ELSEVIER

Comput. Methods Appl. Mech. Engrg. 155 (1998) 235–248

Computer methods in applied mechanics and engineering

Enhanced-Discretization Interface-Capturing Technique (EDICT) for computation of unsteady flows with interfaces

Tayfun Tezduyar*, Shahrouz Aliabadi, Marek Behr

Aerospace Engineering and Mechanics, Army HPC Research Center, University of Minnesota, 1100 Washington Avenue South, Minneapolis, MN 55415, USA

Received 15 May 1997

Abstract

We present the Enhanced-Discretization Interface-Capturing Technique (EDICT) for computation of unsteady flow problems with interfaces, such as two-fluid and free-surface flows. In EDICT, we solve, over a non-moving mesh, the Navier–Stokes equations together with an advection equation governing the evolution of an interface function with two distinct values identifying the two fluids. The starting point for the spatial discretization of these equations are the stabilized finite element formulations which possess good stability and accuracy properties.

To increase the accuracy in modeling the interfaces, we use finite element functions corresponding to enhanced discretization at and near the interface. These functions are designed to have multiple components, with each component coming from a different level of mesh refinement over the same computational domain. The primary component of the functions for velocity and pressure comes from the base mesh called Mesh-1. A subset of the elements in Mesh-1 are identified to be at or near the interface, and depending on where the interface is, this subset could change from one time level to another. A Mesh-2 is constructed by patching together the second-level meshes generated over this subset of elements, and the second component of the functions for velocity and pressure comes from Mesh-2. For the interface function, we have a third component coming from a Mesh-3 which is constructed by patching together the third-level meshes generated over a subset of elements in Mesh-2.

With parallel computation of the test problems presented here, we demonstrate that the EDICT can be used very effectively to increase the accuracy of the base finite element formulations.

1. Introduction

When we need to solve an unsteady flow problem with interfaces, such as fluid–fluid interfaces or free-surface flows, the location of the interface is also an unknown function of time and needs to be determined as part of the overall solution. Depending on the circumstances and the wave lengths at the interface, one can use a fixed mesh or moving mesh method. An interface-tracking method typically requires a moving mesh method, whereas an interface-capturing method can very efficiently be implemented with a fixed mesh approach.

In the category of moving mesh methods, we can mention moving finite element, arbitrary Lagrangian–Eulerian, and the space–time formulations. The stabilized space–time finite element formulation for flows with moving boundaries and interfaces was first introduced, in the context of incompressible flows, in [1,2]. The examples in [2] included 2D simulation of a free-surface flow problem with long-duration time-integration of the free surface.

In the stabilized space–time formulation, the finite element formulation of the problem is written over its space–time domain, and therefore motion of the boundaries and interfaces are taken into account automatically.

* Corresponding author.

PII S0045-7825(97)00194-1

The finite element functions are linear both in space and time, continuous in space, but discontinuous in time. The computations are carried out one space–time 'slab' at a time, where the 'slab' is a slice of the space–time domain between two consecutive time levels. The changes in the shape of the spatial domain is managed by updating the mesh with the combined approaches of moving the mesh (without changing the element connectivities), and remeshing it (i.e. generating a new set of nodes and element) as needed when the mesh distortion becomes too high. Moving the mesh is accomplished by special methods for specific problems and by automatic methods for more general cases. In 3D simulations, especially those based on parallel computing, reducing the frequency of automatic remeshing becomes essential, because the cost of frequent 3D automatic mesh generation could become prohibitive. Therefore, in designing our mesh update methods, we have always taken reducing the frequency of remeshing as a high priority issue. With parallel implementation of the stabilized space–time formulation, we carried out simulations for a number of 3D problems, such as sloshing in a vertically vibrating container [3], a gas stream impinging on a liquid [4] and flow past hydraulic structures [5].

When the interface is complex and very unsteady, especially in 3D simulations, reducing the frequency of remeshing becomes a difficult task, and sometimes not possible. In such cases, interface-capturing methods with fixed meshes remain as practical alternatives. Typically, these interface-capturing methods are more flexible but yield less accurate representation of the interface compared to the interface-tracking methods.

In this paper we present an interface-capturing method with enhanced discretization. We call this the Enhanced-Discretization Interface-Capturing Technique (EDICT). The EDICT was first introduced in [6] and described in more detail in [7]. In EDICT, we start with the basic approach of a volume of fluid (VOF) method [8–10]. The Navier–Stokes equations are solved over a non-moving mesh. An interface function with two distinct values serves as a marker identifying the two fluids. The evolution of the interface function is governed by a time-dependent advection equation. Our objective is to accurately represent and advect the front between the two distinct values of the interface function and to accurately model the interface between the two fluids. We also incorporated into the method a front-sharpening algorithm [11].

Our spatial discretizations are based on the stabilized finite element formulations with the streamline-upwind/Petrov–Galerkin (SUPG) [12], pressure-stabilizing/Petrov–Galerkin (PSPG) [13], and least-squares terms. These terms are added to the standard Galerkin formulation of the momentum equation, incompressibility constraint and the advection equation governing the interface function. They provide stability and accuracy in computation of advection-dominated flows and permit usage of equal-order interpolation functions for velocity and pressure.

To increase the accuracy in representing the interface, we use function spaces corresponding to enhanced discretization at and near the interface. To achieve this, we start with a base mesh that we call Mesh-1. A subset of these elements are identified as those at or near the interface. A more refined Mesh-2 is constructed by patching together second-level meshes generated over each element in this subset. Which elements will be in this subset will change from one time level to another, depending on which elements the interface is passing through. An element which is in this subset now, may be out of it some time later, and come back in again some time after that. For each element in this subset there will be a unique second-level mesh. If an automatic mesh generator is used to generate that, the mesh will be generated only once and stored, to be used later if that element needs a second-level mesh again. The trial and weighting functions for velocity and pressure will all have two components each: one coming from Mesh-1 and the second one coming from Mesh-2.

To further increase the accuracy in representing the interface, we construct a third-level Mesh-3 for the interface function. This is done by identifying a subset of the elements in Mesh-2 as those at or very near the interface. Mesh-3 is constructed by patching together third-level meshes generated over each element in this subset. Which elements in Mesh-2 will be in this subset will change from one time level to other, depending on where the interface is. The construction of Mesh-3 from Mesh-2 will be very similar to the construction of Mesh-2 from Mesh-1. The trial and weighting functions for the interface function will have three components, each coming from one of these three meshes.

Mesh-2 and Mesh-3 will not be re-defined every time step but frequently enough to keep the interface in zones covered with these higher level meshes. In parallel implementation, re-defining these meshes will involve some load balancing cost, but we do not expect this to be a major cost. We will of course have the option of not having a third-level mesh for the interface function. We also have the option of having zones covered by Mesh-2 and Mesh-3 to match. An advantage in keeping them non-matching, however, is that we can keep Mesh-2 wider than Mesh-3, target keeping the interface within Mesh-3, and solve for the interface function only over the part of the computational domain covered by Mesh-2.

At every time step of a simulation, a large, coupled system of nonlinear equations is solved with the Newton–Raphson method. At every step of the Newton–Raphson sequence, a large, coupled system of linear equations is solved iteratively, using diagonal preconditioners with the GMRES [14] update technique. At each of these iterations, the residual of the linear equation system needs to be computed. In performing these residual computations, a vector-based technique is employed to reduce the memory requirements associated with large-scale computations. This matrix-free technique [15,16] totally eliminates the need to compute or store any coefficient matrices, even at the element level. For more information on the iterative computation methods used here, see [5]. The EDICT has already been implemented on the shared-memory parallel computing platform of the SGI multi-processor systems, and this was reported in [6,7].

In Section 2 we review the governing equations. The finite element formulation for the EDICT is presented in Section 3. We provide more details on the concept of enhanced discretization in Section 4. Numerical examples are presented in Section 5, and we end with concluding remarks in Section 6.

2. Governing equations

The flow computations are based on the solution of time-dependent Navier–Stokes equations of incompressible flows. In our notation, Ω and $(0, T)$ will denote the space and time domains, and Γ the boundary of Ω. The symbols $\rho(\boldsymbol{x}, t)$, $\boldsymbol{u}(\boldsymbol{x}, t)$ and $p(\boldsymbol{x}, t)$ represent the density, velocity and pressure, respectively. The external forces (e.g. gravity) are represented by $\boldsymbol{f}(\boldsymbol{x}, t)$. The momentum conservation equations and the incompressibility constraint can be written in the following vector form:

$$\rho\left(\frac{\partial \boldsymbol{u}}{\partial t} + \boldsymbol{u}\cdot\nabla\boldsymbol{u} - \boldsymbol{f}\right) - \nabla\cdot\boldsymbol{\sigma} = \boldsymbol{0} \quad \text{on } \Omega \quad \forall t \in (0, T), \tag{1}$$

$$\nabla\cdot\boldsymbol{u} = 0 \quad \text{on } \Omega \quad \forall t \in (0, T), \tag{2}$$

where

$$\boldsymbol{\sigma}(\boldsymbol{u}, p) = -p\boldsymbol{I} + \boldsymbol{T}, \tag{3}$$

$$\boldsymbol{T} = 2\mu\boldsymbol{\varepsilon}(\boldsymbol{u}). \tag{4}$$

Here, $\boldsymbol{I}$ is the identity tensor, μ is the dynamic viscosity, and $\boldsymbol{\varepsilon}(\boldsymbol{u})$ is the strain rate tensor defined as

$$\boldsymbol{\varepsilon}(\boldsymbol{u}) = \frac{1}{2}(\nabla\boldsymbol{u} + (\nabla\boldsymbol{u})^{\mathrm{T}}). \tag{5}$$

To model fluid–fluid interfaces, we consider two immiscible fluids, A and B, with densities ρ_A and ρ_B and viscosities μ_A and μ_B.

REMARK 1. If we want to model a liquid–gas interaction, we can let Fluid A be the liquid and Fluid B the gas. If we want to model a free-surface problem where Fluid B is irrelevant, we can do that by assigning a sufficiently low density to Fluid B.

An interface function ϕ serves as a marker identifying Fluid A and B with the definition $\phi = \{1$ for Fluid A and 0 for Fluid $B\}$. The interface between the two fluids is approximated to be at $\phi = 0.5$. In this context, ρ and μ are defined as

$$\rho = \phi\rho_A + (1 - \phi)\rho_B, \tag{6}$$

$$\mu = \phi\mu_A + (1 - \phi)\mu_B. \tag{7}$$

The evolution of the interface function ϕ is governed by a time-dependent advection equation:

$$\frac{\partial \phi}{\partial t} + \boldsymbol{u}\cdot\nabla\phi = 0 \quad \text{on } \Omega \quad \forall t \in (0, T). \tag{8}$$

REMARK 2. One can also see Eqs. (6)–(8) as those representing the constitutive law of the fluid system. How accurately this law will be modeled will depend on how accurately the front between $\phi = 1$ and $\phi = 0$ will be represented and advected.

REMARK 3. We will not address here the surface tension effects at the interfaces.

The set of equations given in this section needs to be completed with an appropriate set of boundary conditions, a divergence-free initial condition on the velocity field, and the initial profile of the interface function.

3. Finite element formulation

In writing the stabilized finite element formulation, we first assume that we have some appropriately defined finite-dimensional function spaces for trial and weighting functions corresponding to velocity, pressure and interface function. The trial function spaces will be denoted by $(\mathcal{S}_{\boldsymbol{u}}^h)_n$, $(\mathcal{S}_p^h)_n$ and $(\mathcal{S}_\phi^h)_n$, and the weighting function spaces by $(\mathcal{V}_{\boldsymbol{u}}^h)_n$, $(\mathcal{V}_p^h)_n$ and $(\mathcal{V}_\phi^h)_n$. While the superscript h implies that these are finite-dimensional function spaces, the subscript n implies that corresponding to different time levels we can have different spatial discretizations. We will give a more precise definition of these function spaces later.

The stabilized formulations of Eqs. (1), (2) and (8) can be written as follows: given $\boldsymbol{u}_n^h$ and ϕ_n^h, find $\boldsymbol{u}_{n+1}^h \in (\mathcal{S}_{\boldsymbol{u}}^h)_{n+1}$, $p_{n+1}^h \in (\mathcal{S}_p^h)_{n+1}$ and $\phi_{n+1}^h \in (\mathcal{S}_\phi^h)_{n+1}$, such that $\forall\, \boldsymbol{w}_{n+1}^h \in (\mathcal{V}_{\boldsymbol{u}}^h)_{n+1}$, $\forall\, q_{n+1}^h \in (\mathcal{V}_p^h)_{n+1}$ and $\forall\, \psi_{n+1}^h \in (\mathcal{V}_\phi^h)_{n+1}$:

$$\begin{aligned}
&\int_\Omega \boldsymbol{w}_{n+1}^h \cdot \rho\left(\frac{\partial \boldsymbol{u}^h}{\partial t} + \boldsymbol{u}^h \cdot \nabla \boldsymbol{u}^h - \boldsymbol{f}\right) \mathrm{d}\Omega + \int_\Omega \boldsymbol{\varepsilon}(\boldsymbol{w}_{n+1}^h) : \boldsymbol{\sigma}(p^h, \boldsymbol{u}^h)\, \mathrm{d}\Omega + \int_\Omega q_{n+1}^h \nabla \cdot \boldsymbol{u}_{n+1}^h \, \mathrm{d}\Omega \\
&\quad + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \left(\tau_{\mathrm{SUPG}} \boldsymbol{u}^h \cdot \nabla \boldsymbol{w}_{n+1}^h + \frac{\tau_{\mathrm{PSPG}}}{\rho} \nabla q_{n+1}^h\right) \cdot \left[\rho\left(\frac{\partial \boldsymbol{u}^h}{\partial t} + \boldsymbol{u}^h \cdot \nabla \boldsymbol{u}^h - \boldsymbol{f}\right) - \nabla \cdot \boldsymbol{\sigma}(p^h, \boldsymbol{u}^h)\right] \mathrm{d}\Omega \\
&\quad + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \tau_{\mathrm{CONT}} \nabla \cdot \boldsymbol{w}_{n+1}^h \rho \nabla \cdot \boldsymbol{u}_{n+1}^h \, \mathrm{d}\Omega = \int_{\Gamma_h} \boldsymbol{w}_{n+1}^h \cdot \boldsymbol{h}^h \, \mathrm{d}\Gamma\,,
\end{aligned} \tag{9}$$

$$\int_\Omega \psi_{n+1}^h \left(\frac{\partial \phi^h}{\partial t} + \boldsymbol{u}^h \cdot \nabla \phi^h\right) \mathrm{d}\Omega + \sum_{e=1}^{n_{\mathrm{el}}} \int_{\Omega^e} \tau_\phi \boldsymbol{u}^h \cdot \nabla \psi_{n+1}^h \left(\frac{\partial \phi^h}{\partial t} + \boldsymbol{u}^h \cdot \nabla \phi^h\right) \mathrm{d}\Omega = 0\,, \tag{10}$$

where $\boldsymbol{h}^h$ represents the Neumann-type boundary condition associated with the momentum equation. Here, τ_{SUPG}, τ_{PSPG}, τ_{CONT} and τ_ϕ are the stabilization parameters:

$$\tau_{\mathrm{SUPG}} = \left(\left(\frac{2\|\boldsymbol{u}^h\|}{h}\right)^2 + \left(\frac{4\nu}{h^2}\right)^2\right)^{-1/2}, \tag{11}$$

$$\tau_{\mathrm{PSPG}} = \tau_{\mathrm{SUPG}}\,, \tag{12}$$

$$\tau_{\mathrm{CONT}} = \frac{h}{2}\|\boldsymbol{u}^h\| z\,, \tag{13}$$

where

$$\begin{aligned}
z &= \begin{cases} \left(\dfrac{\mathrm{Re}_u}{3}\right) & \mathrm{Re}_u \leqslant 3 \\ 1 & \mathrm{Re}_u > 3 \end{cases}, \\
\tau_\phi &= \frac{h}{2\|\boldsymbol{u}^h\|}\,,
\end{aligned} \tag{14}$$

where Re_u is the cell Reynolds number.

REMARK 4. In Eq. (9), the first three integrals, together with the right-hand side, represent the Galerkin formulation of (1)–(2). The first series of element-level integrals in the formulation are the SUPG and PSPG stabilization terms. The second series of element-level integrals are the least-squares stabilization terms based on

the incompressibility constraint. In Eq. (10), the first integral represents the Galerkin formulation of (8), while the series of element-level integrals are the SUPG stabilization terms.

REMARK 5. In time discretization, the time derivatives, $\partial \boldsymbol{u}/\partial t$ and $\partial \phi/\partial t$ are represented as follows:

$$\frac{\partial \boldsymbol{u}}{\partial t} = \frac{\boldsymbol{u}^h_{n+1} - \boldsymbol{u}^h_n}{\Delta t}, \tag{15}$$

$$\frac{\partial \phi}{\partial t} = \frac{\phi^h_{n+1} - \phi^h_n}{\Delta t}, \tag{16}$$

where Δt is the time step size between time levels n and $n+1$. In this time discretization, the functions $\boldsymbol{u}^h$, p^h and ϕ^h are represented as follows:

$$\boldsymbol{u}^h \leftarrow (1-\alpha)\boldsymbol{u}^h_n + \alpha \boldsymbol{u}^h_{n+1}, \tag{17}$$

$$p^h \leftarrow p^h_{n+1}, \tag{18}$$

$$\phi^h \leftarrow (1-\alpha)\phi^h_n + \alpha \phi^h_{n+1}, \tag{19}$$

where α is a time-integration parameter controlling the stability and accuracy of the integration. Normally, we set $\alpha = 0.5$. However, $\alpha = 1$ in Eq. (19) as well as for the equation governing the surface motion.

4. Construction of function spaces—enhanced discretization

To construct the function spaces corresponding to time level n, we start with a base mesh which we will call Mesh-1. The set of elements and nodal points will be denoted by ϵ^1_n and η^1_n. The subscript n implies that Mesh-1 might change from one time level to other, reflecting the possibility that occasionally we might replace the base mesh with a different one.

A second-level and more refined mesh will be constructed over a subset $(\epsilon^1_n)^2_n$ of these elements. This Mesh-2 will be generated by patching together the second-level meshes generated over each of the elements in $(\epsilon^1_n)^2_n$ (see Figs. 1 and 2).

REMARK 6. The second occurrence of the subscript n implies that this subset might change from one time level to another, even though ϵ^1_n might be the same. In other words, for a given Mesh-1, which elements of this mesh will be declared to be in $(\epsilon^1_n)^2_n$ might change from one time level to another. An element which might be declared to be in $(\epsilon^1_n)^2_n$ at some time level, might fall out of it at some other time, and yet come back in again at some time later.

REMARK 7. For each element in ϵ^1_n, there will be a unique second-level mesh. Therefore, if an element is declared to be in $(\epsilon^1_n)^2_n$ for a second time, the refined mesh generated over that element will be the same as the one generated at the earlier declaration. This way, if an automatic mesh generator is being used to generate these meshes, the cost for that mesh generation will be a one-time cost.

The set of elements and nodal points for Mesh-2 will be denoted by ϵ^2_n and η^2_n.

A third-level and even more refined mesh will be constructed over a subset $(\epsilon^2_n)^3_n$ of the elements in Mesh-2. This Mesh-3 will be generated by patching together the third-level meshes generated over each of the elements in $(\epsilon^2_n)^3_n$ (see Figs. 1–3).

REMARK 8. The statements in Remarks 6 and 7 apply in this case too, with the mesh level numbers referred to in Remarks 6 and 7 shifted up by one.

The set of elements and nodal points for Mesh-3 will be denoted by ϵ^3_n and η^3_n.

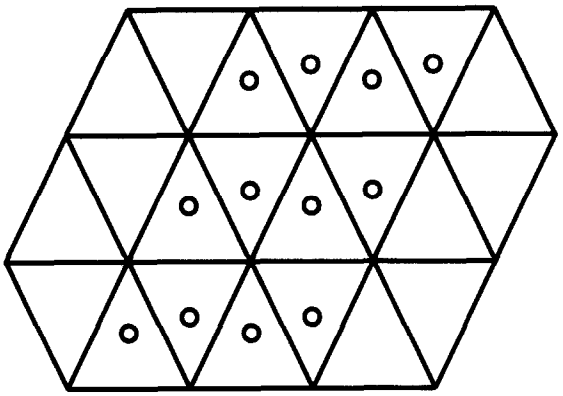

Fig. 1. Mesh-1. The set of elements are denoted by ϵ_n^1. The elements marked by ○ are those declared to be in the subset $(\epsilon_n^1)_n^2$.

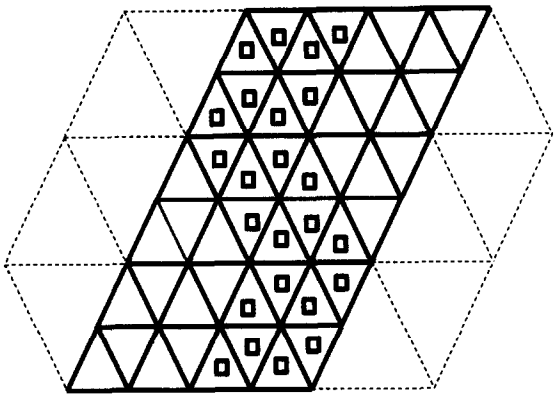

Fig. 2. Mesh-2. The set of elements are denoted by ϵ_n^2. The elements marked by □ are those declared to be in the subset $(\epsilon_n^2)_n^3$.

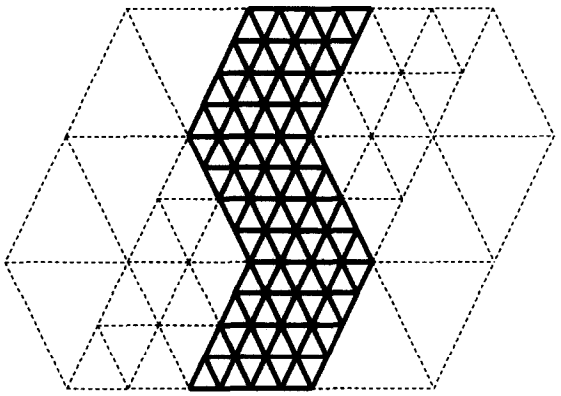

Fig. 3. Mesh-3.

REMARK 9. We will limit ourselves to linear elements in 2D and 3D, as well as bilinear and trilinear elements, respectively, in 2D and 3D.

We construct $\boldsymbol{u}_n^h$ as follows:

$$\boldsymbol{u}_n^h = \boldsymbol{u}_n^1 + \boldsymbol{u}_n^2 . \tag{20}$$

The function $\boldsymbol{u}_n^1$ comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^1, excluding those 'surrounded' by the elements in $(\epsilon_n^1)_n^2$. The function $\boldsymbol{u}_n^1$ also needs to satisfy the Dirichlet-type boundary conditions, except at those nodes that have been surrounded at the boundary of Ω.

The function $\boldsymbol{u}_n^2$ comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^2, excluding those at the boundaries of the zones covered by the elements in ϵ_n^2. However, we do include the nodes at the boundary of Ω unless they coincide with the nodes in η_n^1 that have not been surrounded.

We construct p_n^h in exactly the same way, except for recognizing the fact that the references to Dirichlet-type boundary conditions do not apply:

$$p_n^h = p_n^1 + p_n^2 . \tag{21}$$

We construct ϕ_n^h with more enhancement:

$$\phi_n^h = \phi_n^1 + \phi_n^2 + \phi_n^3 . \tag{22}$$

The function ϕ_n^1 comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^1, excluding those 'surrounded' by the elements in $(\epsilon_n^1)_n^2$. The function ϕ_n^1 also needs to satisfy the Dirichlet type boundary conditions, except at those nodes that have been surrounded at the boundary of Ω.

The function ϕ_n^2 comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^2, excluding those at the boundaries of the zones covered by the elements in ϵ_n^2. We also exclude the nodes surrounded by the elements in $(\epsilon_n^2)_n^3$. However, we do include the nodes at the boundary of Ω unless they coincide with the nodes in η_n^1 that have not been surrounded.

The function ϕ_n^3 comes from a space of functions with the basis set consisting of the shape functions associated with all the nodes in η_n^3, excluding those at the boundaries of the zones covered by the elements in ϵ_n^3. However, we do include the nodes at the boundary of Ω unless they coincide with the nodes in η_n^2 that have not been surrounded.

The weighting function are constructed in a similar fashion:

$$\boldsymbol{w}_n^h = \boldsymbol{w}_n^1 = \boldsymbol{w}_n^2 , \tag{23}$$

$$q_n^h = q_n^1 + q_n^2 , \tag{24}$$

$$\psi_n^h = \psi_n^1 + \psi_n^2 + \psi_n^3 . \tag{25}$$

The components of each weighting function are defined in the same way as we did for the trial functions, except that the weighting functions need to satisfy the homogeneous form of the Dirichlet-type boundary conditions.

Our objective with this enhanced discretization is to capture the interface as accurately as possible by using more refined meshes for the velocity, pressure and interface function, and possibly even more refined meshes for the last. However, we do this in a dynamic fashion by defining $(\epsilon_n^1)_n^2$ and $(\epsilon_n^2)_n^3$ depending on which elements in ϵ_n^1 and ϵ_n^2 the interface is passing through, and re-define these subsets occasionally to track the interface.

REMARK 10. We update $(\epsilon_n^1)_n^2$ and $(\epsilon_n^2)_n^3$ not every time step but with sufficient frequency to keep the interface within the zones covered by these subsets of elements. Although it would not be 'illegal' for the interface to fall out of these zones, we attempt to estimate or keep track of, how many time steps of the simulation the interface will remain inside these zones. How many time steps one can carry out the simulations without re-defining these subsets of elements will depend, among other things, on how 'wide' we decide to keep these zones around the interface.

REMARK 11. Whenever we re-define these subsets, the mesh generation cost will not be a significant one. If we have to use an automatic mesh generator to generate the second- and third-level meshes, we will be able to use and re-use the meshes which have been generated at the very beginning of the simulations and stored.

REMARK 12. In parallel implementation, re-defining the subsets will require that load balancing is re-done. We do not anticipate this to be a major cost, since the METIS mesh partitioning package [17] performs very well, and its parallel version is now functional.

REMARK 13. It is possible to eliminate ϕ_n^3 by not choosing to go to a third-level of refinement. It is also possible to design the second- and third-level meshes in such a way that their zones match. One of the advantages in keeping these zones non-matching is that, by keeping Mesh-2 'wider' than Mesh-3, one can choose to limit the existence (as an unknown) of ϕ to Mesh-2, and therefore solving for it only over the part of the computational domain covered by Mesh-2. With this, we have to make sure that the interface remains in Mesh-2 zone. Since our objective will be to keep the interface in Mesh-3 zone, this would also keep it in Mesh-2 zone, even if the interface occasionally falls slightly out of the Mesh-3 zone.

5. Examples

5.1. 2D sloshing in a container

A container with dimensions 10 cm × 7.5 cm (horizontal × vertical) is filled 2/3 with water and 1/3 with air. It is suddenly subjected to the gravitational acceleration ($g = 9.8\ \mathrm{m/s^2}$) and a horizontal acceleration of magnitude 0.2 g.

We first compute this problem with the stabilized space–time formulation using a highly-refined moving mesh. The computational domain is discretized using 200 elements in the horizontal direction and 150 elements in the vertical direction, and this results in 30 000 quadrilateral elements and 30 351 nodes. The two images in Fig. 4 show, at $t = 0.57$ s, the finite element mesh and the air–water interface. We will refer to this solution, obtained with the Interface-Tracking Technique, as Solution-IT. We will use it to evaluate the solutions obtained with the Interface-Capturing Technique with no enhanced discretization and the EDICT.

We use this structured mesh at its initial non-deformed configuration to compute the same problem with the Interface-Capturing Technique with no enhanced discretization. The two images in Fig. 5 show, at $t = 0.57$ s, the finite element mesh and the air–water interface. We will refer to this solution, obtained with the Interface-Capturing Technique, as Solution-IC.

Time histories of the horizontal forces exerted on the container derived from Solution-IT and Solution-IC are compared in Fig. 6. The agreement between the solutions is satisfactory.

Next, we compute the problem with the Interface-Capturing Technique using unstructured, triangular meshes. The base mesh, Mesh-1, consists of 13 739 elements and 7021 nodes. Mesh-2 is obtained by subdividing each element in $(\epsilon_n^1)_n^2$ into four elements, in a style as shown in Fig. 7. Mesh-3 is obtained by subdividing each element in $(\epsilon_n^2)_n^3$ into four elements, in a style as shown, again, in Fig. 7. We set $(\epsilon_n^2)_n^3 = \epsilon_n^2$; this means that we have the Mesh-2 and Mesh-3 zones matching. We carry out the computations with a time step size of 0.005 s. We redefine $(\epsilon_n^1)_n^2$ at every five time steps, with all elements in ϵ_n^1 within a distance of 0.5 cm from the interface declared to be in $(\epsilon_n^1)_n^2$.

Solution-1 is obtained by using the base discretization, where all trial and weighting functions come only from Mesh-1. Solution-2 is obtained by using the EDICT, where all trial and weighting functions come from Mesh-1 ⊕ Mesh-2 (with no third-level discretization for the interface function). Solution-3 is obtained by using a

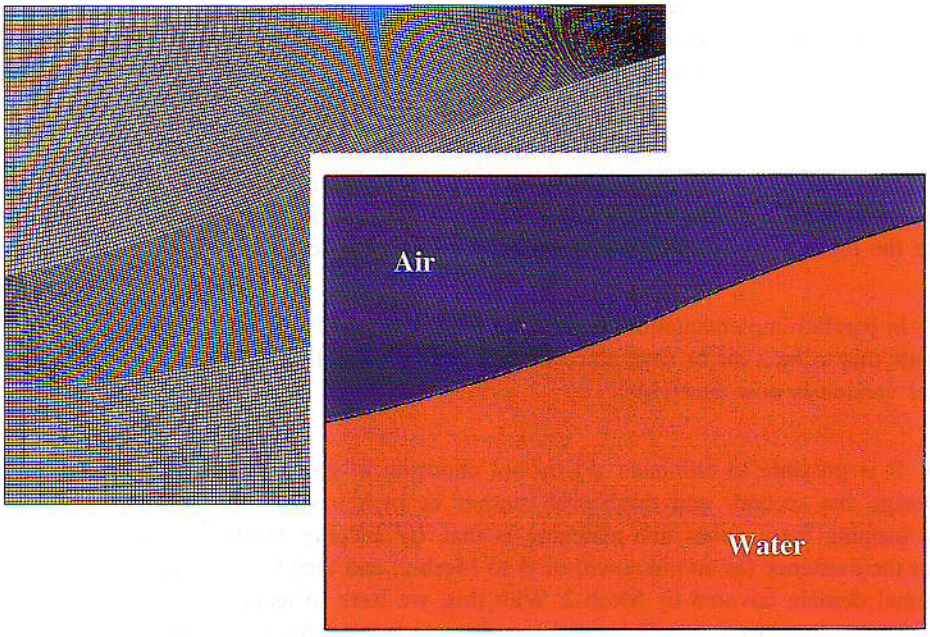

Fig. 4. 2D sloshing in a container. The two images show, at $t = 0.57$ s, the mesh and the air–water interface for Solution-IT.

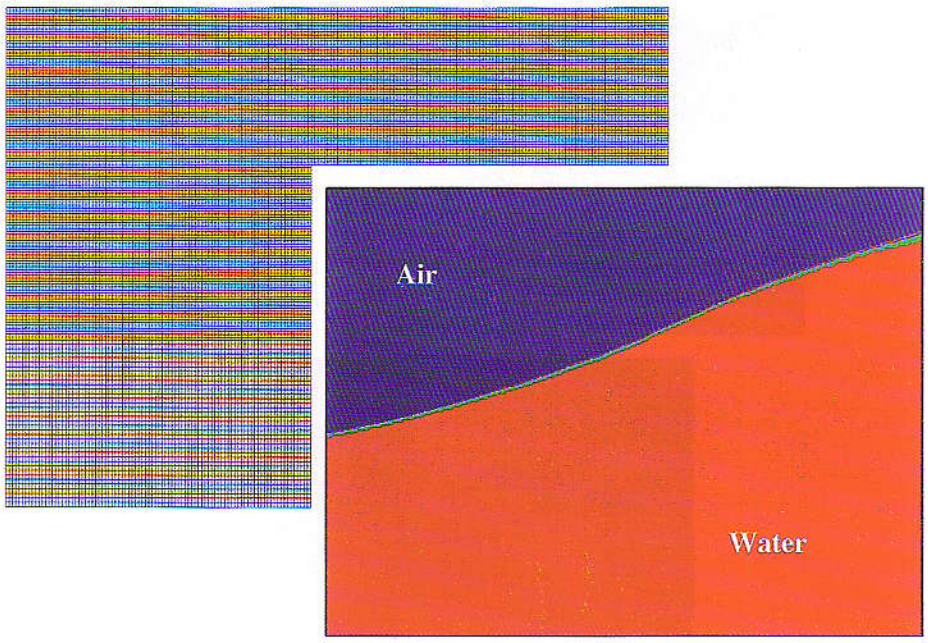

Fig. 5. 2D sloshing in a container. The two images show, at $t = 0.57$ s, the mesh and the air–water interface for Solution-IC.

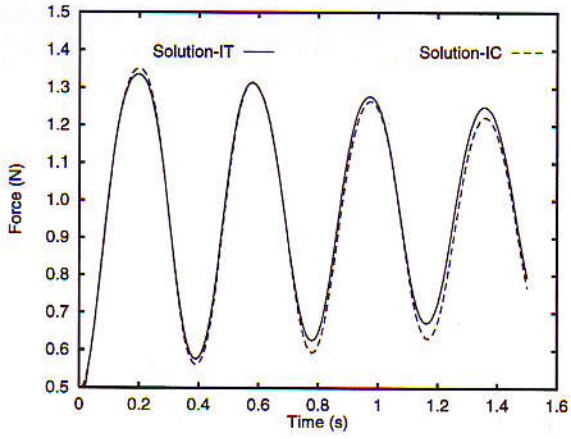

Fig. 6. 2D sloshing in container. Time histories of the horizontal forces exerted on the container derived from Solution-IT and Solution-IC.

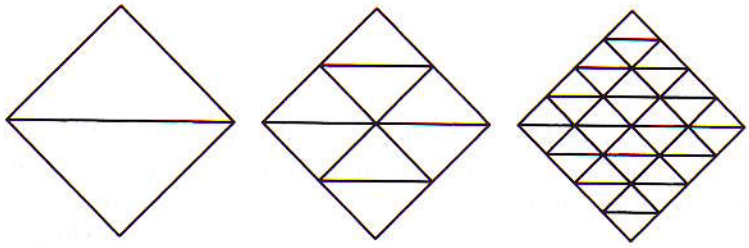

Fig. 7. 2D sloshing in a container. Representive elements from Mesh-1, Mesh-2 and Mesh-3.

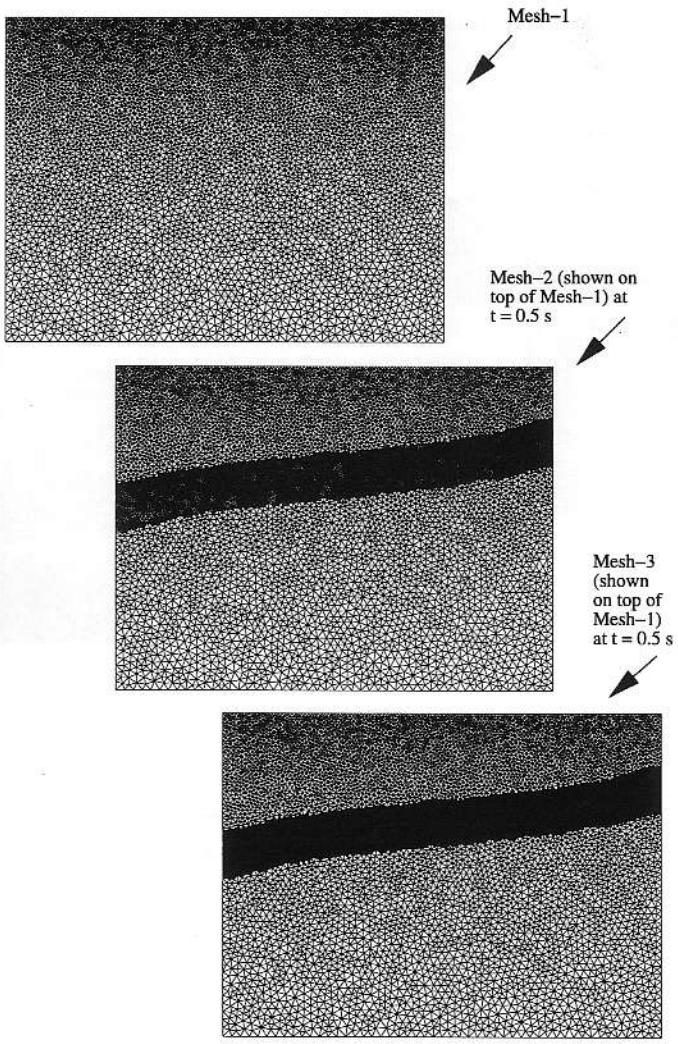

Fig. 8. 2D sloshing in a container.

more enhanced discretization, where the trial and weighting functions for velocity and pressure come from Mesh-1 $\oplus$ Mesh-2, and for the interface function from Mesh-1 $\oplus$ Mesh-2 $\oplus$ Mesh-3.

Fig. 8 shows Mesh-1 together with Mesh-2 and Mesh-3 (both shown on top of Mesh-1) at $t = 0.5$ s. Refinement of 2221 elements of Mesh-1 results in Mesh-2 having 8884 elements and Mesh-3 having 35 536 elements. The graphs in Fig. 9 show the time histories of the horizontal forces exerted on the container for all four solutions. We assume that the target solution is Solution-IT, obtained with the Interface-Tracking Technique, using a highly-refined mesh. It is clear from these graphs that Solution-1 has significant frequency and amplitude errors. Solution-2 is superior to Solution-1, but it is Solution-3 that is in best agreement with Solution-IT.

5.2. Axisymmetric filling / impact

This problem was simulated with the EDICT. We have a circular cylinder with both the radius and height of 10 cm. The lower half of the cylinder is filled with a liquid with density equal to that of water but viscosity 100 times that of water. The upper half is filled with air. At $t = 0.0$ s we start injecting the same liquid through a circular section with radius 2.15 cm and positioned concentrically at the top of the cylinder. The injection stream has a uniform flow speed of 1.0 m/s. Fig. 10 shows the problem geometry.

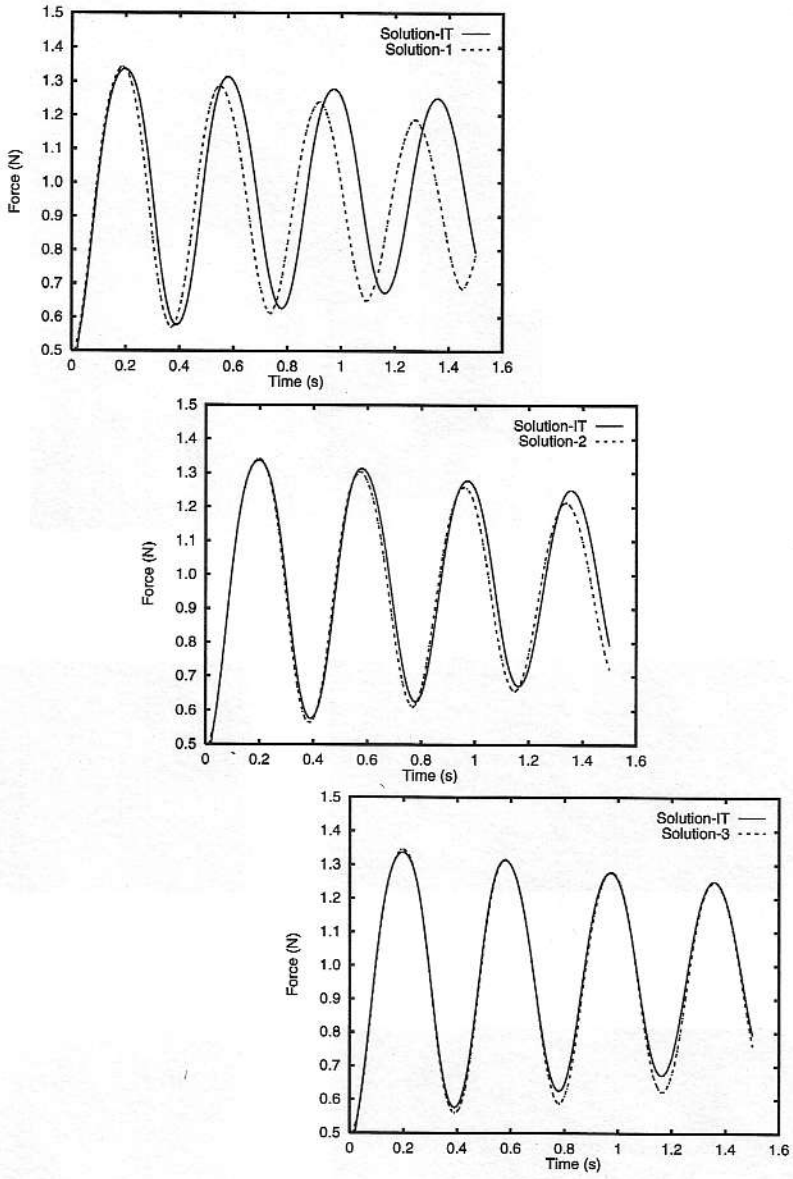

Fig. 9. 2D sloshing in a container. Time histories of the horizontal forces exerted on the container for all four solutions.

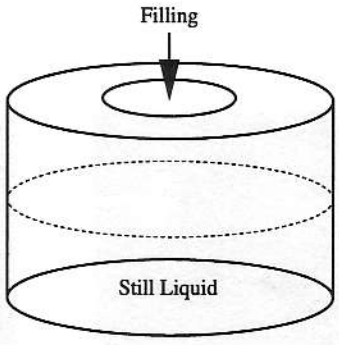

Fig. 10. Axisymmetric filling/impact. Problem geometry.

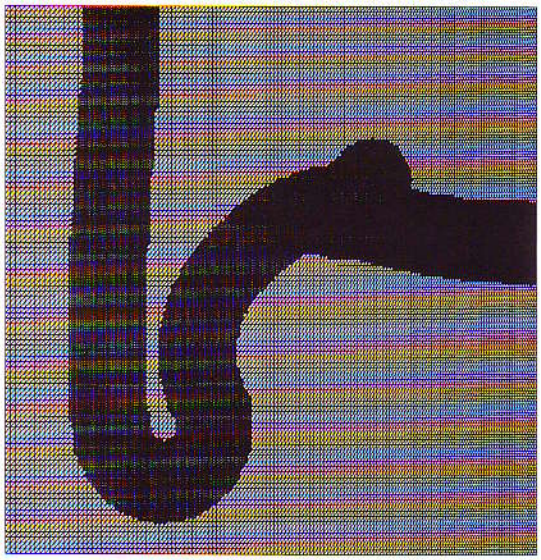

Fig. 11. Axisymmetric filling/impact. Mesh-2 (shown on top of Mesh-1) at $t = 0.15$ s.

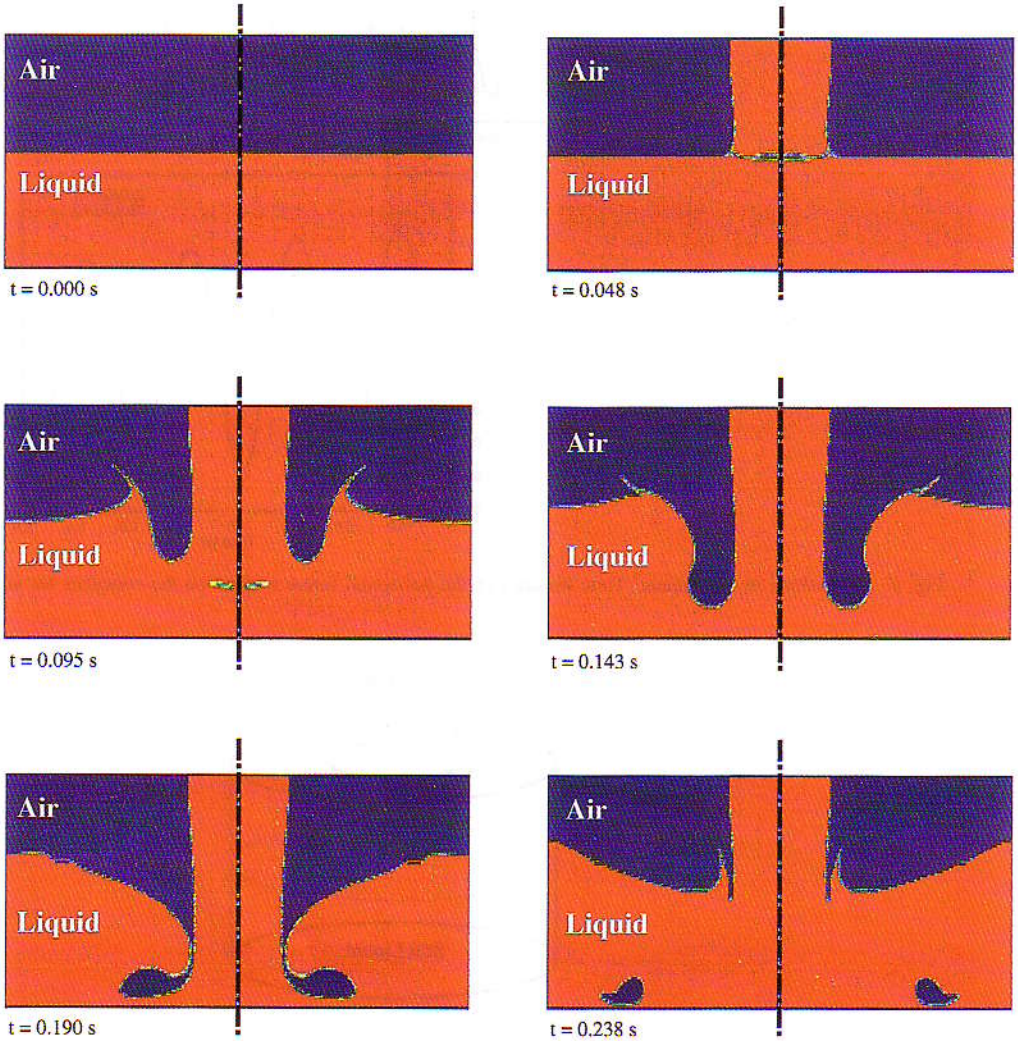

Fig. 12. Axisymmetric filling/impact. The pictures show a sequence of air–liquid interactions.

A finite element mesh with triangular elements is used in the computation. The base mesh, Mesh-1, consists of 30 000 elements and 15 251 nodes. Mesh-2 and Mesh-3 are generated in the same way as they were generated in the previous test problem. However, this time, at every redefinition of $(\epsilon_n^1)_n^2$, all elements in ϵ_n^1 within a distance of 0.7 cm from the interface are declared to be in $(\epsilon_n^1)_n^2$.

The solution is obtained by using an enhanced discretization, where the trial and weighting functions for velocity and pressure come from Mesh-1 $\oplus$ Mesh-2, and for the interface function from Mesh-1 $\oplus$ Mesh-2 $\oplus$ Mesh-3.

Fig. 11 shows Mesh-2 (on top of Mesh-1) at $t = 0.15$ s. Refinement of 8473 elements of Mesh-1 results in Mesh-2 having 33 892 elements and Mesh-3 having 135 568 elements. Fig. 12 shows a sequence of air–liquid interactions seen at different instants during the simulation of this problem. The pictures show the injection stream impacting the still liquid, formation of surface waves, and entrapment of air in the liquid.

6. Concluding remarks

In this paper we have presented the Enhanced-Discretization Interface-Capturing Technique (EDICT) for simulation of unsteady flow problems involving fluid–fluid interfaces and free surfaces. An advection equation governing the evolution of an interface function with two distinct values identifying the two fluids is solved together with the Navier–Stokes equations. To solve these equations, we use the streamline-upwind/Petrov–Galerkin, pressure-stabilizing/Petrov–Galerkin, and least-squares stabilization methods as the base finite element formulations.

Although these formulations already possess good stability and accuracy properties, our objective here was to further increase the accuracy in modeling the interfaces. For this purpose, we use finite element functions with multiple components. Each of these components corresponds to one of the multiple levels of mesh, with the higher level meshes placed at and near the interface. The first component of the functions for velocity and pressure comes from the base mesh. A subset of the elements in this mesh are declared to be at or near the interface. This subset changes in time to track the interface, not at every time level, but frequently enough to keep the interface within the zone covered by this subset. A more refined mesh is constructed by patching together the second-level meshes generated over this subset of elements. The second component of the functions for velocity and pressure come from this more refined mesh. The interface function could have a third component coming from even a more refined mesh which is constructed by patching together the third-level meshes generated over a subset of elements in the second-level mesh.

Here, we have used the EDICT to increase, in an efficient way, the accuracy of our stabilized formulations which function as the base finite element formulations for solution of the Navier–Stokes equations and the advection equation governing the interface function. Other finite element formulations can also be used as base formulations, including those well-tuned to accurately solve the scalar advection equation. The EDICT can be implemented efficiently on parallel computing platforms, and the test problems reported here were computed in a shared-memory parallel computing paradigm. The results from these test computations show that the EDICT yields substantial increases in the accuracy of the interface computations.

Acknowledgments

This work is sponsored by ARPA and by the Army High Performance Computing Research Center under the auspices of the Department of the Army, Army Research Laboratory cooperative agreement number DAAH04-95-2-0003/contract number DAHH04-95-0008. The content does not necessarily reflect the position or the policy of the government, and no official endorsement should be inferred. The CRAY time was provided, in part, by the University of Minnesota Supercomputer Institute.

References

[1] T.E. Tezduyar, M. Behr and J. Liou, A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space–time procedure: I. The concept and the preliminary tests, Comput. Methods Appl. Mech. Engrg. 94 (1992) 339–351.

[2] T.E. Tezduyar, M. Behr, S. Mittal and J. Liou, A new strategy for finite element computations involving moving boundaries and interfaces—the deforming-spatial-domain/space–time procedure: II. Computation of free-surface flows, two-liquid flows, and flows with drifting cylinders, Comput. Methods Appl. Mech. Engrg. 94 (1992) 353–371.
[3] M. Behr and T.E. Tezduyar, Finite element solution strategies for large-scale flow simulations, Comput. Methods Appl. Mech. Engrg. 112 (1994) 3–24.
[4] G. Wren, S. Ray, S. Aliabadi and T. Tezduyar, Simulation of flow problems with moving mechanical components, fluid–structure interactions and two-fluid interfaces, Int. J. Numer. Methods Fluids, 24 (1997) 1433–1448.
[5] T. Tezduyar, S. Aliabadi, M. Behr, A. Behr, A. Johnson, V. Kalro and M. Litke, Flow simulation and high performance computing, Comput. Mech. 18 (1996) 397–412.
[6] T.E. Tezduyar, S. Aliabasi and M. Behr, Enhanced-Discretization Interface-Capturing Technique, in: Y. Matsumoto and Prasperetti, eds., ISAC '97 High Performance Computing on Multiphase Flows (Japan Society of Mechanical Engineers, Tokyo, Japan, 1997).
[7] T.E. Tezduyar, S. Aliabadi and M. Behr, Parallel finite element computing methods for unsteady flows with interfaces, Comput. Fluid Dyn. Rev. ~~to appear.~~ 1998 (eds. M. Hafez and K. Oshima), World Scientific (1998) 643-667.
[8] C.W. Hirt and B.D. Nichols, Volume of fluid (VOF) method for the dynamics of free boundaries, J. Comput. Phys. 39 (1981) 201–225.
[9] D.L. Youngs, Time-dependent multimaterial flow with large fluid distortion, in: K.W. Morton and M.J. Baines, eds., Numerical Methods in Fluid Dynamics, Notes on Numerical Fluid Mechanics (Academic Press, New York, 1984) 273–285.
[10] C.M. Lemos, Higher-order schemes for free-surface flows with arbitrary configurations, Int. J. Numer. Methods Fluids 23 (1996) 545–566.
[11] S. Aliabadi and T. Tezduyar, 3D simulation of free-surface flows with parallel finite element method, in: S. Atluri and G. Yagawa, eds., Proc. Int. Conf. on Computational Engineering Science, San Jose, Costa Rica, 1997.
[12] T.J.R. Hughes and A.N. Brooks, A multi-dimensional upwind scheme with no cross-wind diffusion, in: T.J.R. Hughes, ed., Finite Element Methods for Convection Dominated Flows, AMD-Vol. 34 (ASME, New York, 1979) 19–35.
[13] T.E. Tezduyar, Stabilized finite element formulations for incompressible flow computations, Adv. Appl. Mech. 28 (1991) 1–44. ↙ 1992
[14] Y. Saad and M. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, SIAM J. Sci. Statis. Comput. 7 (1986) 856–869.
[15] Z. Johan, T.J.R. Hughes and F. Shakib, A globally convergent matrix-free algorithm for implicit time-marching schemes arising in finite element analysis in fluids, Comput. Methods Appl. Mech. Engrg. 87 (1991) 281–304.
[16] S.K. Aliabadi and T.E. Tezduyar, Parallel fluid dynamics computations in aerospace applications, Int. J. Numer. Methods Fluids 21 (1995) 783–805.
[17] G. Karypis and V. Kumar, Multilevel k-ways partitioning scheme for irregular graphs, Technical Report 95-064, University of Minnesota, Department of Computer Science, 1995.