of freedom (6-DoF) refers to how the movement of a body is limited, having all six degrees of freedom means that the body can translate and rotate along all three axises in a three dimensional system. Figure 1 shows the possible translations and rotations in three dimensional space. When solving the Navier-Stokes Equations and the



ls

Figure 1: Axises describing the 6 degrees of freedom

continuity equation for a flow it results in forces and these can be calculated by integrating over the body, as in equations (2) and (3)

$$\boldsymbol{F} = \int_S \boldsymbol{F}_{ext} + \boldsymbol{F}_{flow} \tag{2}$$

$$\boldsymbol{M} = \int_S \boldsymbol{M}_{ext} + \boldsymbol{M}_{flow} \tag{3}$$

Where $\boldsymbol{F}$ is the total force, here the buoyancy force or pressure force and the viscous forces and $\boldsymbol{M}$ is the total moment around the mass center due to pressure and viscous forces. $\boldsymbol{F}_{ext}$ is here only the gravity since no other external forces are considered. [Roozbeh Panahi(2006)]

2.4 Rotation and Translation

From the forces, extracted as above, the velocity and the displacement distance for one time step is calculated. It is here necessary to use the accumulated distances. This is because each time step has a mesh updating function that resets the mesh, though the calculations with flux and other quantities are made according to the moved mesh. Using two objects, quaternion and septernion that are objects that can store a translation or rotation and when calling return a tensor for rotation or a vector for translation. An object of the class septernion consists of a vector and a quaternion. The theoretical focus here will be on the quaternion which is the object handling the rotation.

A quaternion is a mathematical object, a vector in four-dimensional vector space, written as a linear combination of the basis elements

$\{1, i, j, k\}$ as $a1 + bi + cj + dk$, where a, b,c and d are real numbers and $i^2 = j^2 = k^2 = ijk = -1$ as found by Hamilton. Here a is called the scalar part and $bi + cj + dk$ is called the vector part. The vector part of the quaternion can be identified to be the same as an element of R^3 vector space.

In OpenFOAM an object of the *quaternion* class consists of a vector and a scalar - the angle of rotation. A *quaternion* can be constructed in different ways, in this work is has been done by specifying the three Euler angles. This means explicitly that one creates three *quaternions*, one for each angle in the three directions and multiply them together as $q_x * q_y * q_z$. The conversion between the Euler angles and the *quaternion* is described mathematically as follows

$$\begin{aligned} w = c_1c_2c_3 - s_1s_2s_3 \quad v_x = s_1s_2c_3 + c_1c_2s_3 \\ v_y = s_1c_2c_3 + c_1s_2s_3 \quad v_z = c_1s_2c_3 - s_1c_2s_3 \\ c_1 = \cos(rot_y) \quad c_2 = \cos(rot_z) \quad c_3 = \cos(rot_x) \\ s_1 = \sin(rot_y) \quad s_2 = \sin(rot_z) \quad s_3 = \sin(rot_x) \end{aligned}$$

where the w is the resulting angle for the *quaternion* and v_i the vector and rot_i is the rotation around that axis, $i = x, y, z$. The conversion from a *quaternion* back to a rotation vector in R^3 is described below.

$$\begin{aligned} w_2 = \sqrt{w} \quad x_2 = \sqrt{v}_x \\ y_2 = \sqrt{v}_y \quad z_2 = \sqrt{v}_z \\ t_{xy} = 2 * v_x * v_y \quad t_{wz} = 2 * w * v_z \\ t_{xz} = 2 * v_x * v_z \quad t_{wy} = 2 * w * v_y \\ t_{yz} = 2 * v_y * v_z \quad t_{wx} = 2 * w * v_x \end{aligned}$$

$$\begin{pmatrix} w_2 + x_2 - y_2 - z_2 & t_{xy} - t_{wz} & t_{xz} + t_{wy} \\ t_{xy} + t_{wz} & w_2 - x_2 + y_2 - z_2 & t_{yz} - t_{wx} \\ t_{xz} - t_{wy} & t_{yz} + t_{wx} & w_2 - x_2 - y_2 + z_2 \end{pmatrix}$$

These two mathematical maps, first from R^3 to the four dimensional vector space of quaternions and then in the opposite direction describe how the implementation is made in OpenFOAM. It can be added to the *FOAMAPP/test/quaternion/quaternionTest.C* with the following lines:

```
//Euler to quaternion:
quaternion q(angle_x,angle_y,angle_z);
//Quaternion to rotation
tensor rotation = q.R();
```

3 Moving Mesh and Diffusion

In a larger number of application it is interesting to examine the interaction between a solid and one or more fluids. To accomplish this it is necessary to move parts of the mesh and for that purpose there are a number of methods, see [Hrvoje Jasak(2007)]. The one here considered is *mesh deformation*, where the cells in the mesh are deformed (stretched or compressed) due to the motion of a part of the mesh.

3.1 Mesh Deformation

Mesh deformation can mathematically be considered as a map between the domain D, which represents the configuration at a certain time t with a boundary B, and D' Δt times later. Now it would only be interesting to consider a very small subset of all the possible maps that fulfill this requirement, namely the ones that will result in a valid mesh in domain D' [Hrvoje Jasak(2007)]. Consider figure 2 which shows the mesh in the xy-pane at the initial set up. Already here the mesh shows some signs of low quality. This is compared to the mesh in figure 3 which shows how many cells have been skewed severely.

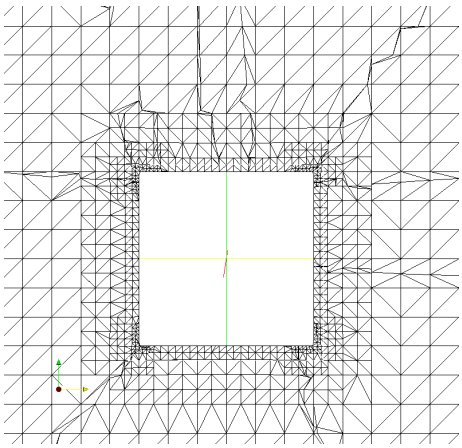


Figure 2: the xy-plane at time 0

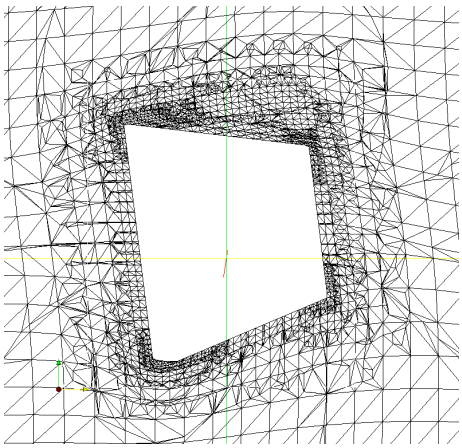


Figure 3: the xy-plane at time 0

4 Code

The idea is to move the points on the box and the cells neighbouring it by describing them explicitly to the *pointDisplacement* and *cellDisplacement* fields, using the ==-operator at each timestep of the solver. This is implemented in the file *krafter.H*. The ==-operator assigns a field of vectors describing the accumulated displacement up to that time step. This means that the diffusion algorithms described in [Moradnia(2007)] in the mesh can be used. This is important as it is not possible to move only a few points in the mesh without destroying the shape of the cells, the neighbouring points needs to be moved as well, this is done by the *motionFvSolvers* by a diffusion algorithm.

The *mesh.update()* that follows right after the implementation of the movements resets the mesh and it is therefore crucial to assign the accumulated displacement at each step. This is because nothing is prescribed to the *cellDisplacement* and *pointDisplacement* dictionaries in the Time-directory and the manipulation is done explicitly during each time step.

4.1 Algorithm

The following algorithm describes the extraction of the forces and the implmentation of the mesh moving scheme.

while Time is running **do**
  Extract forces and moments from pressure, buoyancy.
  Extract forces and moments from viscous effects.
  Divide by mass for forces and inertia for momentum.
  Extract the acceleration by multiplication of the real time interval.
  Reset and move center of gravity.
  Translate patch to origo.
  Rotate in accordance to momentum.
  Translate back to origo.
  Translate according to forces.
end while

4.2 Implementation

There are four files added to the original version of *interDyMFoam* in OpenFOAM-1.5: *initializeForceBalance.H* which initializes the variables and reads the motion fields, *readForceBalanceControls.H* which reads from the *forceFoamDictionary*, *krafter.H* which calculates the forces and moments and moves the mesh and finally *pEqnGravity.H* which reconstructs the pressure. In the file *krafter.H* one finds the implementation of the forces and moments and the extraction of the displacement by translation and rotation and finally the assignment of these displacements to the mesh.

The forces are calculated in *krafter.H* as follows: the buoyancy force from the pressure is summed up for the patch and then multiplied by the area vectors of the patch resulting a pressure force with a given direction, line 63 - line 83 . For the moments the center of gravity *CgCenter*, is subtracted from the position of the points on the patch so that the moment is about the center of gravity rather then just around origo, if the patch is not situated there, line 89 - line 109. The gravity force is the product of the mass and the vector representing the magnitude and direction of the gravity acceleration, line 112. Note that the patch is built from faces which are built from points, cells concerned are the ones that are neighbouring the patch and has one face that is a boundary face of the same. The *Sf()*-function in the extraction of the pressure forces returns the area vectors for the faces, $Cf()$ returns the face centers. The viscous force and moment are extracted from the velocity field where *magSf()* is used to bring out cell face area. These three functions are called from fvMesh which is the base class for the dynamicFvMesh which is the type of mesh used in simulations with *interDyMFoam*. The forces and moments are added toghter and later called *mometStatic* and *forceStatic* in krafter.H. From these the accumulated displacement and angle is calculated.

At the initialization the original positions of the points and cells are extracted and assigned to the fields *tempPoint* and *tempCell*, line 167 in *initializeForceBalance.H*. First copying these to temporary fields on

which the rotation and transformation will be performed, line 203 - line 226 The temporary field is the translated to the origo by subtraction with the center of gravity and using the conversion from quaternion to a rotational matrix given in (4). This produces a new temporary Field. The field is then translated back to its original position. This field is then translated by adding a vector to its position and then the original field is subtracted from the one obtained after translation to extract the displacement. Last the center of gravity is translated to its new position

4.3 Dictionaries

Needed apart from the directories used by *interDyMFoam* are the /constant/forceFoamDict and /0/cellDisplacement and /0/pointDisplacement dictionaries. The *forceFoamDict* is appended below and will be explained hereafter.

The *forceBalanceEnabled* is a Boolean that will turn the forces and movements on or off. M is the mass of the patch, this has to be calculated by hand at the moment, the same holds for *inertia* which is the moments of inertia for the patch. Here *accelLim* is a scalar that is used to limit the movement. The acceleration due to gravity g is set by a vector and a scalar. The center of gravity is assigned in the form of a position vector as *CgCenter*. At last the name of the patch which should be possible to move is defined.

```
forceBalance
{
 forceBalanceEnabled    yes;
}

/* ------------------------------------------------- */

// Mass of the moving part
M             M [1 0 0 0 0 0 0] 108000;

// Mass of the moving part
inertia       inertia [1 2 0 0 0 0 0] 648000;

// Limit the maximum acceleration on
// the system (Bandwidth limitation)
accelLim      accelLim [0 1 -2 0 0 0 0] 1.0e-2;

// Acceleration due to gravity
g             g [0 1 -2 0 0 0 0] 9.81;

// Direction vector of the acceleration due to gravity
gVector       (0 0 -1);

// Direction vector of the acceleration due to gravity
CgCenter (0 0 0);
```

```
/* ------------------------------------------------------- */

motionPatches
(
 cube_region0
);
```

All the the entries in point- and cellDiscplacement was set to have the Dirichlet boundary condition by

```
type fixedValue;
value uniform (0 0 0);
```

5 Compile and Use

The solver and all the required files to compile the solver in 1.5 or 1.5.x is included, extract the archive by:

```
tar -xzvf my6DOFFOAM.tar.gz
```

```
cd my6DOFFoam
```

wmake the solver

```
wmake
```

The case folder consists of a complete set up for a floating cubical box, extract the caseby:

```
tar -xzvf kubtest.tar.gz
```

run the case with the command:

```
my6DOFFoam
```

To change the mesh, extract the mesh archive and edit the *blockMeshDict*, run

```
blockMesh
```

and edit the *system/snappyHexMeshDict*. run

```
snappyHexMesh
```

Copy the files from the /2-directory in the mesh folder to the constant folder in the case folder.

5.1 Possible Alterations

There are a few parameters that are known to have an impact on the performance of the system. First, different settings with the initial configuration of the water can be examined. In *system/setFieldsDict* there are two *boxToCell* regions and the latter of the two can be changed by changing the size of its bounding box. This will influence directly how the box moves due to external forces and moments. In *constant/forceFoamDict*: the gravity will not influence the rotation, changing the weight of the box will define how much the box is submerged. Secondly, the *accelLim* can be set to limit the displacement. The third parameter which can be changed is the under-relaxation factor, see (1) which is found in the *system/fvSolvers* file at the very end. Adding to these possible alterations the mesh. The mesh is appended and can be refined or altered in different ways. First Using a finer mesh will make it the simulation run slower and it will also be more assailable to displacement. In the mesh folder in *system/snappyHexMeshDict* the parameters for the *snappyHexMesh* is set. Setting the level for the surface refinement differently will change the mesh structure close to the box.

6 Results

6.1 Mesh

A mesh was generated in *snappyHexMesh*, first using *blockMesh* to create a background mesh with dimensions 40x40x40 meters and *snappyHexMesh* placed a box as a searchablesurface with the dimension 6x6x6meters in the center of the domain, depicted in figure 4. The *snappyHexMesh* utility created the box without any refinement box in order to keep the cell size fairly large. Since a small cell will suffer more from skewness given a certain displacement, a larger cell will be better of. Case A,B:

```
Number of cells of each type:
    hexahedra:     85552
    polyhedra:     5679
```

Case C:

```
Number of cells of each type:
    hexahedra:     510272
    polyhedra:     0
```

Using a VOF-methods it is important to have hexagonal cells for the quality of the solution. The polyhedral cells are boundary cells and thus closer to the box which causes the iso-surface to look strange in a region close to the box. How much this actually affects the quality of the solution is not clear.

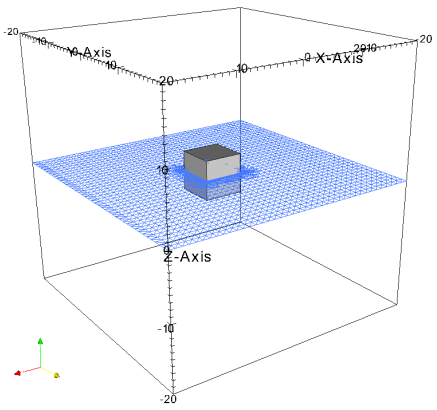


Figure 4: The domain and the surface of the water.The water fills the domain from -20m to 0m along the z-axis

6.2 Boundary Conditions

For all the patches in p, pd and gamma the *zeroGradient* of Dirichlet boundary condition was assigned. This is because all the patches are considered solid walls. For U the box-patch was assigned the *movingWallVelocity*-Boundary Condition which accounts for the movement of the fluid as well as for the movement of the box-patch to make sure that these there is no flux into the box. If not using this boundary condition, the solver will interpret this as if the wall is not moving and let the flux pass to positions where there actually is a box. All the other patches in U are given the boundary condition slip, which when the base patch is *patch* works as free slip with no flow in the direction normal to the patch this will also reflect incoming waves. It can be added that if using the *wall* base patch, the cells next to the wall will be considered as wall as well, which can have effects on the flow. The patches in the two new files added to use the solver, cellDisplacement and pointDisplacement are all assigned *fixedValue* and *uniform (0 0 0)*, this is also called the Neumann boundary condition.

6.3 Initial Values

In order to create some movements in the fluids an extra region of water was placed on top of the one covering half the domain, see figure 10. This was done by setting the second region in *system/setFiledsDict* to raise a bit over the first over some part of the domain. The height of this region will determine how box moves due to the forces the water act on the box. The box is at time zero submerged to fifty percent in the water with its vertical sides orthogonal to the water surface. The under-relaxation factors were set to 0.7 for U and 0.5 for p and gamma.

6.4 Case A: A Calm Surface

Considering a box floating on perfectly still water there should be no reason for the box to flip over. The only force applied to the box is the gravity force which is homogeneous over the box. The horisontal plane through the center of gravity is aligned with the water surface. In the simulation the box was place in the water surface and the simulation ran for 16 seconds until the solver diverged due to cell deformation. The box had flipped over. In figure 5 the initial set up is pictured and as can be seen the surface is perfectly calm. The solver has a tendency to, due to numerical discrepancies and possibly a low mesh quality, introduce disturbances. In figure 6 the solver starts moving the box and in figure 7 it diverges. The reason for this behavior could be linked to the mesh quality and the stability of the solver. This motivates some sort of damping further than the one constructed for the acceleration or another approach than mesh deformation.

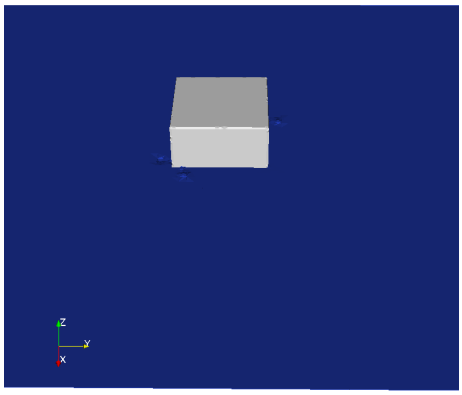


Figure 5: Case A: initial condition, non distrubed surface.

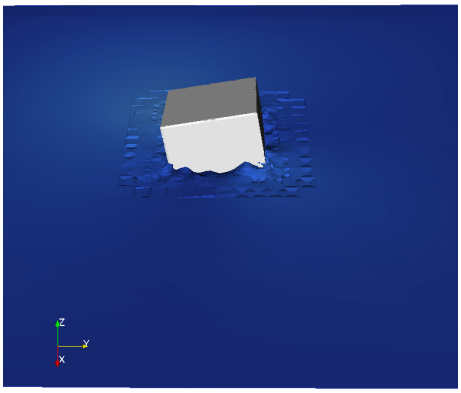


Figure 6: Case A: after 8 seconds, the surface is disturbed from numerical discrepancies.

The pressure is visualized at the first 8 and at the last timestep 9. Here a yz-plane through the cube with pressure glyphs shows the pressure distribution. also the line through the cube at $y = 0$ is the surface of the water.

6.5 Case B: A Coarse mesh

When setting the initial value to be a static wave or rather three meter high region covering part of the domain as can be seen in figure 10 the box is behaving as one could expect. It remains symmetric when the wave passes, figure 11. In figure 12 one can see the box tilt due to the forces from the water and start to tip over on the side in figure 13. Further in figure 14 and in figure 15 the box starts rotating around the

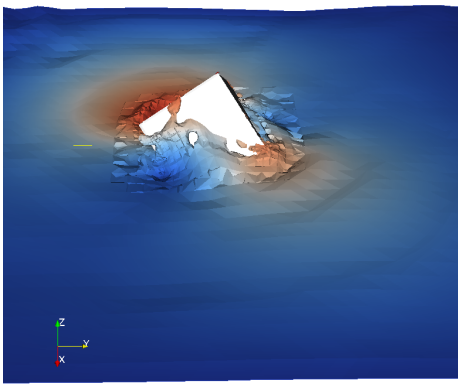


Figure 7: Case A: the box and surface just before divergence, the numerical scheme and the mesh is not accurate enough.

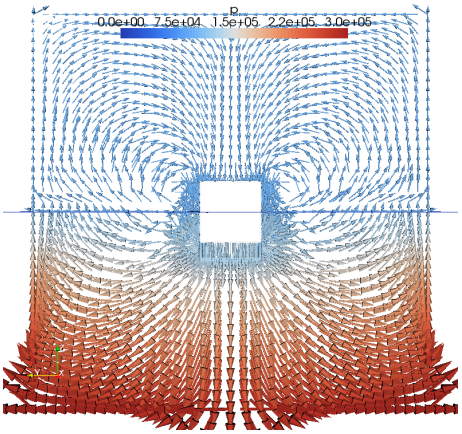


Figure 8: Case A: Slice in the yz-plane with pressure glyphs at the first time step

Z-axis and tilts that much that the solver diverges due to skewed cells, figure 22.

6.6 Case C: A Fine mesh

A mesh with cells, one fourth of the volume of the original ones, that means cells with 0.25 m length were used for This simulation. The result is similar but the resolution was better on the iso-surface. A region with a height of three meters covering the the part of the domain next to the box was set as the initial condition, see figure 18. The box behaves in much the same way as in the case with fewer cells. In figure 19 the wave has passed the box and just hit the opposite wall where is reflects and in figure 20 it returns from the left again and hits

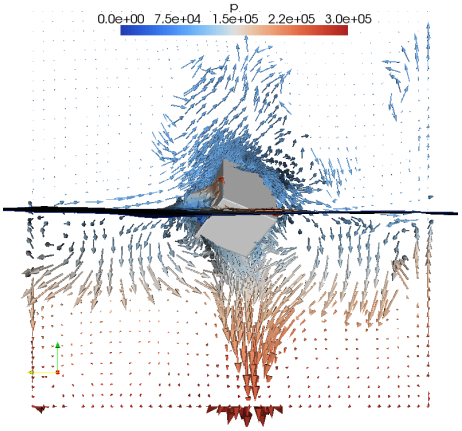


Figure 9: Case A: Slice in the yz-plane with pressure glyphs at the last time step.

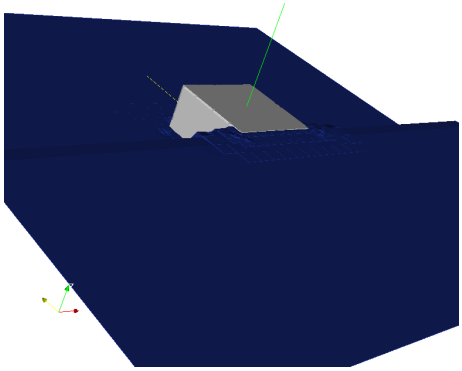


Figure 10: Case B: initial condition where part of the surface one is elevated.

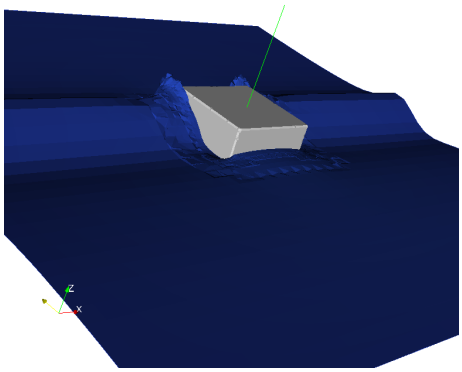


Figure 11: Case B: the wave just passes the box which is still stable

the box a third time. Here the box is already starting to tilt and rotate and the forth time it hits, figure 21 where water covers part of the side which is facing uppwards the solver again diverges.

7 Discussion

This sort of set up, using a cube that floats on water would be an unstable system in any real world experimental set up as well, considering that half the cube is submerged in the water and the center of gravity is right at the level of the surface. It describes though, how the box moves according to the forces induced by the wave hitting it.

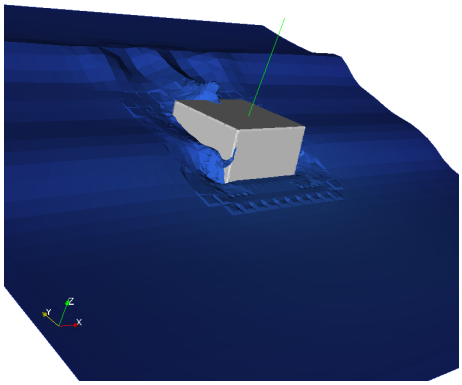


Figure 12: Case B: the box starts tilting a bit sue to the forces from the wave

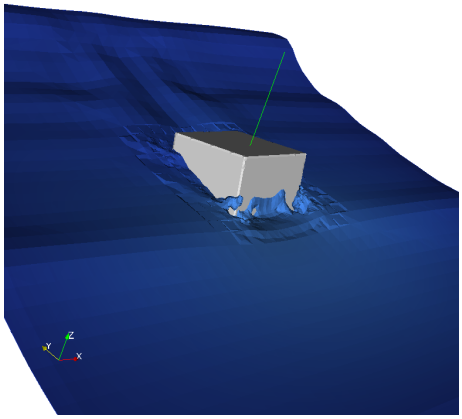


Figure 13: Case B: the box leans over to the left

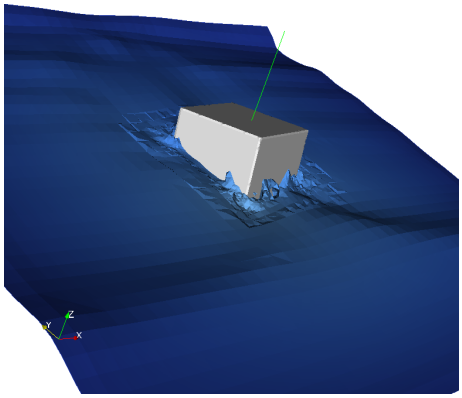


Figure 14: Case B: the box starts to rotate

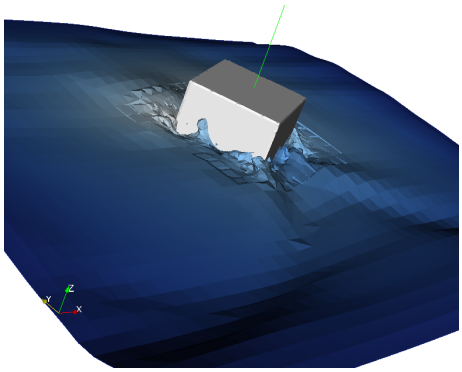


Figure 15: Case B: the box has now rotated to a critical stage

Considering a case where there is no wave induced and the water surface is perfectly still the box would be left balancing on the surface, figure 5 shows the surface before the simulation starts. That the box still flips over after some time and that the water close to the box is behaving differently than expected in a real life experiment is still not quite unexpected. With the errors involved in the method and the fact that the mesh is quite coarse makes this quite plausible.

7.1 Divergence

The solver diverges after about 10 seconds for the case with the coarse mesh. This is due a result of a deterioration in mesh quality which

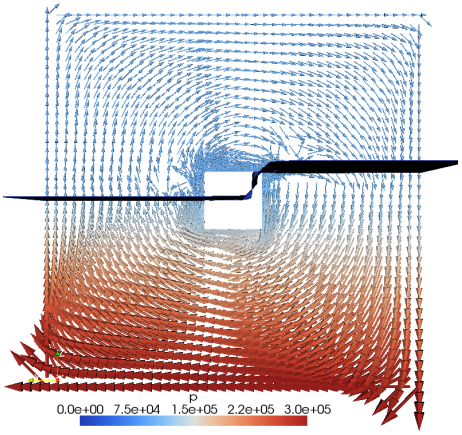


Figure 16: Case A: Slice in the yz-plane with pressure glyphs at the first time step

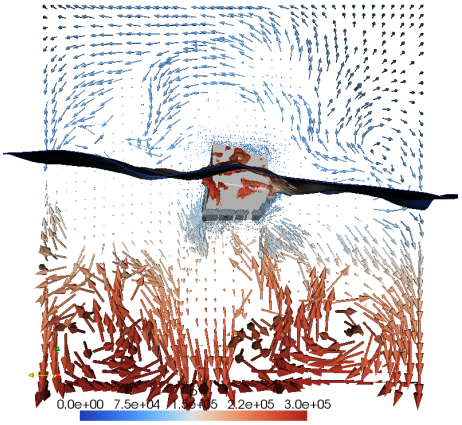


Figure 17: Case A: Slice in the yz-plane with pressure glyphs at the last time step.

is caused by the moving patch. In *paraFoam* there is a filter called mesh quality and when applying this with the setting hexagonal - skew it shows that the cells close to the patch is severely deformed in the last timestep. This could probably be helped by adding some kind of damping or stabilization scheme to the solver. it would probably make it more unphysical but on the other hand, it seems to be overreacting now to tilt over way to fast. Here, the type of spring damping systems referred to in some literature, see [et. al.(2007)] and [Hrvoje Jasak(2007)] could be used.

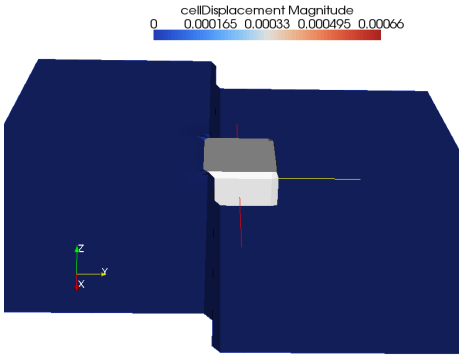


Figure 18: Case C: initial condition, the static wave has a height of 3 meters

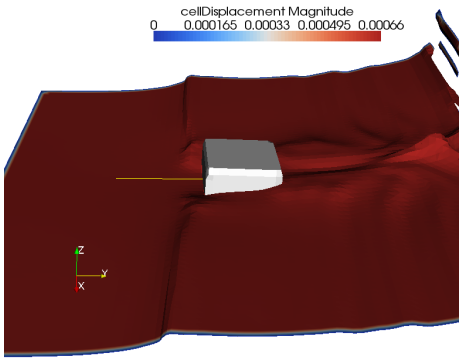


Figure 19: Case C: the wave hits the opposite side and the box is tilting

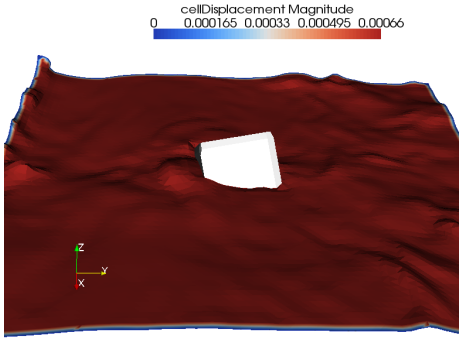


Figure 20: Case C: the wave hits again for the third time, coming from the left

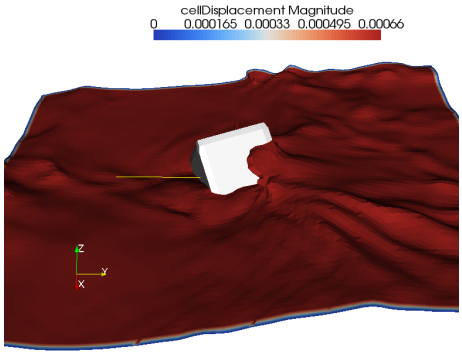


Figure 21: Case C: wave hits the fourth time and now sinks the box and the solver diverges

7.2 Future Work and Improvements

Much work and refinement can be done to make this code more accurate and more efficient both considering the mesh handling as well as the VoF parameters like the under-relaxation factors that could be optimized. One of the things that were considered was to have some kind of spring damping system for both rotation and translation to hinder the box to move more than the mesh allows before its quality deteriorates. Further, here a mesh generated in *snappyHexMesh* was used without a refinement box. Using the simpler *blockMesh* utility would even offer larger cells close to the surface and therefore a more

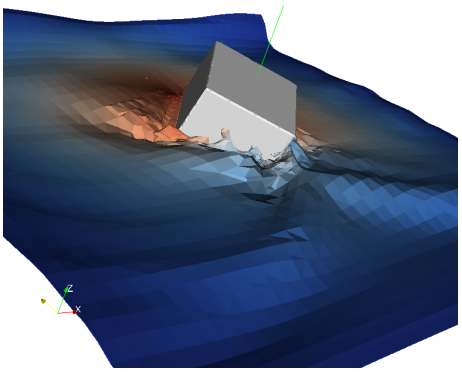


Figure 22: The box from case B, just before divergence.

forgiving mesh. Since the displacement has to be related to the cell size a move with small cells has a larger impact on the mesh quality than if the mesh is coarse.

It would be possible to implement a function that could find the center of gravity and the momentum of inertia for any considered body. Assigning it by hand works well for very simple geometries but quickly with complex structures it becomes very difficult.

References

[gui()] *Best Practice Guidelines for Marine Applicaion of Computational Fluid Dynamics*.

[R.M. Barron(2003)] Ali A salehi Neyshabouri R.M. Barron. Effects of under-relaxation factors on turbulent flow simulations. *International Journal For Numerical Methods in Fluids*, (42), 2003.

[Roozbeh Panahi(2006)] Mohammad S. Seif Roozbeh Panahi, Ebrahim Jahanbakhsh. Development of a vof-fractinal step solver for floating body motion simulation. *Applied ocean research*, 28 (28):171–181, August 2006.

[Hrvoje Jasak(2007)] Zeljko Tukovic Hrvoje Jasak. Automatic mesh motion for the unstructured finite volume method. *Transactions of FAMENA*, 30(2), 2007.

[Moradnia(2007)] Pirooz Moradnia. A tutorial on how to use dynamic mesh solver icodymfoam. *PhD course in CFD with OpenSource software*, Fall 2007.

[et. al.(2007)] Pablo M. Carrica et. al. Ship motions using single-phase level set with dynamic overset grids. *Computers Fluids*, (36):1415–1433, 2007.

8 Isosurface in *Paraview*

In order to create an iso-surface along with parts of the mesh as solids as used in the report one can use the filters included in *paraview*. Start *paraview* by running the *paraFoam* command as usual in the case folder.

- Check all the boxes for the patches in the *object inspector* for the case you are running
- In the *selection inspector* tab, click on create selection and in the ID's roll down menu below choose blocks
- By checking the patch you would like to include (the floating box in the report) the patch becomes highlighted
- In the *filters* menu pick the filter extract selection
  - Click copy
  - Click apply
  - Choose solid in the roll down menu
- choose the case, top one, in the *pipeline browser*
- Choose the *contour* filter for the iso surface.
- Set gamma to 0.5

If you have any problem, remake and be sure to have the right filter marked in the *pipeline browser*