Accurate interface-tracking of surfaces in three dimensions for arbitrary Lagrangian-Eulerian schemes

by

Tormod Bjøntegaard[1] and Einar M. Rønquist[2]

PREPRINT
NUMERICS NO. 9/2011

[1]Norwegian University of Science and Technology, Department of Mathematical Sciences, Trondheim, Norway

[2]Norwegian University of Science and Technology, Department of Mathematical Sciences, mailing address: N-7491 Trondheim, Norway, fax: +47 73 59 35 24, phone: +47 73 59 35 47, email: ronquist@math.ntnu.no

Accurate interface-tracking of surfaces in three dimensions for arbitrary Lagrangian-Eulerian schemes

Tormod Bjøntegaard[‡] and Einar M. Rønquist[§]

October 20, 2011

We extend the computational method presented in [1] for tracking an interface immersed in a given velocity field to three spatial dimensions. The proposed method is particularly relevant to the simulation of unsteady free surface problems using the arbitrary Lagrangian-Eulerian framework, and has been constructed with two goals in mind: (i) to be able to accurately follow the interface; and (ii) to automatically maintain a good distribution of the grid points along the interface. The method combines information from a pure Lagrangian approach with information from an ALE approach. The new method offers flexibility in terms of how an "optimal" point distribution should be defined, and relies on the solution of two-dimensional surface convection problems. We verify the new method by solving model problems both in the single and multiple spectral element case, and we compare this method with other traditional alternatives. We have been able to verify first, second, and third order temporal accuracy for the new method by solving these three-dimensional model problems.

1 Introduction

The ability to accurately follow time-dependent surfaces is very important in many areas of computational science and engineering. An important class of such problems is free surface flows, with the free surface representing the interface between two fluids, e.g., air and water. Computational methods for solving such problems can typically be classified into two categories: methods which explicitly track the free surface (interface-tracking methods; e.g., [17]) and methods where the interface is more implicitly defined (e.g., level set methods [16, 18, 15] or volume-of-fluid methods [8]); we will here focus on the former class.

[‡]Norwegian University of Science and Technology, Department of Mathematical Sciences, Trondheim, Norway

[§]Norwegian University of Science and Technology, Department of Mathematical Sciences, mailing address: N-7491 Trondheim, Norway, fax: +47 73 59 35 24, phone: +47 73 59 35 47, email: ronquist@math.ntnu.no

Interface-tracking methods (or sometimes also referred to as front-tracking methods) comprise a few essential steps. At any particular point in time, a velocity field is typically determined from the governing equations within the fluid(s), e.g., by solving the Navier-Stokes equations. By integrating this velocity field, it is possible to obtain a new position of the interface.

A pure Lagrangian approach applied to an evolving interface is simply based on integrating the velocity of the fluid particles along the surface to obtain the position of the surface at a later point in time. However, in the context of a numerical approximation (e.g., using finite-element-based methods), a pure Lagrangian approach is often not a very practical approach since it typically results in large deformations of the computational domain.

In the context of free surface flows, the arbitrary Lagrangian-Eulerian (ALE) formulation of the governing equations has been very successful as a point of departure for a numerical approximation [9, 5, 11]. A typical approach to updating the free surface is to enforce a kinematic condition along the surface. This condition has its origin in a continuum description, and says that the normal fluid velocity has to be equal to the normal domain velocity at any point along the surface. An important consequence of this condition is the fact that a fluid particle which is present somewhere along the free surface at a particular time will also be present at the free surface at a later time.

While the kinematic condition enforces a normal condition, a tangential domain velocity also needs to be specified along the surface; a common choice is to enforce a homogeneous Dirichlet condition for the tangential component [19, 2]. This choice typically reduces the deformation of the computational domain compared to a pure Lagrangian approach, however, it offers limited control over the quality of the grid used to represent the free surface. In particular, the *distribution* of the grid points along the free surface may deteriorate over time, which may ultimately result in severe loss of accuracy (or even breakdown of the simulation). This latter issue may be dealt with in various ways, e.g., through remeshing or other mesh update strategies [12, 6]. However, the temporal accuracy will typically suffer using such a strategy.

The issue of a non-optimal evolution of the surface representation is particularly acute in the context of using high order finite elements or spectral elements. The reason for this is related to the fact that such methods depend on locally regular mappings between a reference domain and the corresponding physical element. If the distribution of the surface points along the free surface becomes very distorted, this mapping may not be so regular anymore, resulting in a loss of spatial accuracy. This will again affect the calculation of tangent and normal vectors, as well as the local curvature, since the computation of these quantities depends on the coupling between many surface points [10, 20].

One could also imagine enforcing the kinematic condition together with a tangential component of the domain velocity in such a way that the integration of the total domain velocity would: (i) result in an accurate representation of the free surface; and (ii) maintain a good distribution of the grid points along the surface [6, 3]. An obvious challenge with this approach is how to define the overall domain velocity in such a way that not only good spatial accuracy is achieved (with no need for remeshing), but in a way that will also ensure good temporal accuracy (better than first order). The work presented in [1] suggested alternative ways to achieve these two goals in two spatial dimensions.

The work presented here is an extension of [1] to three spatial dimensions. In the original work the overall strategy when advancing the interface from time-level t^n to t^{n+1} was based on searching for particles on the interface at t^n which, through a pure Lagrangian motion, end up at desirable positions at time t^{n+1}. Similar to the two-dimensional study [1], we only discuss the evolution of a surface when it is "immersed" in a known velocity field; no partial differential equation will be solved to obtain this velocity field. The main reason for this is the dearth of available analytical solutions in the context of solving problems with an evolving interface (e.g., the full Navier-Stokes equations with free surfaces). We let the three-dimensional surface evolve in time, and different computational strategies for predicting the surface evolution will be tested and compared. Numerical tests will illustrate the similarities and differences between the methods, and conclusions will be made based on these.

2 Problem description

The motivation behind this work is to solve the three-dimensional Navier-Stokes equations in a domain with a free surface. In this work the focus is on how to maintain a good representation of the boundary throughout such a simulation. By solving the Navier-Stokes equations we get a pressure field, p, and a velocity field, $\mathbf{u}$, at a given time, t. It is this velocity field on the surface of the domain which determines the shape of the domain as time evolves. In particular, if we consider the surface as a collection of surface particles, the motion of the particle at location $\mathbf{x} = (x_1, x_2, x_3)$ is governed by

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{u}(\mathbf{x}, t), \tag{2.1}$$

where $\mathbf{u} = (u_1, u_2, u_3)$ is the velocity field and t is time.

In the following we assume that the Navier-Stokes equations are formulated in the arbitrary Lagrangian-Eulerian(ALE) framework. In this framework a third field, $\mathbf{w}$, is introduced and denoted as the domain velocity (or grid velocity in the discrete case). The *shape* of the evolving surface only depends on the normal component of the velocity field, $\mathbf{u}$; hence, the domain (or grid) velocity is connected to the fluid velocity through the kinematic condition

$$\mathbf{w} \cdot \mathbf{n} = \mathbf{u} \cdot \mathbf{n}, \tag{2.2}$$

where $\mathbf{n}$ is a normal vector. Thus, if the motion of all the surface particles satisfies

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{w}(\mathbf{x}, t) \tag{2.3}$$

instead of (2.1), the shape of the surface will remain the same.

Typically, in a numerical approximation the surface is represented by a selection of grid-points, and updating the numerically approximated surface involves moving each individual grid node. Hence, in the update from time-level t^n to t^{n+1} we may for each grid-node integrate (2.3), where $\mathbf{w}$ satisfies (2.2). The *temporal* accuracy may be achieved by integrating this equation with the desired accuracy, however the *spatial* accuracy will depend on $\mathbf{w} \cdot \mathbf{t}$, where $\mathbf{t}$ can be any vector in the tangent plane. This tangential component of the grid-velocity may be chosen freely, and this is something we wish to exploit here; see Figure 2.1. The aim of this work is to indirectly choose $\mathbf{w} \cdot \mathbf{t}$ in such a way that both the temporal and the spatial accuracy is maintained.

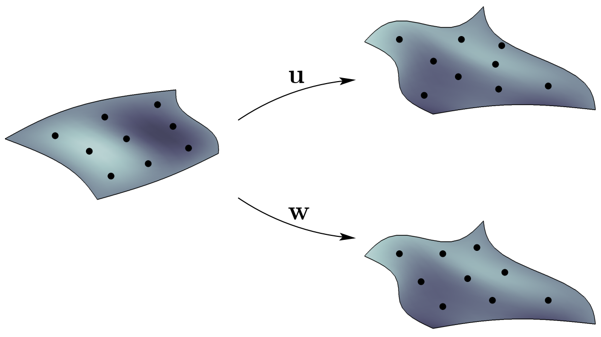


Figure 2.1: The evolution of nine grid-points by integrating (2.3). In the first case $\mathbf{w} = \mathbf{u}$ is chosen, while in the other case we are able to control the distribution of the grid points by a clever choice of $\mathbf{w} \cdot \mathbf{t}$.

2.1 Spectral element discretization

In the following we consider front-tracking in the context of solving the Navier-Stokes equations using a spectral element spatial discretization [4]. In particular, we restrict our attention to the situation where our domain can be viewed as a deformed sphere, and where the surface is represented by six spectral elements; see Figure 2.2. This test problem will suffice for the limited deformations we consider in this study. Most importantly, we assume that we don't have a surface which "folds over"; we will comment on this aspect later.

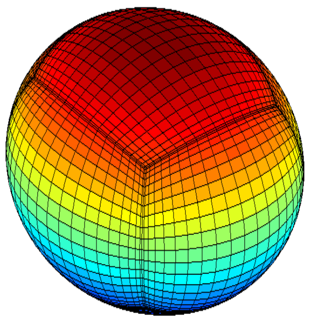


Figure 2.2: A sphere represented by 6 surface elements.

2.1.1 Surface parameterization

Using a standard spectral element discretization, the surface Γ is represented by K spectral elements, Γ^k, $k = 1, \ldots, K$; see Figure 2.2. For a given $(\xi_1, \xi_2) \in \hat{\Gamma} = [-1, 1]^2$, the corresponding coordinate on Γ^k is given through the mapping $\mathcal{F}^k$, $\mathbf{x} = (x_1^k(\xi_1, \xi_2), x_2^k(\xi_1, \xi_2), x_3^k(\xi_1, \xi_2))$; see Figure 2.3. In general, any field variable φ associated with the front on element k is represented in terms of the reference variables ξ_1 and ξ_2. In particular, an Nth order polynomial approximation of φ over Γ^k at time t^n can be expressed in terms of the following nodal basis:

$$\varphi^{n,k}(\xi_1, \xi_2) = \sum_{l=0}^{N} \sum_{m=0}^{N} (\varphi^n)^k_{lm} \ell_l(\xi_1) \ell_m(\xi_2). \tag{2.4}$$

Here, $\ell_m(\xi)$ is the Nth order Lagrangian interpolant associated with the $N+1$ GLL-points, $\xi_{GLL,j}$, $j = 0, \ldots, N$, such that $\ell_m(\xi_{GLL,j}) = \delta_{jm}$. For example, the coordinates $x_i, i = 1, 2, 3$, over Γ^k at time t^n are represented as

$$(x_i)^{n,k}(\xi_1, \xi_2) = \sum_{l=0}^{N} \sum_{m=0}^{N} (x_i^n)^k_{lm} \ell_l(\xi_1) \ell_m(\xi_2). \tag{2.5}$$

We use underscore to denote a vector comprising all the nodal values associated with a field variable on element k, e.g., the vector $\underline{x}_1^{n,k}$ represents all the values $(x_1^n)^k_{lm}$, $l, m = 0, \ldots, N$, at time-level t^n on element k. We also denote $\underline{\mathbf{x}}^{n,k} = (\underline{x}_1^{n,k}, \underline{x}_2^{n,k}, \underline{x}_3^{n,k})$. The following surface variables are assumed to be known for a second order temporal scheme:

$$\underline{\mathbf{x}}^{n,k}, \underline{\mathbf{x}}^{n-1,k}, \underline{\mathbf{u}}^{n,k}, \underline{\mathbf{u}}^{n-1,k}, \underline{\mathbf{w}}^{n-1,k}, \underline{\mathbf{w}}^{n-2,k}, \quad k = 1, \ldots, K.$$

The motivation behind storing these particular fields will be apparent when the new strategy is described.

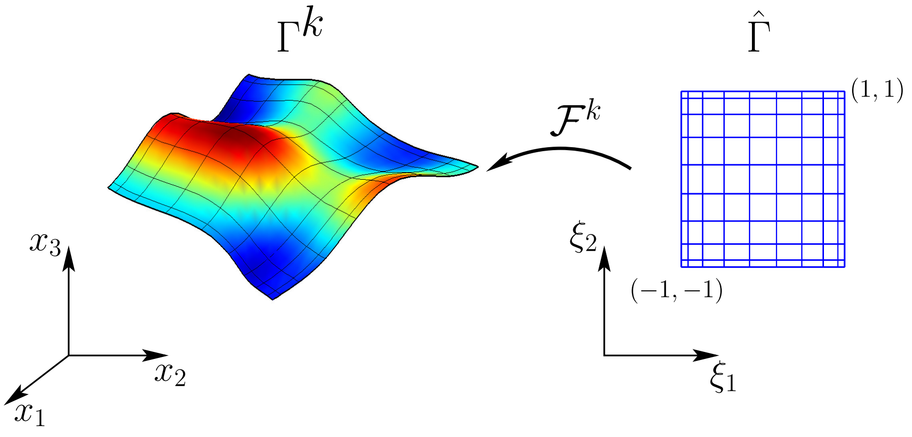


Figure 2.3: The mapping of the reference domain to the physical domain for a single spectral element.

3 New strategy

We will here describe the new strategy for a closed surface, which can be viewed as a deformed sphere, discretized by 6 spectral elements. The element configuration on a sphere is given in Figure 2.2. The focus here is on how to update the computational domain from t^n to t^{n+1}; our main objective is to track the surface with higher than first order accuracy in time and at the same time control the spatial accuracy.

We will assume that the entire surface can be viewed as a collection of "surface particles" which remain on the surface during the simulation. In order to determine the position for a grid node at time t^{n+1}, the idea will be to search for the particle at time t^n which, when moved through a pure Lagrangian motion, ends up in a "good position". With "good position" we mean that the collection of grid nodes, which are moved in this fashion, on each element at time t^{n+1} should make the mapping from the reference domain to the physical domain as smooth as possible. In order to move a particle through a Lagrangian motion we need to integrate (2.1). Hence, in the case of using a multi-step time integrator, we need information about the velocity of a given fluid particle at different time-levels in order to achieve higher than first order accuracy. Obviously, the surface particle which is located at a grid node at t^{n+1} will in general *not* be located at a grid node at t^n (and earlier time levels); see Figure 3.1(a). Hence, we don't have immediate access to this particle's velocity. Specifically, a surface particle at the position defined by (ξ_1^*, ξ_2^*) for element k at t^{n+1} will, in general, be located at a different and unknown position $(\hat{\xi}_1, \hat{\xi}_2)$ and possibly also a different element at time level t^n. Thus, in order to get this particle's velocity at t^n or earlier time levels, we would first need to find the correct element and $(\hat{\xi}_1, \hat{\xi}_2)$, and then evaluate the relevant fields in (2.4).

The surface at t^{n+1} is unknown and a main issue is to quantify what we mean by a "good position" without knowing the exact shape of the surface. Here, for the update of each grid node, we choose to do this by finding the surface particle at t^n which is located along a particular vector at t^{n+1}; see Figure 3.1(b). It is here that the restriction of not considering "folding surfaces" becomes relevant, as we require that only one surface particle ends up along this direction at t^{n+1}.

We have three different classes of grid nodes to update: (i) the corner points of each element, (ii) the element edges of each element, and (iii) the interior points of each element. The directional vectors associated with these three types of nodes need to satisfy different requirements, and this section is devoted to how these vectors may be chosen. Once these vectors have been constructed, we discuss in Section 4 how we search for the relevant particles at t^n . The overall operation for the update of the surface from t^n to t^{n+1} involves three main consecutive steps:

1. Update the corner points (8 in total)

2. Update the element edges (12 in total)

3. Update all the interior surface points

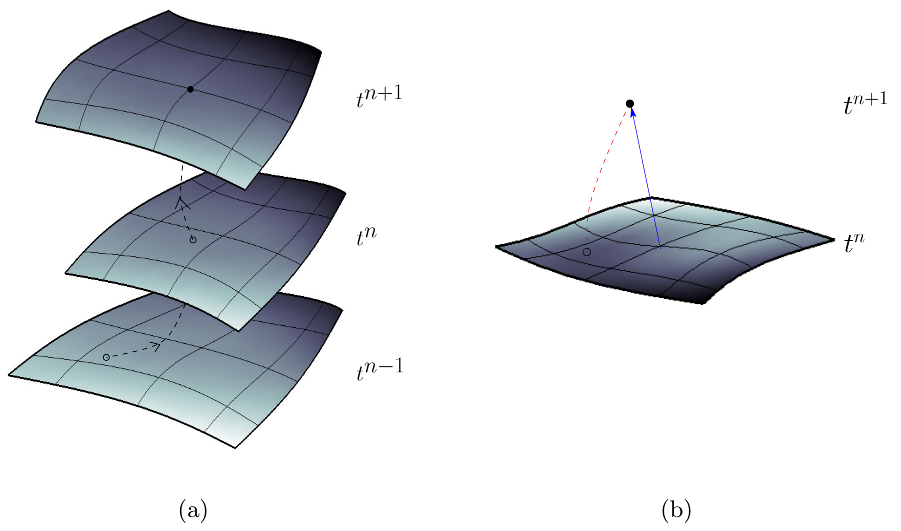


Figure 3.1: (a) The path of a surface particle located at a grid node at time t^{n+1}. The position of this particle at time level t^n and t^{n-1} will depend on the underlying velocity field. (b) A directional vector and the particle at t^n (open circle) which is located along this vector at time t^{n+1} (full circle).

3.1 Update of the corner points

We now describe a procedure which tries to position the corner points in such a way that the areas of the surface elements are similar *and* that the angles of the connecting edges are close to 90 degrees; see Figure 3.2. Here, the red circle is the centre of the domain bounded by the closed surface at t^n. We now construct eight vectors originating from this central point, and we want each of the corner points at t^{n+1} to be located along one of these vectors. We also wish these vectors to adapt to the shape of the surface; for instance we want to update the corner-points differently if the surface is close to a sphere compared to a surface with an ellipsoidal shape.

The first step is to compute a measure of the general shape of the known surface at t^n. This measure may be represented by the six points found by evaluating the coordinates in the centre of the reference domain for each element; see Figure 3.3. Once these center points are determined, we compute the directional vectors

$$
\begin{aligned}
\mathbf{p}^1 &= (\mathbf{x}^1 - \mathbf{x}^c) + (\mathbf{x}^2 - \mathbf{x}^c) + (\mathbf{x}^5 - \mathbf{x}^c)\\
\mathbf{p}^2 &= (\mathbf{x}^1 - \mathbf{x}^c) + (\mathbf{x}^2 - \mathbf{x}^c) + (\mathbf{x}^3 - \mathbf{x}^c)\\
\mathbf{p}^3 &= (\mathbf{x}^1 - \mathbf{x}^c) + (\mathbf{x}^3 - \mathbf{x}^c) + (\mathbf{x}^4 - \mathbf{x}^c)\\
\mathbf{p}^4 &= (\mathbf{x}^1 - \mathbf{x}^c) + (\mathbf{x}^4 - \mathbf{x}^c) + (\mathbf{x}^5 - \mathbf{x}^c)\\
\mathbf{p}^5 &= (\mathbf{x}^2 - \mathbf{x}^c) + (\mathbf{x}^5 - \mathbf{x}^c) + (\mathbf{x}^6 - \mathbf{x}^c)\\
\mathbf{p}^6 &= (\mathbf{x}^2 - \mathbf{x}^c) + (\mathbf{x}^3 - \mathbf{x}^c) + (\mathbf{x}^6 - \mathbf{x}^c)\\
\mathbf{p}^7 &= (\mathbf{x}^3 - \mathbf{x}^c) + (\mathbf{x}^4 - \mathbf{x}^c) + (\mathbf{x}^6 - \mathbf{x}^c)\\
\mathbf{p}^8 &= (\mathbf{x}^4 - \mathbf{x}^c) + (\mathbf{x}^5 - \mathbf{x}^c) + (\mathbf{x}^6 - \mathbf{x}^c)
\end{aligned}
$$

where the center of the domain is defined as

$$\mathbf{x}^c = \frac{\mathbf{x}^1 + \mathbf{x}^2 + \mathbf{x}^3 + \mathbf{x}^4 + \mathbf{x}^5 + \mathbf{x}^6}{6}$$

The computation of $\mathbf{p}^1$ is displayed in Figure 3.4. The idea now is to find the eight particles on the surface at t^n which, when moved by a pure Lagrangian motion, are located somewhere along these directional vectors at t^{n+1}.

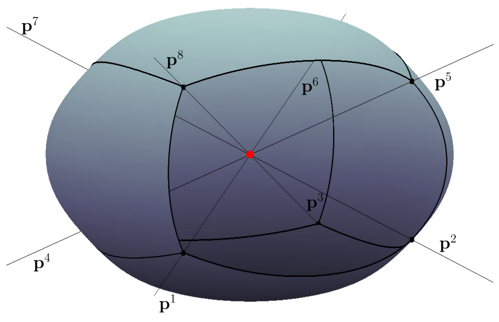


Figure 3.2: Starting point and directional vectors for the corner-update.

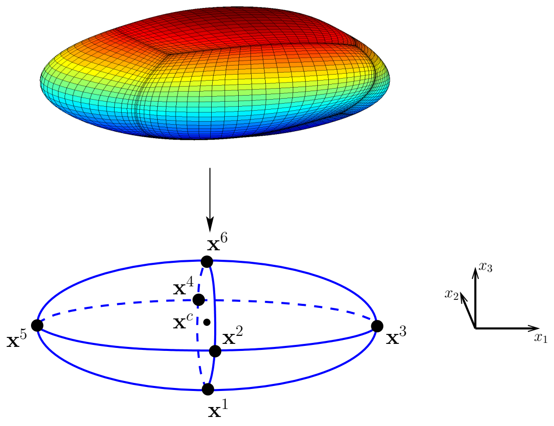


Figure 3.3: The six points determining the general shape of the surface. Here, $\mathbf{x}^c$ is the centre of the domain bounded by the closed surface, while $\mathbf{x}^1, \ldots, \mathbf{x}^6$ are the midpoints of the six surface elements.

3.2 Update of the points on the element edges

In the previous step we discussed the update of the corner points for each element; we now focus on how to update the grid-points along the element edges. Again, we want to construct starting points and directional vectors which determine the directions for the updated points. As starting points for each edge we choose to distribute points along the chord connecting the two relevant corner points computed in the previous step; see Figure 3.5. In order to determine the directional vectors we first construct a hexahedron from the 8 corner-points computed in the previous step. Next, we compute outer unit normals on each of the six faces of this hexahedron. Hence, for a point on the edge, we have a non-unique normal. The directional vector for a point is simply the average of the two normals computed on the opposing faces; see Figure 3.6. As before, we want to find the particle at t^n which, when moved through a Lagrangian motion, ends up somewhere along the direction vector associated with this particle.

3.3 Update of the interior points

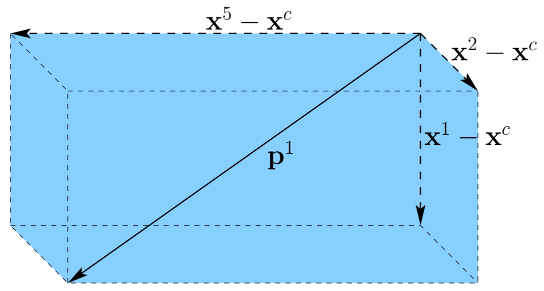
At this stage we have updated the boundary of each element, and what remains are the interior points. The overall strategy here is the same as for the other types of grid nodes; we need a starting point and a directional vector for each interior point. For the edges, we chose starting points along the chord connecting the two end-points. For a surface in three dimensions, however, the best way to do this is not obvious. We know the boundary of each element at t^{n+1}, and the idea now is to construct an intermediate surface which is solely determined from this known boundary information *and* which has a good distribution of grid points, i.e., the mapping from the reference domain to this new surface is smooth. The way we choose to do this here is by using the Gordon-Hall algorithm [7] on all three physical coordinates. At this stage we know the value of $\underline{\mathbf{x}}_{ij}^{n+1,k}$ on the boundary of each surface element (where at least one of i or j is 0 or N). The Gordon-Hall algorithm is then given by

$$\begin{aligned}
\underline{\mathbf{x}}_{GH}^{n+1,k} &= \underline{\mathbf{x}}^{n+1,k}, && \text{on } \partial\Gamma^{n+1,k},\\
\left(\mathbf{x}_{GH}^{n+1,k}\right)_{ij} &= \mathbf{x}_{ij}^{A,k} + \mathbf{x}_{ij}^{B,k} - \mathbf{x}_{ij}^{C,k}, && \text{for } 1 \le i,j \le N-1,
\end{aligned}$$

where

$$\begin{aligned}
\mathbf{x}_{ij}^{A,k} &= \frac{(1-\xi_i)}{2}\mathbf{x}_{0j}^{n+1,k} + \frac{(1+\xi_i)}{2}\mathbf{x}_{Nj}^{n+1,k},\\
\mathbf{x}_{ij}^{B,k} &= \frac{(1-\xi_j)}{2}\mathbf{x}_{i0}^{n+1,k} + \frac{(1+\xi_j)}{2}\mathbf{x}_{iN}^{n+1,k},\\
\mathbf{x}_{ij}^{C,k} &= \frac{(1-\xi_i)}{2}\frac{(1-\xi_j)}{2}\mathbf{x}_{00}^{n+1,k} + \frac{(1+\xi_i)}{2}\frac{(1-\xi_j)}{2}\mathbf{x}_{N0}^{n+1,k}\\
&\quad + \frac{(1-\xi_i)}{2}\frac{(1+\xi_j)}{2}\mathbf{x}_{0N}^{n+1,k} + \frac{(1+\xi_i)}{2}\frac{(1+\xi_j)}{2}\mathbf{x}_{NN}^{n+1,k}.
\end{aligned}$$

The collection of points $\left(\underline{\mathbf{x}}_{GH}^{n+1,k}\right)$ will then give us a surface which solely depends on the boundary-points which were found in the previous section. We will denote this surface as the "GH-surface". We assume that the shape of the "GH-surface" will not be too different from the actual surface which we want to approximate at t^{n+1}. In Figure 3.7 we have one



$\mathbf{x}^5 - \mathbf{x}^c$
$\mathbf{x}^2 - \mathbf{x}^c$
$\mathbf{x}^1 - \mathbf{x}^c$
$\mathbf{p}^1$

Figure 3.4: One of the directional vectors, $\mathbf{p}^1$, which determines the direction we want the first of eight corner points at t^{n+1} to be located along. The computation of $\mathbf{p}^1$ is based on points which indicates the *general shape* of the surface at the known t^n-configuration.

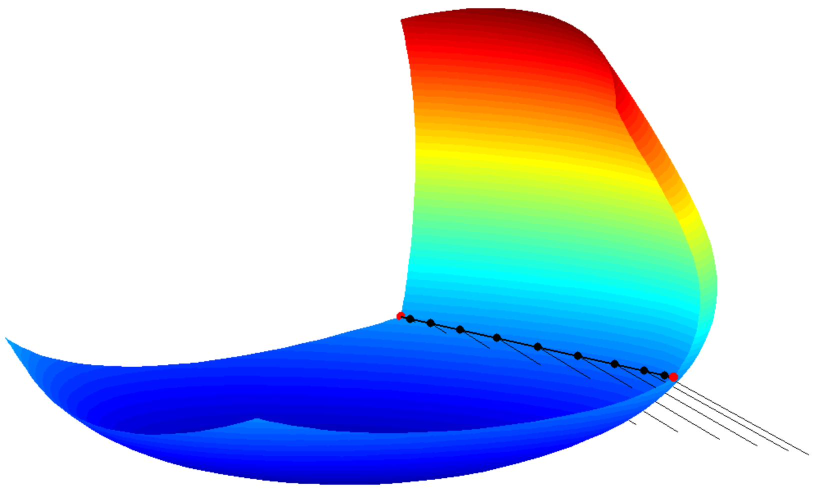


Figure 3.5: Based on the already updated corner points, we construct the chord between two corner points and use a GLL-distribution along this chord as starting points for the directional vectors.

example of a numerically represented surface and its associated “GH-surface”. However, it is the real surface at time t^{n+1} that we want to approximate well, so the larger the difference is between these two, the more important it is to have good directional vectors.

3.3.1 Directional vectors: Alternative 1

One alternative for each directional vector is to use an extension of the chord-distribution in two dimensions [1]. In two dimensions one way to do this was to distribute starting points along the chord connecting the end-points, and use the normal of this chord as a directional vector. Such a strategy would mean that the directional vector for each point would be the same, and only the starting points would differ. To extend this to three dimensions for a given point ij, we construct the two chords from the boundary. Both these chords have a normal plane, and as our directional vector we use the intersection of these two normal planes; see Figure 3.8.

3.3.2 Directional vectors: Alternative 2

As a second alternative we simply compute the normal on the “GH-surface” at each starting point, and use this normal as the directional vector.

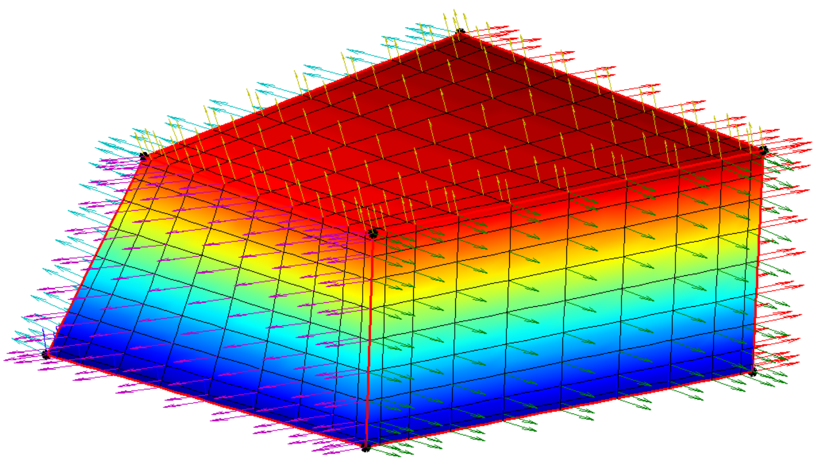
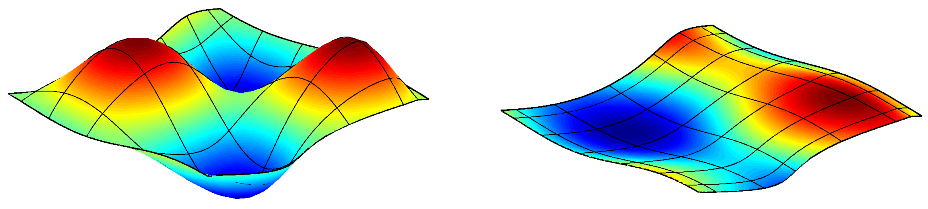


Figure 3.6: Unit normals on the faces of the hexahedron computed from the eight updated corner points. The directional vectors on each edge is the algebraic mean of the two normal vectors.



(a) Surface (b) GH-surface

Figure 3.7: An example of a surface (a) and the associated GH-surface (b) on one spectral element. Here the difference is large since the interior structure on the surface doesn't appear on the boundary.

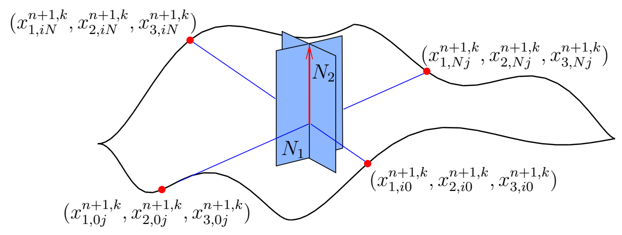

$(x_{1,iN}^{n+1,k}, x_{2,iN}^{n+1,k}, x_{3,iN}^{n+1,k})$
$(x_{1,Nj}^{n+1,k}, x_{2,Nj}^{n+1,k}, x_{3,Nj}^{n+1,k})$
N_2
N_1
$(x_{1,i0}^{n+1,k}, x_{2,i0}^{n+1,k}, x_{3,i0}^{n+1,k})$
$(x_{1,0j}^{n+1,k}, x_{2,0j}^{n+1,k}, x_{3,0j}^{n+1,k})$

Figure 3.8: Computation of the directional vector for the grid point with indices ij on element k using alternative 1.

4 Implementation

In the previous section we discussed possible ways of updating the grid from t^n to t^{n+1}. This resulted in directions in which we wanted to update each grid node. Next, the idea is to find a particle at the surface at time t^n which when moved by a pure Lagrangian motion ends up along this direction at time t^{n+1}. For a given particle at t^n, the position at t^{n+1} is found by integrating

$$\frac{\mathrm{d}\mathbf{x}}{\mathrm{d}t} = \mathbf{u} \tag{4.1}$$

from t^n to t^{n+1}. Here, we will use Adams-Bashforth schemes for the integration of (4.1); e.g., the position of a given particle at t^{n+1} is given by

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \left(\frac{3}{2}\mathbf{u}^n - \frac{1}{2}\mathbf{u}^{n-1} \right), \tag{4.2}$$

when using a second order scheme. Hence, for a given particle we need its position at t^n, its velocity at t^n and its velocity at t^{n-1}. The relevant fields are available through the polynomial representations (2.4). One choice (k, ξ_1^*, ξ_2^*) at t^n corresponds to one particle at this time level, and the position and velocity for this particle is available through $(x_i)^{n,k}(\xi_1^*, \xi_2^*)$ and $(u_i)^{n,k}(\xi_1^*, \xi_2^*)$, $i = 1, 2, 3$, respectively. However, since the velocity field $(u_i)^{n-1}$ is represented on the geometry at time level t^{n-1} we don't have the velocity at t^{n-1} easily available for this particular particle since (k, ξ_1^*, ξ_2^*) at t^{n-1} in general corresponds to a different particle. Hence, if we use $(u_i)^{n-1,k}(\xi_1^*, \xi_2^*)$ in (4.2) we will *not* be able to follow the surface at second order precision, and can at best hope for first order accuracy. With this in mind, it would be convenient to have representations of the velocity components $(u_i)^{n-1}$ at the t^n geometry configuration. If we denote this representation $\widetilde{u}_i^n$, $(u_i)^{n,k}(\xi_1^*, \xi_2^*)$ and $(\widetilde{u}_i)^{n,k}(\xi_1^*, \xi_2^*)$ will correspond to the ith velocity component at t^n and t^{n-1}, respectively, for the particle corresponding to (k, ξ_1^*, ξ_2^*) at t^n. In the next section we will discuss how these representations $\widetilde{u}_i^n$ can be computed.

4.1 Finding a fluid particle's earlier velocities

We will base our strategy on the idea presented in [14] and used for time-dependent surfaces in two space dimensions in [1]. This strategy involves integrating surface convection problems, which take the form

$$\begin{aligned} \frac{\partial \varphi}{\partial \tau} + \mathbf{u} \cdot \nabla_s \varphi &= 0, \qquad && \text{on } \Gamma(t^* + \tau), \\ \varphi &= \varphi_0, && \text{on } \Gamma(t^*). \end{aligned} \tag{4.3}$$

Here, Γ is the time-dependent interface, t^* will either be t^{n-1} or t^{n-2}, and ∇_s is the surface gradient. The ALE-formulation of this problem reads: find $\varphi \in X$ such that

$$\begin{aligned} &\frac{\mathrm{d}}{\mathrm{d}\tau}(\varphi, v) + c(\varphi, v) = 0, && \forall v \in X, \\ &\varphi|_{\Gamma^k}(\xi_1, \xi_2; \tau = 0) = \varphi_0^k(\xi_1, \xi_2), \qquad && k = 1, \ldots, K, \end{aligned} \tag{4.4}$$

where X is an appropriate function space and

$$(\varphi, v) = \sum_{k=1}^{K} \int_{\hat{\Gamma}} \varphi v (J^s)^k \, \mathrm{d}\hat{\Gamma}, \tag{4.5}$$

$$c(\varphi, v) = \sum_{k=1}^{K} \left(\int_{\hat{\Gamma}} v(\mathbf{u} - \mathbf{w}) \cdot \nabla_s \varphi (J^s)^k \, \mathrm{d}\hat{\Gamma} - \int_{\hat{\Gamma}} v\varphi \frac{\mathrm{d}(J^s)^k}{\mathrm{d}\tau} \, \mathrm{d}\hat{\Gamma} \right). \tag{4.6}$$

Here,

$$\frac{\mathrm{d}}{\mathrm{d}\tau} = \frac{\partial}{\partial \tau} + \mathbf{w} \cdot \nabla_s.$$

To express the other terms we need the following definitions

$$\begin{aligned}
\mathbf{w}_{t_1} &= \left(\frac{\partial w_1}{\partial \xi_1}, \frac{\partial w_2}{\partial \xi_1}, \frac{\partial w_3}{\partial \xi_1} \right), \\
\mathbf{w}_{t_2} &= \left(\frac{\partial w_1}{\partial \xi_2}, \frac{\partial w_2}{\partial \xi_2}, \frac{\partial w_3}{\partial \xi_2} \right), \\
\mathbf{g}_1 &= \left(\frac{\partial x_1}{\partial \xi_1}, \frac{\partial x_2}{\partial \xi_1}, \frac{\partial x_3}{\partial \xi_1} \right), \\
\mathbf{g}_2 &= \left(\frac{\partial x_1}{\partial \xi_2}, \frac{\partial x_2}{\partial \xi_2}, \frac{\partial x_3}{\partial \xi_2} \right), \\
g_{\alpha\beta} &= \mathbf{g}_\alpha \cdot \mathbf{g}_\beta, \\
\mathbf{g}_\alpha \cdot \mathbf{g}^\beta &= \delta_{\alpha\beta}, \\
g &= \sqrt{g_{11} g_{22} - g_{12}^2}.
\end{aligned} \tag{4.7}$$

Then,

$$\begin{aligned}
J^s &= g, \\
\nabla_s \varphi &= \frac{\partial \varphi}{\partial \xi_\alpha} \mathbf{g}^\alpha, \\
\frac{\mathrm{d}J^s}{\mathrm{d}\tau} &= \frac{(\mathbf{g}_1 \cdot \mathbf{w}_{t_1}) g_{22} + (\mathbf{g}_2 \cdot \mathbf{w}_{t_2}) g_{11} - (\mathbf{g}_1 \cdot \mathbf{w}_{t_2} + \mathbf{g}_2 \cdot \mathbf{w}_{t_1}) g_{12}}{g},
\end{aligned}$$

where $\alpha = 1, 2$ and summation over repeated indices is assumed.

We discretize the convection problem (4.4) using a standard spectral element method in space and arrive at the following set of ordinary differential equations:

$$\begin{aligned}
\frac{\mathrm{d}\underline{B}^s \underline{\varphi}}{\mathrm{d}\tau} &= -\underline{C}^s \underline{\varphi}, \\
\underline{\varphi}(\tau = 0) &= \underline{\varphi}_0.
\end{aligned} \tag{4.8}$$

Here, $\underline{B}^s$ is the surface mass matrix, $\underline{C}^s$ is the convection operator and $\underline{\varphi}$ is a global vector comprising this fields nodal values on all the surface elements. For a second order scheme, we need to solve (4.8) three times, with initial conditions $\underline{\varphi}_0 = \underline{u}_i^{n-1}$, $i = 1, 2, 3$, and integration domain $(0, \Delta t)$. For a third order scheme we need to solve six such problems with initial conditions $\underline{\varphi}_0 = \underline{u}_i^{n-1}$ and $\underline{\varphi}_0 = \underline{u}_i^{n-2}$, $i = 1, 2, 3$, and integration domains $(0, \Delta t)$ and $(0, 2\Delta t)$, respectively. We denote the output as $\underline{\tilde{u}}_i^n$ for the problems with integration domain $(0, \Delta t)$ and $\underline{\tilde{\tilde{u}}}_i^n$ for the problems with integration domain $(0, 2\Delta t)$. In this work we use the classical explicit fourth order Runge-Kutta method for solving this problem.

4.2 Finding the correct particle

At this stage we have developed a general strategy for how we want the grid-nodes to move, which resulted in starting points and directional vectors, and the idea is to search for the particle at t^n which ends up at the desired location (dictated by these vectors) at t^{n+1} by a pure Lagrangian motion. The assumption that we have a surface which doesn't "fold over" with respect to these vectors ensures that this problem has a unique solution. Using a second order scheme as an example we want to find the particle such that $\mathbf{x}^{n+1}$ from (4.2) is a valid location for each grid node. By solving three convection problems as described in the previous section we were able to represent $\mathbf{u}^{n-1}$ on the t^n-geometry. Hence, $\mathbf{x}^{n,k}(\xi_1^*, \xi_2^*)$, $\mathbf{u}^{n,k}(\xi_1^*, \xi_2^*)$ and $\widetilde{\mathbf{u}}^{n,k}(\xi_1^*, \xi_2^*)$ now corresponds to the position at t^n, the velocity at t^n and the velocity at t^{n-1}, respectively, for the unique particle at element k with reference coordinates (ξ_1^*, ξ_2^*). This is all the information we need, and the remaining task is now to find the correct particle, represented by the element number and the location on the reference domain.

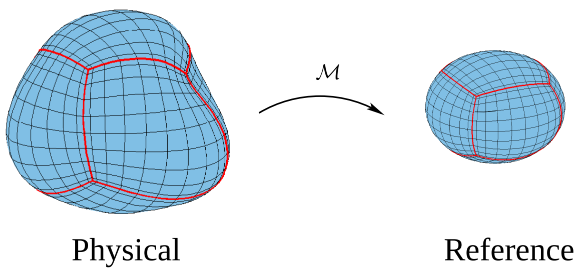
4.2.1 The mono-domain case

When we have a problem which is modelled by a single spectral element and the system is closed such that no "surface particles" enter or leave the domain, we may deal with this by either a direct search for a set (ξ_1^*, ξ_2^*) *or* extending the idea from [1] by solving one-dimensional convection problems in multiple directions.

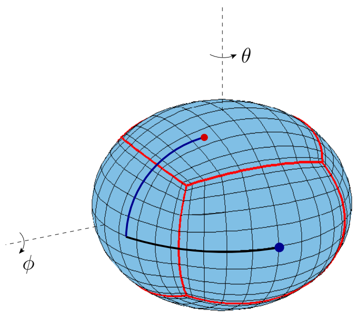
4.2.2 The multi-domain case

In the multi-element case, the situation becomes more complicated due to the fact that particles may be located on one element at t^n and on a different one at t^{n+1}, and this is something we need to be able to deal with. We still consider the six-element configuration which can be viewed as a deformed sphere. In our search we want to be able to access all surface particles on the deformed surface. The first thing we do is construct a regular *reference sphere* with the same elemental structure and polynomial degree. Here, all the elements have the same shape and size and we have control over the location of the elemental boundaries. Next, we map the relevant fields from the (deformed) physical surface to the reference sphere by simply representing each nodal value on the physical domain to the corresponding node on the reference sphere; see Figure 4.1(a). Hence, we may now search for the surface particle which satisfies our requirements on the reference sphere instead of the (deformed) physical domain. In particular, we define a starting point on the reference sphere, and from this starting point we first rotate θ radians in the latitude direction and next ϕ radians in the longitude direction, and our search consists of finding the unique set (θ^*, ϕ^*) $(0 \leq \theta^*, \phi^* \leq 2\pi)$ which satisfies our criteria; see Figure 4.1(b). In this way we are able to access all the points on the sphere, and from a given (θ^*, ϕ^*) we are able to map this to a corresponding (k, ξ_1^*, ξ_2^*) due to the known shape of the reference sphere.

From the resulting (k, ξ_1^*, ξ_2^*) we may then find the relevant position and velocities from the expansions (2.4).



(a)



(b)

Figure 4.1: (a) The mapping, $\mathcal{M}$, from the physical domain to the reference sphere. (b) An illustration of the search strategy on the reference sphere. Here, the blue circle is our starting point, and the "correct" particle represented by the red circle is found by rotations in both the latitude and longitude direction.

4.3 Algorithm

The algorithm for a first order scheme for the update from t^n to t^{n+1} is given in Algorithm 1, the second order version is given in Algorithm 2, while the third order version is given in Algorithm 3.

Equations which will be referenced in the algorithms:

First order:

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{u}^n, \tag{4.9}$$

$$\underline{\mathbf{w}}^n = \frac{\underline{\mathbf{x}}^{n+1} - \underline{\mathbf{x}}^n}{\Delta t}, \tag{4.10}$$

Second order:

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \left(\frac{3}{2}\mathbf{u}^n - \frac{1}{2}\widetilde{\mathbf{u}}^n \right), \tag{4.11}$$

$$\underline{\mathbf{w}}^n = \frac{2}{3}\left(\frac{\underline{\mathbf{x}}^{n+1} - \underline{\mathbf{x}}^n}{\Delta t} \right) + \frac{1}{3}\underline{\mathbf{w}}^{n-1}, \tag{4.12}$$

Third order:

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \left(\frac{23}{12}\mathbf{u}^n - \frac{4}{3}\widetilde{\mathbf{u}}^n + \frac{5}{12}\widetilde{\widetilde{\mathbf{u}}}^n \right), \tag{4.13}$$

$$\underline{\mathbf{w}}^n = \frac{12}{23}\left(\frac{\underline{\mathbf{x}}^{n+1} - \underline{\mathbf{x}}^n}{\Delta t} \right) + \frac{16}{23}\underline{\mathbf{w}}^{n-1} - \frac{5}{23}\underline{\mathbf{w}}^{n-2}. \tag{4.14}$$

Algorithm 1 First order algorithm for the update $\underline{\mathbf{x}}^n$ to $\underline{\mathbf{x}}^{n+1}$

```
Input: $\underline{\mathbf{x}}^n, \underline{\mathbf{u}}^n, \underline{\mathbf{w}}^{n-1}$
  1. Set $\underline{\mathbf{w}}^n = \underline{\mathbf{w}}^{n-1}$. This corresponds to a zeroth order extrapolation.
  2. Compute starting points and directional vectors for the corners as described in Section 3.1
  for All 8 corners do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.9) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  3. Compute starting points and directional vectors for the element edges as described in Section 3.2
  for All interior points on all 12 edges do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.9) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  4. Compute starting points and directional vectors for the interior points on each element as described in Section 3.3
  for All interior points on all 6 elements do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.9) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  5. Update $\underline{\mathbf{w}}^n$ according to (4.10).
```

Algorithm 2 Second order algorithm for the update $\underline{\mathbf{x}}^n$ to $\underline{\mathbf{x}}^{n+1}$

```
Input: $\underline{\mathbf{x}}^n, \underline{\mathbf{x}}^{n-1}, \underline{\mathbf{u}}^n, \underline{\mathbf{u}}^{n-1}, \underline{\mathbf{w}}^{n-1}, \underline{\mathbf{w}}^{n-2}$
  1. Set $\underline{\mathbf{w}}^n = 2\underline{\mathbf{w}}^{n-1} - \underline{\mathbf{w}}^{n-2}$. This corresponds to a first order extrapolation.
  2. Integrate (4.8) from $(0, \Delta t)$ three times with initial conditions $\underline{\varphi}_0 = \underline{u}_i^{n-1}$, $i = 1, 2, 3$ by using the classical RK4-method. Use first order polynomial interpolation for $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{w}$ in $(0, \Delta t)$, where $(0, \Delta t)$ corresponds to $(t^{n-1}, t^n)$ in "real time" $\Rightarrow \underline{\widetilde{\mathbf{u}}}^n$.
  3. Compute starting points and directional vectors for the corners as described in Section 3.1
  for All 8 corners do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.11) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  4. Compute starting points and directional vectors for the element edges as described in Section 3.2
  for All interior points on all 12 edges do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.11) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  5. Compute starting points and directional vectors for the interior points on each element as described in Section 3.3
  for All interior points on all 6 elements do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.11) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  6. Update $\underline{\mathbf{w}}^n$ according to (4.12).
```

Algorithm 3 Third order algorithm for the update $\underline{\mathbf{x}}^n$ to $\underline{\mathbf{x}}^{n+1}$

```
Input: $\underline{\mathbf{x}}^n, \underline{\mathbf{x}}^{n-1}, \underline{\mathbf{x}}^{n-2}, \underline{\mathbf{u}}^n, \underline{\mathbf{u}}^{n-1}, \underline{\mathbf{u}}^{n-2}, \underline{\mathbf{w}}^{n-1}, \underline{\mathbf{w}}^{n-2}, \underline{\mathbf{w}}^{n-3}$
  1. Set $\underline{\mathbf{w}}^n = 3\underline{\mathbf{w}}^{n-1} - 3\underline{\mathbf{w}}^{n-2} + \underline{\mathbf{w}}^{n-3}$. This corresponds to a second order extrapolation.
  2. Integrate (4.8) from $(0, \Delta t)$ three times with initial conditions $\underline{\varphi}_0 = \underline{u}_i^{n-1}$, $i = 1, 2, 3$ by using the classical RK4-method. Use second order polynomial interpolation for $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{w}$ in $(0, \Delta t)$, where $(0, \Delta t)$ corresponds to $(t^{n-1}, t^n)$ in "real time" $\Rightarrow \widetilde{\underline{\mathbf{u}}}^n$. Integrate (4.8) from $(0, 2\Delta t)$ three times with initial conditions $\underline{\varphi}_0 = \underline{u}_i^{n-2}$, $i = 1, 2, 3$ by using the classical RK4-method. Use second order polynomial interpoltion for $\mathbf{x}$, $\mathbf{u}$ and $\mathbf{w}$ in $(0, 2\Delta t)$, where $(0, 2\Delta t)$ corresponds to $(t^{n-2}, t^n)$ in "real time" $\Rightarrow \widetilde{\widetilde{\underline{\mathbf{u}}}}^n$.
  3. Compute starting points and directional vectors for the corners as described in Section 3.1
  for All 8 corners do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.13) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  4. Compute starting points and directional vectors for the element edges as described in Section 3.2
  for All interior points on all 12 edges do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.13) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  5. Compute starting points and directional vectors for the interior points on each element as described in Section 3.3
  for All interior points on all 6 elements do
    Find the set $(\theta^*, \phi^*)$ which corresponds to the particle such that $\mathbf{x}^{n+1}$ from (4.13) is located in the desired direction. This search for $(\theta^*, \phi^*)$ involves a Newton iteration.
  end for
  6. Update $\underline{\mathbf{w}}^n$ according to (4.14).
```

4.4 Computational complexity

We will here briefly discuss the cost of the new algorithm when we consider a deformed sphere represented by six elements. Step 2 in Algorithm 2 and Algorithm 3 involves solving problems of the type (4.8), which scales like $\mathcal{O}(N^3)$ when exploiting that the operators are based on tensor-product forms [4]. The main bottleneck is however the steps 3., 4. and 5. in the algorithms which involve searching for the correct particle. For each grid-node we wish to find (θ^*, φ^*) which satisfies our criterion, and at each iteration the guess (θ, φ) corresponds to an element number k and reference values (ξ_1, ξ_2). Hence, we need to make several evaluations on the form

$$\varphi(\xi_1, \xi_2) = \sum_{m,n=0}^{N} \varphi_{mn} \ell_m(\xi_1) \ell_n(\xi_2). \tag{4.15}$$

This can be done rather efficiently by first evaluating

$$\ell_j(\xi_1), \quad \ell_j(\xi_2), \qquad j = 0, \ldots, N. \tag{4.16}$$

The computation of $\ell_j(\xi)$ requires about $2N$ operations, hence (4.16) requires approximately $4N^2$ operations. By setting

$$L_{ij} = \ell_i(\xi_1)\ell_j(\xi_2), \qquad 0 \le i, j \le N,$$

the evaluation (4.15) corresponds to

$$\varphi(\xi_1, \xi_2) = \sum_{m,n=0}^{N} \varphi_{mn} L_{mn}, \tag{4.17}$$

at the cost of approximately $2N^2$. For a second order scheme we need to do 9 such evaluations at each iteration (3 fields $\mathbf{x}^n$, $\mathbf{u}^n$, $\widetilde{\mathbf{u}}^n$ with three components each), while for a third order scheme we need 12. We need to do this for approximately $6N^2$ points, hence, in total we get

$$\begin{aligned} \textbf{Second order}: &\quad \mathcal{N}_{it} \cdot 6N^2 \cdot (18N^2 + 4N^2) = 132\mathcal{N}_{it}N^4, \\ \textbf{Third order}: &\quad \mathcal{N}_{it} \cdot 6N^2 \cdot (24N^2 + 4N^2) = 168\mathcal{N}_{it}N^4, \end{aligned}$$

where $\mathcal{N}_{it}$ is the average number of iterations needed in Newton's method. In our simulations we typically achieve convergence in about three iterations.

5 Numerical results

In this section we will present numerical results where we compare our new strategy with other strategies. The main motivation behind this work is to solve the three-dimensional Navier-Stokes equations with a free surface, however, it is difficult to construct a test-problem which satisfies these equations and where the evolution of the surface also depends on the time-dependent solution of the Navier-Stokes equations. Instead, we will here only focus on the step of updating the geometry in such a solver and we will assume that we "release" the surface in a known velocity field (which in a real case should come from the Navier-Stokes solver at each time-step). These numerical tests are constructed by first deciding the evolution of the *exact* front, and next computing a velocity field which satisfies this evolution. Finally, we may add suitable tangential components to this velocity field which doesn't alter the *shape* of the front but may alter the motion of the fluid particles. From now on we will only use the known velocity-field in our strategies (unless stated otherwise) and the known shape of the exact front at a given time, T, is only used in the error computation.

In order to determine the quality of the numerically computed surface at a given time, T, we have two error measures:

1. Error "hitting the front"

   For each grid node we compute the distance to the exact front, and the reported error is the maximum over all grid nodes; see Figure 5.1. Hence, the quality of the grid doesn't come into consideration for this measure.

2. Error computing the surface area

   We compute the surface area from the numerically computed front, and compare this with the surface area of the exact front, S_e. The surface area is given by:

$$S_e = \int \mathrm{d}S,$$

   and the numerically computed area of our front is given by

$$S_n = \sum_{k=1}^{K} \sum_{i,j=0}^{N} w_i w_j g_{ij}^k,$$

   where w_i is the ith GLL-weight and g_{ij}^k corresponds to the quantity in (4.7). The reported error is $|S_e - S_n|$.

5.1 Test problem 1: a single element

Here we compare the new strategy on a front discretized by one spectral element which is released in a velocity field. The problem is constructed such that a motion of the grid nodes on the boundary of the domain through a pure Lagrangian motion yields a good point distribution. We compare the motion of the interior grid points by using a pure Lagrangian strategy and the new strategy with alternative 1 for the update of the interior points. In

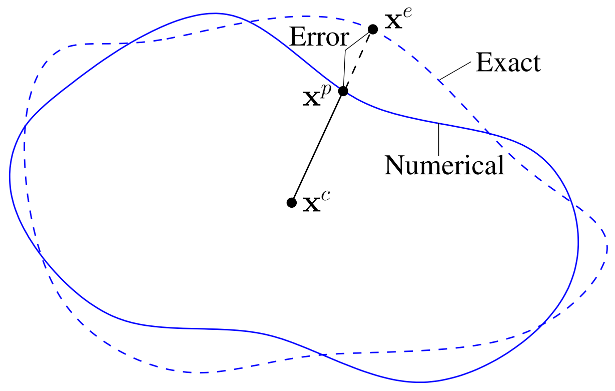
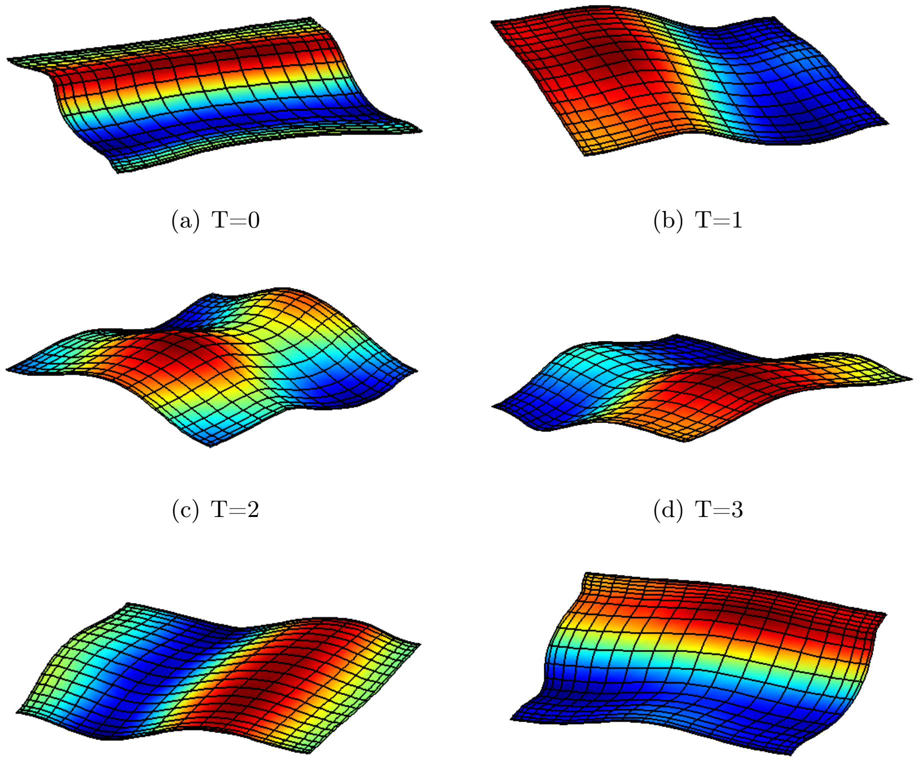


Figure 5.1: A two-dimensional projection of the exact and numerically approximated surface to a plane containing the vector $\mathbf{x}^p - \mathbf{x}^c$. Here, $\mathbf{x}^c$ is the centre of the deformed sphere, $\mathbf{x}^p$ is a point on the numerical surface and $\mathbf{x}^e$ is the corresponding point on the exact surface. The error "hitting the front" is defined as the distance between $\mathbf{x}^p$ and $\mathbf{x}^e$.

Figures 5.2 and 5.3 we see the numerically approximated surface at a few snapshots during the simulation for the new strategy and the Lagrangian strategy, respectively. Figures 5.4 and 5.5 display the error for both these approaches for first, second, and third order schemes. We see that both approaches are able to achieve the expected accuracy in terms of hitting the front, however, due to the distorted grid we get a large error in terms of computing the area when using the Lagrangian approach. With the new strategy the error decays with the expected rate for the first, second, and third order scheme. Here, we also note that the spatial error with $N = 18$ seems to be about 10^{-5}.

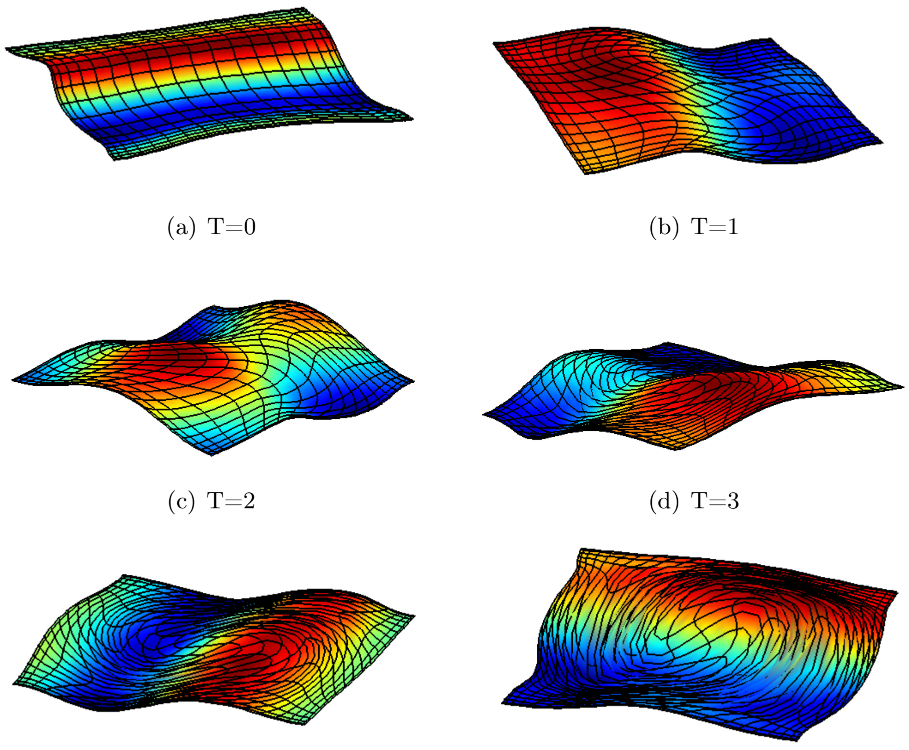


(a) T=0 (b) T=1

(c) T=2 (d) T=3

(e) T=4 (f) T=5

Figure 5.2: The front represented at selected times using the new strategy to solve test problem 1.

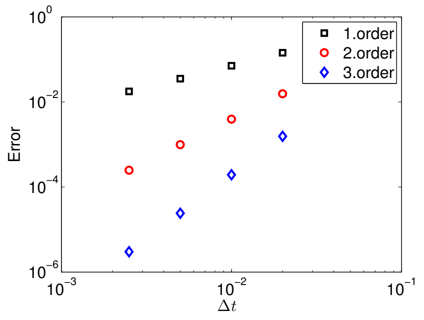


(a) T=0 (b) T=1

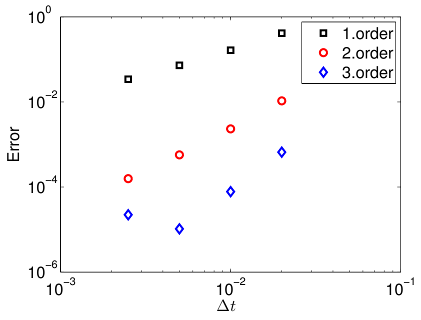
(c) T=2 (d) T=3

(e) T=4 (f) T=5

Figure 5.3: The front represented at selected times using a Lagrangian strategy to solve test problem 1.

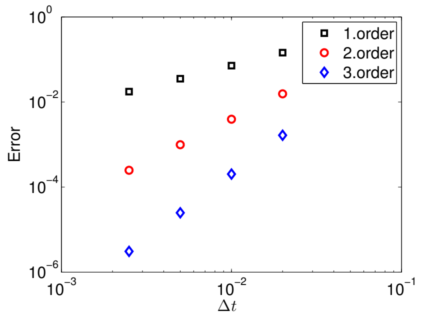


(a) Error front

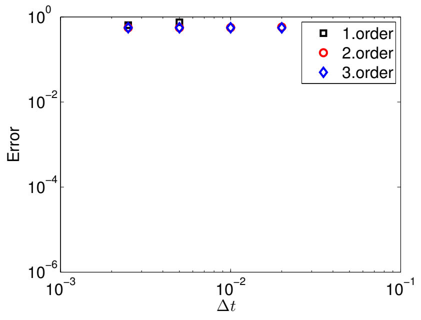


(b) Error area

Figure 5.4: The error for test problem 1 when using the new strategy with Alternative 1 with $N = 18$ and $T = 5$.



(a) Error front



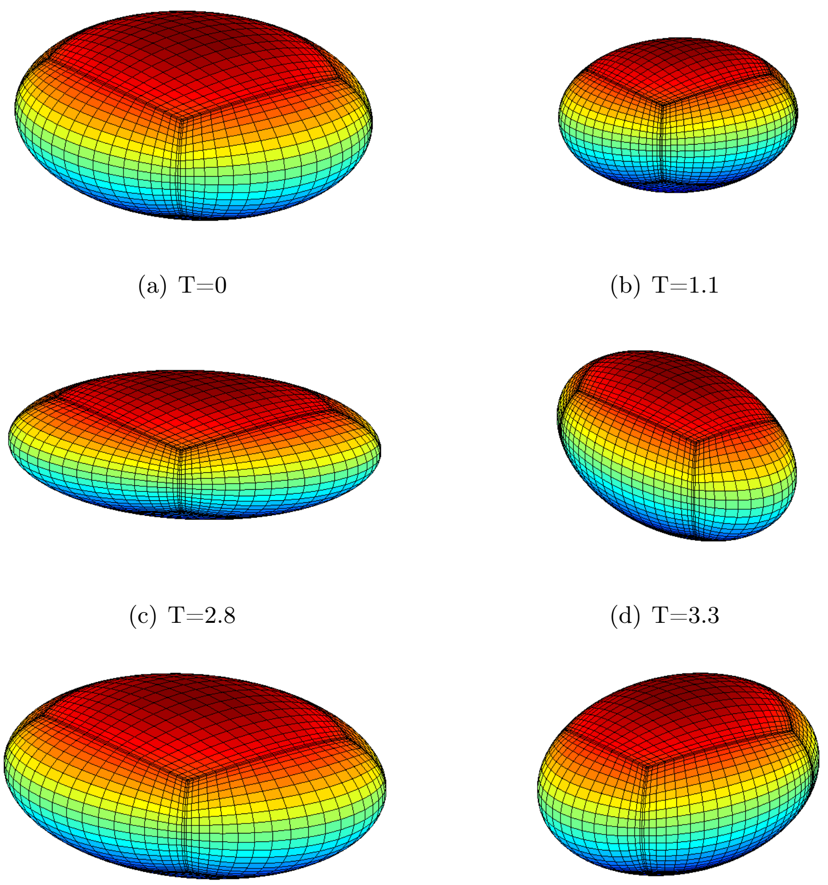
(b) Error area

Figure 5.5: The error for test problem 1 when moving the grid-nodes in a pure Lagrangian motion with $N = 18$ and $T = 5$.

5.2 Test problem 2 - a closed surface

Next, we move to a closed surface which is represented by six spectral elements; the initial surface is released in a known velocity field. The process of updating the grid from t^n to t^{n+1} using the new strategy involves using Algorithms 1, 2 and 3 for the first, second and third order approach. Here, alternative 2 is used for updating the interior grid-points on all 6 elements. In this example we compare the new strategy with a strategy where all the grid-points are moved in a pure Lagrangian fashion. We also compare with a strategy where all the grid-points are moved in the normal direction, which is a common strategy in the ALE-setting. Computing the normals based on the numerically represented surface, however, makes the simulation break down, so here we compute the normals from the analytically known surface. This is obviously something we usually won't have access to in a real simulation.

In Figure 5.6 we see that we are able to maintain a good grid quality during the simulation, and we are able to achieve first, second and third order accuracy both in terms of "hitting the front" and when measuring the surface area. In Figure 5.7 we have used a pure Lagrangian motion for all grid-nodes. Now the grid quality is totally dependent on the velocity field, and we see that the grid becomes heavily distorted due to a substantial velocity component in the tangential directions. In Figure 5.8 we have the evolution of the grid when we are using a normal strategy where all grid nodes are only moved in the normal direction, and we see that the grid becomes somewhat distorted. In Figures 5.9, 5.10 and 5.11 we have the error at $T = 16$ with $N = 21$ for the three approaches. All three approaches behave as expected in regards to hitting the front. In terms of the approximated surface area, we get the expected behaviour for the new strategy and the Lagrangian strategy. We also note that the normal strategy here actually behaves better than the new strategy, so in princible a motion in the normal direction doesn't seem like a bad idea for this example. However, as noted earlier we have used the analytically known surface to compute the normals in this case. Computing the normals from the numerically approximated surface is here unstable and eventually leads the simulation to break down. We may also note that if we had replaced $\underline{\widetilde{\mathbf{u}}}$ with $\underline{\mathbf{u}}$ in Algorithm 3, we would achieve first order accuracy with a slightly better constant than the overall first order scheme described in Algorithm 1 for this test-problem.


(a) T=0

(b) T=1.1

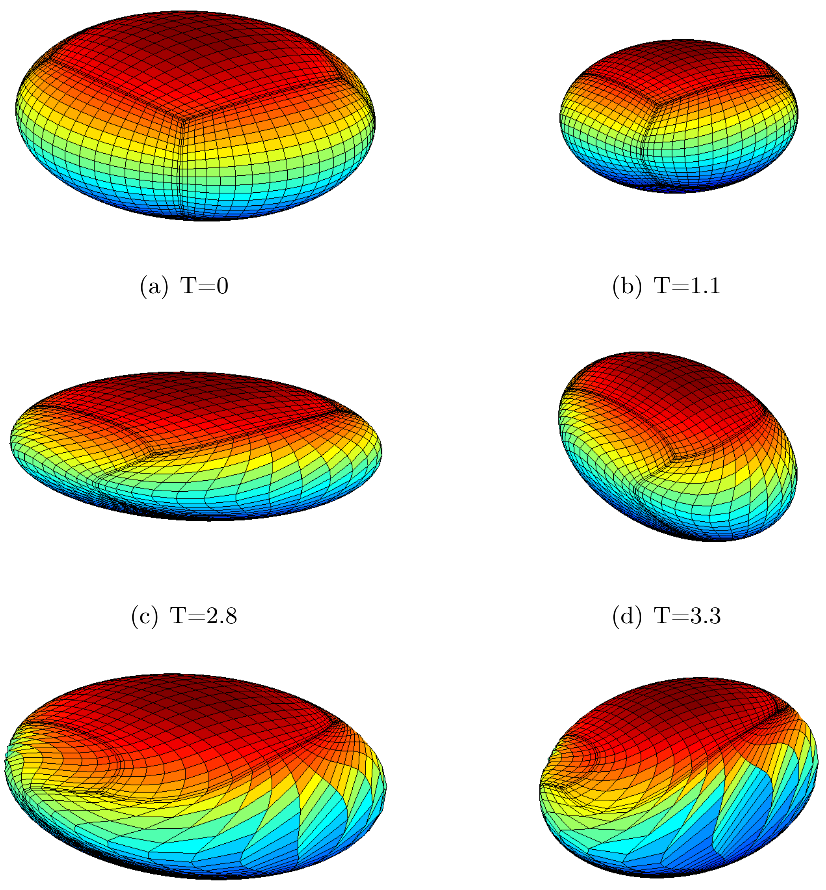
(c) T=2.8

(d) T=3.3

(e) T=6

(f) T=8

Figure 5.6: The front representation at selected times using the new strategy to solve test problem 2.


(a) T=0

(b) T=1.1

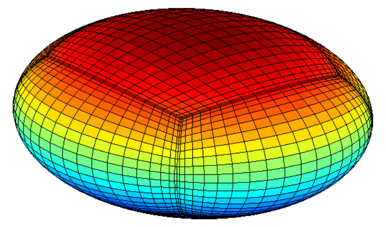
(c) T=2.8

(d) T=3.3

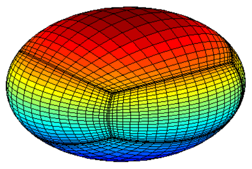
(e) T=6

(f) T=8

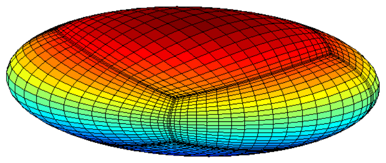
Figure 5.7: The front representation at selected times using the Lagrangian strategy to solve test problem 2.



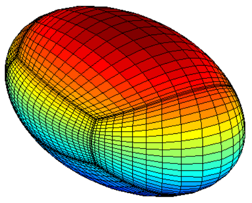
(a) T=0



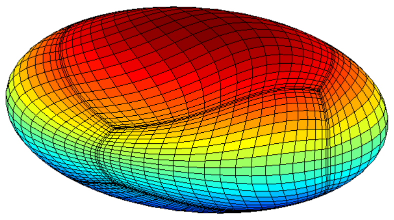
(b) T=1.1



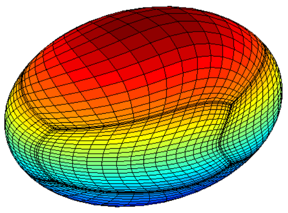
(c) T=2.8



(d) T=3.3

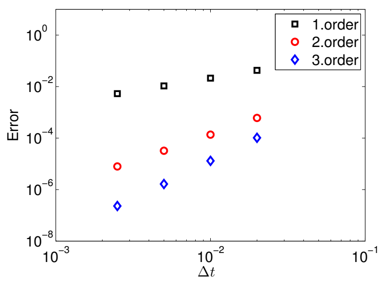


(e) T=6

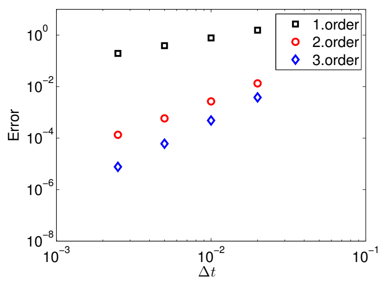


(f) T=8

Figure 5.8: The front representation at selected times using the normal strategy with analytical normals to solve test problem 2.

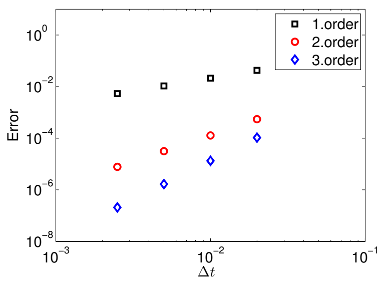


(a) Error front

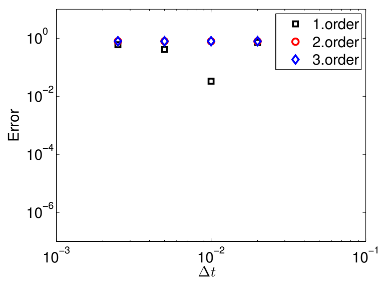


(b) Error area

Figure 5.9: The error for test problem 2 when using the new strategy with Alternative 2 with $N = 21$ and $T = 16$.

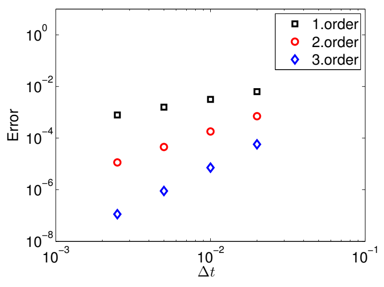


(a) Error front

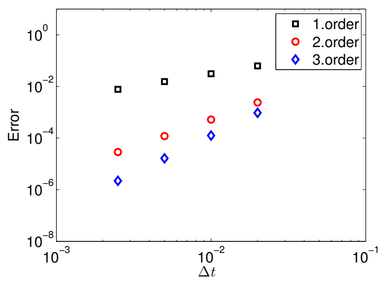


(b) Error area

Figure 5.10: The error for test problem 2 when moving the grid-nodes in a pure Lagrangian motion with $N = 21$ and $T = 16$.



(a) Error front



(b) Error area

Figure 5.11: The error for test problem 2 when moving the grid-nodes in the normal direction (where the normals are analytically computed) with $N = 21$ and $T = 16$.

6 Conclusions

We have extended the computational approach presented in [1] for tracking an interface immersed in a given velocity field to three spatial dimensions. The proposed method is particularly relevant to the simulation of unsteady free surface problems using the arbitrary Lagrangian-Eulerian framework, and has been constructed with two goals in mind: (i) to be able to accurately follow the interface; and (ii) to automatically maintain a good distribution of the grid points along the interface. The method combines information from a pure Lagrangian approach with information from an ALE approach.

We have been able to construct three-dimensional model problems allowing us to quantify the discretization error incurred. In particular, it has allowed us to verify and compare the temporal accuracy of different methods: the new approach, a pure Lagrangian approach, and an approach honoring the kinematic condition in the normal direction, but imposing a homogeneous Dirichlet condition for the tangential component of the grid velocity (called the Normal approach). Using the new approach we have been able to achieve both of our primary objectives; we have verified first, second, and third order temporal accuracy for the two model problems considered here. The new method is particularly important in the context of using high order spatial discretization schemes. Both the Lagrangian approach and the Normal approach generally give a non-optimal point distribution along the interface, something which again may result in large errors in the computation of important surface quantities (e.g., normal and tangent vectors, local curvature, length etc.). Such errors may, in the worst case, result in a complete breakdown of the interface-tracking.

In the two test problems considered in this study, we use a single spectral element (an open surface) and multiple spectral elements (a closed surface), respectively. The polynomial approximation in each spatial direction on the reference element is N. Hence, the number of nodal points on a three-dimensional surface is $\mathcal{O}(N^2)$. The complexity of the proposed method for tracking the evolving surface is $\mathcal{O}(N^4)$, which may appear expensive. However, in the context of solving three-dimensional Navier-Stokes problems using spectral elements, the number of unknowns will scale as $\mathcal{O}(N^3)$, and the computational cost for a single (spatial) operator evaluation will scale as $\mathcal{O}(N^4)$ [13]. Hence, using the proposed method in the context of solving such problems, the added computational cost will be modest.

Acknowledgment

The work has been supported by the Research Council of Norway under Contract 185336/V30. The support is gratefully acknowledged.

References

[1] T. Bjøntegaard and E.M. Rønquist. Accurate interface tracking for arbitrary Lagrangian-Eulerian schemes. *Journal of Computational Physics*, 228(12):4379–4399, 2009.

[2] R. Bouffanais and M.O. Deville. Mesh update techniques for free-surface flow solvers using spectral elements. *Journal of Scientific Computing*, 27(1-3):137–149, 2006.

[3] W. Dettmer and D. Perić. A computational framework for free surface fluid flows accounting for surface tension. *Computer Methods in Applied Mechanics and Engineering*, 195:3038–3071, 2006.

[4] M.O. Deville, P.F. Fischer, and E.H. Mund. *High-Order Methods for Incompressible Fluid Flow.* Cambridge University Press, 2002.

[5] J. Donea, S. Giuliani, and J.P Halleux. An arbitrary Lagrangian-Eulerian finite element method for transient dynamic fluid-structure interactions. *Computer Methods in Applied Mechanics and Engineering*, 33:689–723, 1982.

[6] F. Duarte, R. Gormaz, and S. Natesan. Arbitrary Lagrangian-Eulerian method for Navier-Stokes equations with moving boundaries. *Computer Methods in Applied Mechanics and Engineering*, 193:4819–4836, 2004.

[7] W.J. Gordon and C.A. Hall. Construction of curvilinear co-ordinate systems and applications to mesh generation. *International Journal for Numerical Methods in Engineering*, 7:461–477, 1973.

[8] C. W. Hirt and B.D. Nichols. Volume of Fluid (VOF) Method for the Dynamics of Free Boundaries. *Journal of Computational Physics*, 39:201–225, 1981.

[9] C.W. Hirt, A.A. Amsden, and J.L Cook. An arbitrary Lagrangian-Eulerian computing method for all flow speeds. *Journal of Computational Physics*, 14:227–253, 1974.

[10] L.W. Ho and A.T. Patera. Variational formulation of three-dimensional viscous free-surface flows: Natural imposition of surface tension boundary conditions. *International Journal for Numerical Methods in Fluids*, 13:691–698, 1991.

[11] A. Huerta and A. Rodríguez-Ferran (eds.). The Arbitrary Lagrangian-Eulerian Formulation. *Computer Methods in Applied Mechanics and Engineering*, 193(39-41):4073–4456, 2004.

[12] A.A. Johnson and T.E. Tezduyar. Mesh update strategies in parallel finite element computations of flow problems with moving boundaries and interfaces. *Computer Methods in Applied Mechanics and Engineering*, 119:73–94, 1994.

[13] Y. Maday and A.T. Patera. Spectral element methods for the Navier-Stokes equations. *in: A.K. Noor, J. T. Oden (Eds.), State of the Art Surveys in Computational Mechanics, ASME, New York*, pages 71–143, 1989.

[14] Y. Maday, A.T. Patera, and E.M. Rønquist. An Operator-Integration-Factor Splitting

Method for Time-Dependent Problems: Application to Incompressible Fluid Flow. *Journal of Scientific Computing*, 5(4):263–292, 1990.

[15] S. Osher and R.P. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces.* Springer-Verlag, 2002.

[16] S. Osher and J.A. Sethian. Fronts Propagating with Curvature-Dependent Speed: Algorithms Based on Hamilton-Jacobi Formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[17] B. Ramaswamy and M. Kawahara. Arbitraty Lagrangian-Eulerian finite element method for unsteady convective incompressible viscous free surface flow. *International Journal for Numerical Methods in Fluids*, 7:1053–1074, 1987.

[18] J.A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science.* Cambridge University Press, 1999.

[19] M.A. Walkey, P.H. Gaskell, P.K. Jimack, M.A. Kelmanson, and J.L. Summers. Finite Element Simulation of Three-Dimensional Free-Surface Flow Problems. *Journal of Scientific Computing*, 24(2):147–162, 2005.

[20] H. Zhou and J.J. Derby. An assessment of a parallel, finite element method for three-dimensional, moving-boundary flows driven by capillarity for simulation of viscous sintering. *International Journal for Numerical Methods in Fluids*, 36:841–865, 2001.