Implementation of a standard level set method for incompressible two-phase flow simulations

Niklas Johansson
Abstract

Implementation of a standard level set method for incompressible two-phase flow simulations

Niklas Johansson

The level set method is a powerful way of tracking surfaces by defining the surface as a zero level set of a continuous function that is usually a signed distance function. The level set method is one of the best methods for simulating multi-phase flow because it can easily handle fast topological changes, as well as splitting and merging of fluids. In this thesis, a standard level set method was implemented in C++, using the finite element method library deal.II, to simulate incompressible two-phase flow on some benchmark problems. The results show a significant change of mass in the simulations, something that should not be allowed to happen when simulating incompressible fluids. The mass changes mainly occur in the reinitialization phase, where the level set function is rebuilt to look more like a signed distance function.
## Contents

1. **Introduction**  
   1

2. **Two-phase Flow**  
   2.1 Incompressible Navier-Stokes Equations  
   2.1.1 Variational Formulation  
   2  

3. **The Level Set Method**  
   3.1 Standard Level Set Method  
   3  
   3.2 Reinitialization  
   3.2.1 What Is Reinitialization?  
   3  
   3.2.2 Why Reinitialize?  
   4  
   3.2.3 Reinitialization Difficulties  
   4  
   3.2.4 Reinitialization Methods  
   3.2.4.1 PDE Method  
   4  
   3.2.4.2 Fast Marching Method  
   5  

4. **Numerical Results**  
   4.1 Vortex Test  
   4.1.1 Without Reinitialization  
   6  
   4.1.2 With Reinitialization  
   11  
   4.2 Rising Bubble Test  
   19  

5. **Conclusions**  
   28  

6. **Acknowledgements**  
   29
1 Introduction

Computational Fluid Dynamics (CFD) is today one of the main fields of computer simulations. By simulating a certain process, we can come to quicker and cheaper conclusions compared to an expensive and time consuming experimental approach with prototypes. By doing simulations we can quickly make changes and see how the system reacts to these changes in order to improve efficiency or optimize the process. Multi-phase flow is a subcategory within the CFD field, which involves two or more different interacting fluids. Predicting such physical systems numerically is very challenging. It is important to track surfaces, not only in CFD simulations but also in other fields such as image processing and shape optimization problems. In incompressible fluid simulations, volume conservation of the fluids is often very important and must be considered when deciding which numerical method to use, to ensure that the volume (area for problems in two dimensions) is preserved at all times in the simulation. This is especially true for long-time simulations where even small volume changes get the chance to add up over time, and therefore can render results invalid.

One way of defining an interface, is to explicitly parameterize the interface to define which points that belong to the surface. This is however not the easiest thing to do when dealing with surfaces that can move, split and merge over time. For that method to work one would need to have grid points located on the interface, and have the grid change as the interface moves. An alternative way is to use the level set method discussed in books by Sethian [12] and Osher and Fedkiw [10], an implicit approach that instead defines the interface as an isocontour of a level set function $\phi$, a function that is defined in the whole computational domain. The standard function that people use for the latter approach, is the signed distance function, a function defined as the distance to the interface in every point in the domain, and with different signs depending on what side of the interface it belongs to. That way, the zero isocontour of the signed distance function will represent the interface. By defining the interface in this implicit manner, there are no restrictions on the mesh like there are with the explicit approach, so the grid can be kept the same during the whole simulation if desired, but of course it is still possible to refine the mesh if needed.

I will show results from simulations done with the standard level set method on benchmark problems using the finite element method (FEM), to find data that can be used to compare this method to other methods, such as the conservative level set method discussed in papers by Olsson and Kreiss [8] and Olsson et al. [9], where different $\phi$ functions are used.

2 Two-phase Flow

The term two-phase flow refers to the motion of two different interacting fluids or with fluids that are in different phases, e.g. some parts of it can be in liquid form and some in a gaseous state. Phase transitions are also possible but they require more work than a standard liquid/gas system (e.g. water and air) without phase transitions and will not be looked at in this thesis, only immiscible incompressible fluids will be considered. Furthermore, a low enough Reynolds number is assumed so that the flow is laminar and not turbulent.

![Figure 1: Illustration of a two-phase problem. The interface $\Gamma$ separates the two fluids that reside in region $\Omega_1$ and $\Omega_2$.](image)

2.1 Incompressible Navier-Stokes Equations

The incompressible Navier-Stokes equations describe the motion of a single incompressible fluid. The first equation is the momentum conservation equation, and the second equation handles the
conservation of mass that results from the fact that the fluids are incompressible.

$$\rho \left( \frac{\partial \vec{v}}{\partial t} + (\nabla \vec{v}) \vec{v} \right) = -\nabla p + \nabla \cdot (\mu (\nabla \vec{v} + (\nabla \vec{v})^T)) + \vec{f}$$ (1)

Here, \( \vec{v} \) denotes the velocity of the fluid, \( \rho \) the density, \( p \) the pressure, \( \mu \) the dynamic viscosity and \( \vec{f} \) the external forces, which in case of gravity can be expressed as \( \vec{f} = \rho g \hat{g} \), where \( g \) is the acceleration caused by gravity and \( \hat{g} \) denotes the direction of gravity. When dealing with two-phase flow however, there are two fluids interacting with each other and not just one, which means that the Navier-Stokes equations have to be modified a bit to work in a two-phase environment. One way to deal with the jumps in normal stress at the interface \( \Gamma_i \) between the two fluids, is to add a force term to account for surface tension at the interface. In Equation 2, the surface tension is denoted by \( \vec{f}_i \) and is confined to the interface by the Dirac delta function. Also, there are discontinuities in the viscosity and density variables because the properties of the fluids are usually different.

$$\rho \left( \frac{\partial \vec{v}}{\partial t} + (\nabla \vec{v}) \vec{v} \right) = -\nabla p + \nabla \cdot (\mu (\nabla \vec{v} + (\nabla \vec{v})^T)) + \vec{f} + \vec{f}_i \delta(\Gamma_i, \bar{x})$$ (2)

### 2.1.1 Variational Formulation

When using FEM to find the velocity field \( \vec{v} \) and pressure \( p \), we solve the weak form of the equations instead of the original ones. First we define some spaces for the admissible velocity and pressure solutions.

\[
V_v = H^1(\Omega)^d \\
V_p = L_2(\Omega)
\]

Here, \( H^1(\Omega) \) is the space of all functions that are square-integrable and have square-integrable derivatives, \( L_2(\Omega) \) is the set of all square-integrable functions, and \( d \) is the number of dimensions.

The weak form is obtained by multiplying the equations with a test function and integrating over the whole domain \( \Omega \).

\[
\int_\Omega \rho \left( \frac{\partial \vec{v}}{\partial t} + (\nabla \vec{v}) \vec{v} \right) \cdot \Phi \, d\Omega = -\int_\Omega \nabla p \cdot \Phi \, d\Omega + \int_\Omega \nabla \cdot (\mu (\nabla \vec{v} + (\nabla \vec{v})^T)) \cdot \Phi \, d\Omega + \int_\Omega \vec{f} \cdot \Phi \, d\Omega + \int_\Omega \vec{f}_i \delta(\Gamma_i, \bar{x}) \cdot \Phi \, d\Omega
\]

(3)

\[
\int_\Omega \Psi \nabla \cdot \vec{v} \, d\Omega = 0
\]

(4)

The objective is to find the pair \((\vec{v}(\cdot, t), p(\cdot, t)) \in V_v \times V_p\) such that equations [3] and [4] hold for all test functions \((\Phi, \Psi) \in V_v \times V_p\). Discretization of space into elements demands that we replace the solution function spaces \(V_v\) and \(V_p\) by their subspaces \(V_v^h\) and \(V_p^h\), where the solutions are restricted to be continuous and piecewise polynomials within each element.

Equation [3] and [4] leads to a system of equations, which in matrix representation has the following structure:

\[
\begin{bmatrix}
A & B \\
B^T & 0
\end{bmatrix}
\begin{bmatrix}
\vec{v} \\
\bar{p}
\end{bmatrix} =
\begin{bmatrix}
\vec{f}^e \\
0
\end{bmatrix},
\]

(5)

where A and B are matrices, and \( \vec{V} \) and \( \bar{p} \) are vectors containing all unknown velocities and pressures. Matrix A and vector \( \vec{F}^e \) depends on what time discretization scheme we decide to use.

When the pressure and velocity fields are represented by inappropriate combinations of interpolation functions, and when dealing with flows with high Reynolds numbers, instabilities can appear, usually as oscillations in the pressure field. To account for this instability, several stabilized finite element techniques have been developed, such as streamline-upwind/Petrov-Galerkin (SUPG), pressure-stabilizing/Petrov-Galerkin (PSPG) and Galerkin/least-squares (GLS) formulations. For more details about stabilized FEM, see Nagraeth et al. [7], Donea and Huerta [3] and Tezduyar [14].
3 The Level Set Method

The level set method has since it was developed in the 1980s, revolutionized solutions to problems involving moving surfaces in computational science. The idea behind the level set method is to have the zero level set of a level set function $\phi$ act as a marker that determines the position of the interface. The level set function divides the domain into two or more regions, and by looking at the sign of $\phi$ in a certain point we can tell what region the point belongs to. In two-phase flow simulations the sign of the level set function tells what fluid occupies that point. The method was first introduced in a paper by Osher and Sethian [11] and later applied to two-phase incompressible flow by Sussman et al. [13]. A big advantage of the level set method is that merging and splitting of regions are easy when using this method, and that the grid does not have to be changed when the interface moves, and can be kept the same throughout the whole simulation. The level set method is described in detail in books by Osher and Fedkiw [10] and Sethian [12].

3.1 Standard Level Set Method

The standard way of doing the level set method is to use a signed distance function as the level set function $\phi$. If $\Omega^+$ and $\Omega^-$ are two regions with a shared boundary $\partial \Omega$, then the signed distance function is defined as

$$\phi(\vec{x}) = \begin{cases} 
    d(\vec{x}) & \forall \vec{x} \in \Omega^+ \\
    -d(\vec{x}) & \forall \vec{x} \in \Omega^-
\end{cases}$$

where $d(\vec{x}) = \min_{\vec{x}_B \in \partial \Omega} |\vec{x} - \vec{x}_B|$.

![Figure 2: Representing an interface in 1D by a signed distance function as the level set function $\phi$.](image)

The signed distance function has got a couple of nice properties that makes it a nice function to use. The first thing is that it is a very simple function without unnecessary oscillations, the other is that its gradient has got length 1 almost everywhere, which can simplify the computation of the normal $\hat{n} = \nabla \phi / |\nabla \phi|$ a bit because we do not have to normalize the gradient. This way, the gradient of the level set function can be used directly to approximate the normal. This simplification is not something I used in my code however, because the gradient is not guaranteed to be exactly 1 everywhere, even though that is the goal. By keeping the gradients from becoming too large, the numerical errors are minimized to avoid loss in accuracy, because larger gradients mean bigger errors. The level set function $\phi$ is transformed by the advection equation, where the level set function is transported along with the velocity field $\vec{v}$.

$$\phi_t + \nabla \cdot (\phi \vec{v}) = 0$$

(6)

If the velocity field has divergence zero, then the equation simplifies to

$$\phi_t + \vec{v} \cdot \nabla \phi = 0,$$

(7)

which is the case for all incompressible flows and also for velocity field in the vortex test described later in section 4.1.

3.2 Reinitialization

3.2.1 What Is Reinitialization?

The term reinitialization refers to the process of updating the level set function $\phi$, so that the intended shape and properties of the level set function is preserved as much as possible throughout
the simulation. The good properties of the signed distance function for example is something we want to keep as the simulation progresses. Although this might sound like an easy task, it really is not, because of the many factors one needs to consider. What method is best suited for the task? Is it stable? Does it scale well for larger problems? These are just a few things to consider.

3.2.2 Why Reinitialize?

As the level set function is transported in dynamic problems and because of numerical errors, the level set function will change its shape and will no longer have the desired look that we want it to, i.e. a signed distance function, and therefore the properties of the level set function will change over time, and \(|\nabla \phi| = 1\) will no longer be true. Steep and flat gradients are not desirable in a level set function as that can cause numerical instabilities and loss in accuracy, as large gradients produce larger errors. What we want is to have a smooth function with no unnecessary oscillations. That way, we can use interpolation to determine the normal vectors to points on the interface, something we would not be able to do if the function was poorly constructed.

3.2.3 Reinitialization Difficulties

When the level set function is reinitialized, the interface may move a little in the process, which is not at all desirable but close to impossible to avoid. This unwanted movement of the interface gives rise to some questions, like when should we reinitialize? Everytime we reinitialize, the interface moves, so reinitialization should only be done when necessary, as too frequent reinitializations will cause the interface movement to have too big of an impact on the final results, but too few will cause the level set function to lose its good properties, with loss of accuracy as a result.

3.2.4 Reinitialization Methods

There are a couple of ways to reinitialize the signed distance function. One popular method is to iteratively solve a PDE called the reinitialization equation (Eq. 8) until a steady state solution is obtained, so that the values in all grid points are updated simultaneously. Another popular way is to use the fast marching method (FMM), where the values get correctly updated one at a time, in a front propagating manner, moving away from the interface. Other possible methods are the fast sweeping method, the algebraic Newton method or the inefficient brute force method where distances to all points on the interface are calculated and compared without finesse or thought. In a paper by Hysing and Turek they discuss and compare these different reinitialization methods.

3.2.4.1 PDE Method

When you do the reinitialization using the PDE method, what you want to do is solve the following reinitialization equation suggested by Sussman et al. to steady state, i.e. until a state where the solution no longer change has been reached:

\[
\phi_t = \text{sign}(\phi_0)(1 - |\nabla \phi|) \tag{8}
\]

Here, \(\phi_0\) is the current level set function that we want to reinitialize and work as the initial condition for the reinitialization equation. Note that the \(t\) in Equation 8 is a pseudo-time and has nothing to do with the real physical time. The sign function is approximated by a smeared out function

\[
\text{sign}(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + \varepsilon^2}} \tag{9}
\]

in order to obtain better numerical results than with the original discontinuous function. The value of \(\varepsilon\) determines how much smoothing is applied, and an appropriate value can be the distance between grid points. However, this PDE is not very stable, which means that measures to improve stability must be taken by introducing an artificial diffusion term. Equation 8 now looks like this with added diffusion:

\[
\phi_t = \text{sign}(\phi_0)(1 - |\nabla \phi|) + \alpha \nabla^2 \phi \tag{10}
\]

One thing to note, is that the only part of the level set function that we are interested in is the zero level set and a small area around it. This means that it is not necessary for the whole level set function to become a signed distance function, as long as it behaves that way close to the interface. Thus, we can stop the reinitialization before the steady state solution has been reached, which is good because a steady state solution can take a long time to reach. If the level set function \(\phi\) already is close to a signed distance function before the reinitialization phase, usually only a few iterations are needed.
3.2.4.2 Fast Marching Method

The fast marching method was developed by J. A. Sethian as a way of solving the Eikonal equation, see Sethian [12] for details. The method is closely related to and is based on the same principles as the very useful Dijkstra’s algorithm that is used for shortest path computations, e.g. in network routers.

\[ F(\bar{x})|\nabla T(\bar{x}, t)| = 1 \]  
(Eikonal equation)

Here \( T \) is the time it takes for a wave traveling away from the interface in the normal direction with speed \( F(\bar{x}) \), to reach a point \( \bar{x} \). Under the condition that the speed function \( F(x) = 1 \), the Eikonal equation has the signed distance function as a solution. The way the method works is that the solution is constructed by having a front that propagates away from the interface one grid point at a time by adding the point that is closest to the interface and assigning it the correct value according to Algorithm 1. The method is a one-pass algorithm with an algorithmic complexity of \( O(N \log N) \), where \( N \) is the number of grid points, which makes the fast marching method quite fast.

Algorithm 1 Fast marching method update procedure

- Start by marking all boundary points as Known and all neighbors to a known point as a Trial point.
- while Trial points left do
  1. Let A be the Trial point with the lowest T value and mark it as Known.
  2. All neighbors of A that are not Known are marked as a Trial point.
  3. Compute new T values in all neighbors of A that are Trial points.
- end while

The fast marching method was implemented in my code using a level set method library called LSMLIB, developed by Chu and Prodanovic [2], with hopes that I would be able to make comparisons between the fast marching method and the PDE method. That did however not work as intended because of an unidentified bug in the library that sometimes caused some grid points of the level set function to be assigned incorrect values after the reinitialization, as if those points had been left out of the algorithm.

4 Numerical Results

To evaluate the standard level set method, the method was used on two benchmark problems. The first test is a pure advection problem with a given velocity field, in which the level set function is advected. In the second test, we apply the level set method to a real physical problem, a rising bubble driven by buoyancy. The simulations were made using FEM on cartesian grids of different sizes with bilinear finite elements, different reinitialization intervals and number of iterations. The programs were written in C++ using the open source finite element library deal.II, developed by Bangerth et al. [1], as a base. The library handles all FEM related things in the code (generating meshes, test functions, boundary conditions, constructing and solving equation systems etc.). The resulting equation systems from the weak form of the PDEs were solved using the iterative Generalized Minimal Residual method (GMRES) because it is more general and robust than other methods, such as the conjugate gradient method that requires a symmetric matrix, which is not always the case. In each time step we start by solving the Navier-Stokes equations (Eq. 2) to find the velocity field (except in the vortex test where the velocity field is already given). The level set function is then transported with the computed velocity field by the advection equation (Eq. 7), and then sometimes reinitialized by the reinitialization equation (Eq. 10).

4.1 Vortex Test

In this model problem test, we want to see how the method behaves when advected by an externally created velocity field in the form of a vortex. The equation that is solved is Equation 4 on the unit square with homogeneous Neumann boundary conditions. Note that this is a pure advection problem to test the transport properties of the level set method and is not governed by the Navier-
Stokes equations. The velocity field is given by the following expressions:

$$
\begin{align*}
    u &= 2 \sin (2\pi y) \sin^2 (\pi x) \cos (\pi t) \\
    v &= -2 \sin (2\pi x) \sin^2 (\pi y) \cos (\pi t)
\end{align*}
$$

where $u$ and $v$ are the x and y components of the velocity field $\vec{v}$. The initial condition is a circle with radius 0.15 centered in $(0.5, 0.75)$, with $\phi$ starting as a signed distance function with positive values inside the circle and negative outside, as shown in Figure 3.

![Figure 3: Initial configuration and domain for the vortex test.](image)

The direction of the vortex changes at $t = 0.5$, which means that when $t$ equals 1, $\phi$ should ideally once again be the same as the initial condition, but of course there are always error sources when dealing with discretizations. The ratio between time step $k$ and grid size $h$ was kept constant. Three different grids were used, $h = \{\frac{1}{128}, \frac{1}{256}, \frac{1}{512}\}$ with corresponding time steps $k = \{\frac{5}{10000}, \frac{5}{20000}, \frac{5}{40000}\}$. The time stepping for the advection equation (Eq. 7) was done using implicit Euler, which gives us the following equation:

$$
\frac{\phi^{n+1} - \phi^n}{k} + \vec{v}^{n+1} \cdot \nabla \phi^{n+1} = 0
$$

The weak form of Equation 12 then becomes

$$
\int_{\Omega} \phi \cdot \Phi \, d\Omega + k \int_{\Omega} \vec{v}^{n+1} \cdot \nabla \phi \cdot \Phi \, d\Omega = \int_{\Omega} \phi^n \cdot \Phi \, d\Omega.
$$

### 4.1.1 Without Reinitialization

Figures 4-9 show results from the simulations done without any reinitialization steps.
Figure 4: Results from a simulation with $h=1/128$, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 5: Results from a simulation with $h=1/256$, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 6: Results from a simulation with $h=1/512$, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.

Without reinitialization, the grid size has only a small impact on solution quality. At $t = 1$ the level set function once again looks almost exactly like it did in the beginning. It is also easy to see that between $t = 0$ and $t = 1$ the level set function is far from a signed distance function, which means that if this would be used to simulate a real physical flow problem, we would be having some problems. To compare the results, the final interface at $t=1$ for all simulations were merged together and can be seen in Figure 7, but the results are too similar to visually spot any big differences when no reinitialization is used. To better see the differences, Figure 8 shows a zoomed in version of Figure 7.
Figure 7: Interface comparison at t=1 without reinitialization. The black line represent the theoretical boundary, the blue the results on the coarsest grid, the red with a medium sized grid, and the green on the finest grid.

Figure 8: Same as Figure 7 but zoomed in on a small region to better illustrate the differences. The black line represent the theoretical boundary, the blue the results on the coarsest grid, the red with a medium sized grid, and the green on the finest grid.

Figure 9 and Table 1 shows how the area changes during the simulation and that it is far from conserved, but rather shows a first-order convergence rate.
Figure 9: Area conservation plot for different grid sizes.

Table 1: Final area change at t=1.

<table>
<thead>
<tr>
<th>Grid size h</th>
<th>Area change</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/128</td>
<td>-0.84%</td>
</tr>
<tr>
<td>1/256</td>
<td>-0.41%</td>
</tr>
<tr>
<td>1/512</td>
<td>-0.20%</td>
</tr>
</tbody>
</table>

4.1.2 With Reinitialization

To reinitialize the level set function into something that is closer to a signed distance function, the PDE method was used. The reinitialization equation (Eq. 10) was solved using Explicit Euler to compute the next iteration. The variable $\alpha$ in the artificial diffusion term in Eq. 10 was chosen to be proportional to the grid size $h$, so that $\alpha = 0.0625h$, which results in enough stability while not changing the general behavior of the PDE too much. Simulation were made with two different reinitialization intervals, one iteration in every time step, and four iterations every ten time steps. The results are shown in figures 10-15 where we can see that the level set function now looks more like a signed distance function, but also that the zero level set quickly becomes distorted on the lower resolutions.
Figure 10: Results from a simulation with $h=1/128$ and one reinitialization iteration in every time step, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 11: Results from a simulation with $h=1/256$ and one reinitialization iteration in every time step, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 12: Results from a simulation with $h=1/512$ and one reinitialization iteration in every time step, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 13: Results from a simulation with $h=1/128$ and 4 reinitialization iterations every 10 time steps, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 14: Results from a simulation with $h=1/256$ and 4 reinitialization iterations every 10 time steps, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 15: Results from a simulation with h=1/512 and 4 reinitialization iterations every 10 time steps, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 16: Interface comparison at $t=1$ with one reinitialization iteration every time step. The black line represent the theoretical boundary, the blue the results on the coarsest grid, the red with a medium sized grid, and the green on the finest grid.

Figure 17: Interface comparison at $t=1$ with four reinitialization iterations every ten time steps. The black line represent the theoretical boundary, the blue the results on the coarsest grid, the red with a medium sized grid, and the green on the finest grid.

Figure 16 and 17 show that the interface has moved significantly from its original position because the reinitialization steps have distorted the interface during the whole simulation, which shows the problem of using reinitialization. The reinitialization effects can also clearly be seen in Figure 18 where we see that the area has decreased significantly, something that is not at all desirable in an incompressible flow simulation like this, where we want the area to be conserved at all times. The deformations of the interface with lower resolutions occur mainly at around $t=0.5$. 

18
when the flow changes direction, and the original circle has been stretched to its maximum. This happens because the interface in the upper parts gets very thin, only a few grid cells wide on the coarsest grid (see figures 10c and 13c).

Figure 18: Area conservation plot using different reinitialization intervals and grid sizes.

Table 2: Final area change at t=1.

<table>
<thead>
<tr>
<th>Grid size $h = 1/128$</th>
<th>Reinitialization interval</th>
<th>Reinitialization iterations</th>
<th>Area change</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h = 1/128$</td>
<td>1</td>
<td>1</td>
<td>-4.39%</td>
</tr>
<tr>
<td>$h = 1/256$</td>
<td>1</td>
<td>1</td>
<td>-2.74%</td>
</tr>
<tr>
<td>$h = 1/512$</td>
<td>1</td>
<td>1</td>
<td>-1.34%</td>
</tr>
<tr>
<td>$h = 1/128$</td>
<td>10</td>
<td>4</td>
<td>-2.11%</td>
</tr>
<tr>
<td>$h = 1/256$</td>
<td>10</td>
<td>4</td>
<td>-0.94%</td>
</tr>
<tr>
<td>$h = 1/512$</td>
<td>10</td>
<td>4</td>
<td>-0.55%</td>
</tr>
</tbody>
</table>

4.2 Rising Bubble Test

In this benchmark problem, proposed by Hysing et al. [5], a rising bubble was simulated in $\mathbb{R}^2$ from $t = 0$ to $t = 3$. The governing equations are the incompressible Navier-Stokes equations (Eq. 2), on the computational domain $[0,1] \times [0,2]$, with a no-slip boundary condition at the top and bottom edges, and a free slip boundary condition on the left and right edges. The initial configuration is a circular bubble at rest, located at $[0.5, 0.5]$ with radius 0.25, consisting of a fluid with a lower density and viscosity than the surrounding fluid (see Figure 19). This time the used grid sizes were $h=1/80$, $1/160$ and $1/320$. The time step to grid size ratio was kept constant, starting with a time step of 0.005 for the coarsest grid. The PDE method for reinitialization was used to try to keep the level set function looking close to a signed distance function, and just like in the vortex test, the artificial diffusion $\alpha$ in the reinitialization equation (Eq. 10) was chosen to be proportional to the grid size $h$, but this time by a factor 0.1.

The surface tension $\vec{f}_i$ in the Navier-Stokes equations was implemented as a volume force $\sigma \kappa \hat{n}$, where $\sigma$ is the surface tension coefficient, $\kappa$ the curvature of the interface and $\hat{n}$ the unit normal at the interface pointing into fluid 1. If the level set function is a signed distance function, then $\hat{n} = \nabla \phi$, so the surface tension term can be written as $\vec{f}_i = \sigma \kappa \nabla \phi$. The curvature is calculated using the formula $\kappa = -\nabla \cdot \hat{n}$. 


Figure 19: Computational domain, initial configuration and boundary conditions for the rising bubble test.

Table 3: Simulation parameter values for the rising bubble test.

<table>
<thead>
<tr>
<th>$\rho_1$</th>
<th>$\rho_2$</th>
<th>$\mu_1$</th>
<th>$\mu_2$</th>
<th>$g$</th>
<th>$\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1</td>
<td>0.01</td>
<td>0.001</td>
<td>0.98</td>
<td>0.0245</td>
</tr>
</tbody>
</table>

The time stepping schemes used to solve this problem was BDF-2 for the main equations (Eq. 2), and Explicit Euler for the reinitialization equation (Eq. 10).
Figure 20: Results from a simulation with $h=1/80$ and 4 reinitialization iterations every time step, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.

Figure 21: Results from a simulation with $h=1/80$ and 10 reinitialization iterations every 5 time steps, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.

Figure 22: Results from a simulation with $h=1/160$ and 4 reinitialization iterations every time step, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
Figure 23: Results from a simulation with $h=1/160$ and 10 reinitialization iterations every 5 time steps, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.

Figure 24: Results from a simulation with $h=1/320$ and 4 reinitialization iterations every time step, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.

Figure 25: Results from a simulation with $h=1/320$ and 10 reinitialization iterations every 5 time steps, showing $\phi$ and its contours at five different points in time. The black curve shows the interface.
In figures 20-25 we see that the bottom of the bubble flattens out as the bubble rises. Clearly the level set function does not look like a signed distance function after a while. This is not something we need to worry about as long as the level set function is well behaved near the zero level set that represents the interface. It needs to be well behaved near the interface to be able to properly calculate the surface tension forces. In Figure 26 that shows the magnitude of the gradient, it can be seen that the level set function is not a signed distance function, because if it was, the whole region would be green, which corresponds to a gradient with length 1, but that is clearly not the case. Close to the zero level set that defines the interface between the two fluids however, there is a narrow region with the signed distance properties that we want and need.

Figure 26: Plot showing $|\nabla \phi|$ and the zero level set at two different points in time on the finest grid with reinitialization every time step (cf. Figure 24).

Figure 28 shows that the area is once again not conserved in the simulations, like we want it to be, but rather shows a very significant loss of as much as 9 % in the worst case. The area loss was extra severe with frequent reinitializations. This means that some changes has to be made if one would want to do longer simulations with reasonable results.

Figure 27: Legend associated with figures 28-34.
Figure 28: Bubble area changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.

Figure 29: Bubble perimeter changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.

Figure 29 shows how the bubble perimeter changes in the simulations. As expected, the simulations with smallest perimeters also have the largest area losses. In Figure 30 we can see that the circularity is larger in the simulations with lower resolution.
Figure 30: Bubble circularity changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.

The bubble's center of mass error in x-direction is shown in Figure 31 and the bubble’s velocity in x-direction is shown in Figure 32. The center of mass should ideally not move in the x-direction because the problem is completely symmetric. However, there are still some small errors. The source of these errors is probably the iterative solver that does not solve the equation systems exactly, but rather with a certain tolerance.

Figure 31: Bubble error in x-position changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.
Figure 32: Bubble velocity in x-direction changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.

Figures 33 and 34 show the y-component of the bubble’s center of mass and the rise velocity, respectively. With lower resolution, the bubble rises faster because the bubble has become smaller than in the simulations with higher resolution.

Figure 33: Bubble y-position changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.
Figure 34: Bubble velocity in y-direction changes over time for different grid sizes and reinitialization intervals. See legend in Figure 27.

Figure 35: Bubble shapes at t=3 with reinitialization in every time step with \( h = 1/80 \)(shown in red), 1/160(shown in blue), 1/320(shown in green).

In figures 35 and 36, the final positions of the interfaces are shown, and it can clearly be seen that the bubble has become smaller and risen faster on the lower resolutions.
Figure 36: Bubble shapes at $t=3$ with reinitialization every 5 time steps with $h = 1/80$ (shown in red), $1/160$ (shown in blue), $1/320$ (shown in green).

In tables 4 and 5 I present data collected in the simulations and compare them to results observed in simulations by Hysing et al. [5]. There we can see that my results on the finest grid does not completely agree with the results found by Hysing et al. [5].

Table 4: Collected data from simulations with reinitialization in every time step, and reference values observed in simulations by Hysing et al. [5].

<table>
<thead>
<tr>
<th>$1/h$</th>
<th>80</th>
<th>160</th>
<th>320</th>
<th>Reference values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area change</td>
<td>-8.92%</td>
<td>-5.66%</td>
<td>-1.55%</td>
<td>0.9012 ± 0.0001</td>
</tr>
<tr>
<td>Min circularity</td>
<td>0.9029</td>
<td>0.9034</td>
<td>0.9014</td>
<td></td>
</tr>
<tr>
<td>Time for min circularity</td>
<td>1.8750</td>
<td>1.8900</td>
<td>1.8987</td>
<td>1.9</td>
</tr>
<tr>
<td>Max rise velocity</td>
<td>0.2446</td>
<td>0.2435</td>
<td>0.2422</td>
<td>0.2419 ± 0.0002</td>
</tr>
<tr>
<td>Time for max velocity</td>
<td>0.9550</td>
<td>0.9350</td>
<td>0.9263</td>
<td>[0.921, 0.932]</td>
</tr>
<tr>
<td>Center of mass at $t=3$</td>
<td>1.1037</td>
<td>1.0921</td>
<td>1.0840</td>
<td>1.081 ± 0.001</td>
</tr>
</tbody>
</table>

Table 5: Collected data from simulations with reinitialization every five time steps, and reference values observed in simulations by Hysing et al. [5].

<table>
<thead>
<tr>
<th>$1/h$</th>
<th>80</th>
<th>160</th>
<th>320</th>
<th>Reference values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area change</td>
<td>-3.74%</td>
<td>-2.48%</td>
<td>-0.67%</td>
<td>0.9012 ± 0.0001</td>
</tr>
<tr>
<td>Min circularity</td>
<td>0.8996</td>
<td>0.9017</td>
<td>0.9009</td>
<td></td>
</tr>
<tr>
<td>Time for min circularity</td>
<td>1.8700</td>
<td>1.8850</td>
<td>1.8987</td>
<td>1.9</td>
</tr>
<tr>
<td>Max rise velocity</td>
<td>0.2431</td>
<td>0.2426</td>
<td>0.2419</td>
<td>0.2419 ± 0.0002</td>
</tr>
<tr>
<td>Time for max velocity</td>
<td>0.9500</td>
<td>0.9375</td>
<td>0.9250</td>
<td>[0.921, 0.932]</td>
</tr>
<tr>
<td>Center of mass at $t=3$</td>
<td>1.0957</td>
<td>1.0872</td>
<td>1.0826</td>
<td>1.081 ± 0.001</td>
</tr>
</tbody>
</table>

5 Conclusions

From what we have seen is the benchmark tests, the standard level set method does a poor job at mass conservation. Another thing to note, is that very frequent reinitializations is necessary to keep the level set function well behaved near the interface, in order to get accurate results, but by doing so we also lose much area. From the results shown in table 4 and 5 it is clear that my results do not fully agree with the results observed by Hysing et al. [5]. One reason for this is probably because they had better mass conservation in their simulations. The lack of mass conservation
is an issue that must be addressed if someone wants to use the standard level set method to get accurate results that agree with the laws of physics.

6 Acknowledgements

The simulations were performed on resources provided by the Swedish National Infrastructure for Computing (SNIC) at Uppsala Multidisciplinary Center for Advanced Computational Science (UPPMAX).
References


