HERMIS: The International Journal of Computer Mathematics and its Applications, 4 (2003) 63-88.

STABILIZED FINITE ELEMENT METHODS FOR FLOWS WITH MOVING BOUNDARIES AND INTERFACES

Tayfun E. Tezduyar
Mechanical Engineering, Rice University – MS 321
6100 Main Street, Houston, TX 77005, USA

Abstract

We highlight some of the finite element methods we developed for computation of flows with moving boundaries and interfaces. The methods developed can be classified into two main categories: interface-tracking and interface-capturing techniques. Both classes of techniques are based on stabilized formulations, and determination of the stabilization parameters used in these formulations is also highlighted here. The interface-tracking techniques are based on the Deforming-Spatial-Domain/Stabilized Space–Time (DSD/SST) formulation, where the mesh moves to track the interface. The interface-capturing techniques, developed for two-fluid flows, are formulated typically over non-moving meshes, using an advection equation in addition to the flow equations. The advection equation governs the time-evolution of an interface function marking the interface location. We also highlight some of the additional methods we developed to increase the scope and accuracy of these two classes of techniques.

I. Introduction

In computation of flow problems with moving boundaries and interfaces, depending on the problem, we can use an interface-tracking or interface-capturing technique. An interface-tracking technique requires meshes that "track" the interfaces. The mesh needs to be updated as the flow evolves. In an interface-capturing technique for two-fluid flows, the computations are based on fixed spatial domains, where an interface function, marking the location of the interface, needs to be computed to "capture" the interface. The interface is captured within the resolution of the finite element mesh covering the area where the interface is.

The interface-tracking and interface-capturing techniques we have developed (see [1], [2], [3], [4], [5], [6]) are based on stabilized formulations. The stabilized methods are the streamline-upwind/Petrov-Galerkin (SUPG) [7], [8], [9] and pressure-stabilizing/Petrov-Galerkin (PSPG) [1] formulations. An earlier version of the pressure-stabilizing formulation for Stokes flows was reported in [10]. These stabilized formulations prevent numerical oscillations and other instabilities in solving problems with high Reynolds and/or Mach numbers and shocks and strong boundary layers, as well as when using equal-order interpolation functions for velocity and pressure and other unknowns. Furthermore, this class of stabilized formulations substantially improve the convergence rate in iterative solution of the large, coupled nonlinear equation system that needs to be solved at every time step of a flow computation. Such nonlinear systems are typically solved with the Newton–Raphson method, which involves, at its every iteration step, solution of a large, coupled linear equation system. It is in iterative solution of such linear equation systems that using a good stabilized method makes substantial difference in convergence, and this was pointed out in [11].

In these stabilized formulations, judicious selection of the stabilization parameter, which is almost always known as "τ", plays an important role in determining the accuracy of the formulation. This stabilization parameter involves a measure of the local length scale (also known as "element length") and other parameters such as the local Reynolds and Courant numbers. Various element lengths and τs were proposed starting with those in [7], [8], [9], followed by the one introduced in [12], and those proposed in the subsequently reported SUPG and PSPG methods. A number of τs, dependent upon spatial and temporal discretizations, were introduced and tested in [13]. More recently, τs which are

applicable to higher-order elements were proposed in [14]. Ways to calculate τs from the element-level matrices and vectors were first introduced in [15]. These new definitions are expressed in terms of the ratios of the norms of the relevant matrices or vectors. They take into account the local length scales, advection field and the element-level Reynolds number. Based on these definitions, a τ can be calculated for each element, or even for each element node or degree of freedom or element equation. Certain variations and complements of these new τs were described in [16], [17], [18]. In later sections, we will describe, for the semi-discrete and space–time formulations of the advection–diffusion equation and the Navier–Stokes equations of incompressible flows, some of these new ways of calculating the stabilization parameters. These stabilization parameters are based on the local length scales for the advection- and diffusion-dominated limits.

The Deforming-Spatial-Domain/Stabilized Space–Time (DSD/SST) formulation [1], developed for moving boundaries and interfaces, is an interface-tracking technique, where the finite element formulation of the problem is written over its space–time domain. At each time step the locations of the interfaces are calculated as part of the overall solution. As the spatial domain occupied by the fluid changes its shape in time, the mesh needs to be updated. In general, this is accomplished by moving the mesh with the motion of the nodes governed by the equations of elasticity, and full or partial remeshing (i.e., generating a new set of elements, and sometimes also a new set of nodes) as needed. The stabilized space–time formulations were used earlier by other researchers to solve problems with fixed spatial domains (see for example [19]).

In computation of two-fluid flows with interface-tracking techniques, sometimes the interface might be too complex or unsteady to track while keeping the frequency of remeshing at an acceptable level. Not being able to reduce the frequency of remeshing in 3D might introduce overwhelming mesh generation and projection costs, making the computations with the interface-tracking technique no longer feasible. In such cases, interface-capturing techniques, which do not normally require costly mesh update steps, could be used with the understanding that the interface will not be represented as accurately as we would have with an interface-tracking technique. Because they do not require mesh update, the interface-capturing techniques are more flexible than the interface-tracking techniques. However, for comparable levels of spatial discretization, interface-capturing methods yield less accurate representation of the interface. These methods can be used as practical alternatives in carrying out the simulations when compromising the accurate representation of the interfaces becomes less of a concern than facing major difficulties in updating the mesh to track such interfaces. The need to increase the accuracy of our interface-capturing techniques without adding a major computational cost lead us to seeking techniques with a different kind of "tracking". The Enhanced-Discretization Interface-Capturing Technique (EDICT) was first introduced in [20], [21] to increase accuracy in representing an interface.

The governing equations and core methods, including the determination of the stabilization parameters, are described in Sections II–VIII. The EDICT and the other methods developed to increase the scope and accuracy of the core methods are described in Sections IX–XIV.

II. GOVERNING EQUATIONS

Let $\Omega_t \subset \mathbb{R}^{n_{sd}}$ be the spatial fluid mechanics domain with boundary Γ_t at time $t \in (0,T)$, where the subscript t indicates the time-dependence of the spatial domain. The Navier–Stokes equations of incompressible flows can be written on Ω_t and $\forall t \in (0,T)$ as

$$\rho(\frac{\partial \mathbf{u}}{\partial t} + \mathbf{u}\cdot\boldsymbol{\nabla}\mathbf{u} - \mathbf{f}) - \boldsymbol{\nabla}\cdot\boldsymbol{\sigma} = 0, \tag{1}$$

$$\boldsymbol{\nabla}\cdot\mathbf{u} = 0, \tag{2}$$

where ρ, $\mathbf{u}$ and $\mathbf{f}$ are the density, velocity and the external force, respectively. The stress tensor $\boldsymbol{\sigma}$ is defined as

$$\boldsymbol{\sigma}(p,\mathbf{u}) = -p\mathbf{I} + 2\mu\boldsymbol{\varepsilon}(\mathbf{u}), \qquad \boldsymbol{\varepsilon}(\mathbf{u}) = \frac{1}{2}((\boldsymbol{\nabla}\mathbf{u}) + (\boldsymbol{\nabla}\mathbf{u})^T). \tag{3}$$

Here p is the pressure, $\mathbf{I}$ is the identity tensor, $\mu = \rho\nu$ is the viscosity, ν is the kinematic viscosity, and $\boldsymbol{\varepsilon}(\mathbf{u})$ is the strain-rate tensor. The essential and natural boundary conditions for Eq. (1) are represented as

$$\mathbf{u} = \mathbf{g} \text{ on } (\Gamma_{\mathrm{t}})_{\mathrm{g}}, \qquad \mathbf{n}\cdot\boldsymbol{\sigma} = \mathbf{h} \text{ on } (\Gamma_{\mathrm{t}})_{\mathrm{h}}, \tag{4}$$

where $(\Gamma_t)_g$ and $(\Gamma_t)_h$ are complementary subsets of the boundary Γ_t, $\mathbf{n}$ is the unit normal vector, and $\mathbf{g}$ and $\mathbf{h}$ are given functions. A divergence-free velocity field $\mathbf{u}_0(\mathbf{x})$ is specified as the initial condition.

If there are no moving boundaries or interfaces, the spatial domain does not need to change with respect to time, and the subscript t can be dropped from Ω_t and Γ_t. This might be the case even for flows with moving boundaries and interfaces, if the formulation is not based on definining the spatial domain to be the part of the space occupied by the fluid(s). For example, fluid–fluid interfaces can be modeled over a fixed spatial domain by assuming that the domain is occupied by two immiscible fluids, A and B, with densities ρ_A and ρ_B and viscosities μ_A and μ_B. In this approach, a free-surface problem can be modeled as a special case where Fluid B is irrelevant and assigned a sufficiently low density. An interface function ϕ serves as the marker identifying Fluid A and B with the definition $\phi = \{1 \text{ for Fluid A and } 0 \text{ for Fluid B}\}$. The interface between the two fluids is approximated to be at $\phi = 0.5$. In this context, ρ and μ are defined as $\rho = \phi\rho_A + (1-\phi)\rho_B$ and $\mu = \phi\mu_A + (1-\phi)\mu_B$. The evolution of the interface function ϕ, and consequently the motion of the interface, is governed by a time-dependent advection equation, written on Ω and $\forall t \in (0,T)$ as

$$\frac{\partial\phi}{\partial t} + \mathbf{u}\cdot\boldsymbol{\nabla}\phi = 0. \tag{5}$$

To generalize Eq. (5), let us consider the following time-dependent advection–diffusion equation, written on Ω and $\forall t \in (0,T)$ as

$$\frac{\partial\phi}{\partial t} + \mathbf{u}\cdot\boldsymbol{\nabla}\phi - \boldsymbol{\nabla}\cdot(\nu\boldsymbol{\nabla}\phi) = 0, \tag{6}$$

where ϕ represents the quantity being transported (e.g., temperature, concentration), and ν is the diffusivity. The essential and natural boundary conditions associated with Eq. (6) are represented as

$$\phi = \mathrm{g} \text{ on } \Gamma_{\mathrm{g}}, \qquad \mathbf{n}\cdot\nu\boldsymbol{\nabla}\phi = \mathrm{h} \text{ on } \Gamma_{\mathrm{h}}. \tag{7}$$

A function $\phi_0(\mathbf{x})$ is specified as the initial condition.

III. STABILIZED SEMI-DISCRETE FORMULATIONS

A. Advection–diffusion equation

Let us assume that we have constructed some suitably-defined finite-dimensional trial solution and test function spaces $\mathcal{S}_\phi^h$ and $\mathcal{V}_\phi^h$. The stabilized finite element formulation of Eq. (6) can then be written as follows: find $\phi^h \in \mathcal{S}_\phi^h$ such that $\forall w^h \in \mathcal{V}_\phi^h$:

$$\begin{aligned}
&\int_\Omega w^h\left(\frac{\partial\phi^h}{\partial t} + \mathbf{u}^h\cdot\boldsymbol{\nabla}\phi^h\right)d\Omega + \int_\Omega \boldsymbol{\nabla}w^h\cdot\nu\boldsymbol{\nabla}\phi^h d\Omega - \int_{\Gamma_{\mathrm{h}}} w^h\mathrm{h}^h d\Gamma \\
+ &\sum_{e=1}^{n_{el}}\int_{\Omega^e}\tau_{\mathrm{SUPG}}\mathbf{u}^h\cdot\boldsymbol{\nabla}w^h\left(\frac{\partial\phi^h}{\partial t} + \mathbf{u}^h\cdot\boldsymbol{\nabla}\phi^h - \boldsymbol{\nabla}\cdot\left(\nu\boldsymbol{\nabla}\phi^h\right)\right)d\Omega = 0.
\end{aligned} \tag{8}$$

Here n_{el} is the number of elements, Ω^e is the domain for element e, and τ_{SUPG} is the SUPG stabilization parameter. For various ways of calculating τ_{SUPG}, see [15], [16], [17], [18].

B. Navier–Stokes equations of incompressible flows

Given Eqs. (1)–(2), let us assume that we have some suitably-defined finite-dimensional trial solution and test function spaces for velocity and pressure: $\mathcal{S}_{\mathbf{u}}^h$, $\mathcal{V}_{\mathbf{u}}^h$, $\mathcal{S}_p^h$ and $\mathcal{V}_p^h = \mathcal{S}_p^h$. The stabilized finite element formulation of Eqs. (1)–(2) can then be written as follows: find $\mathbf{u}^h \in \mathcal{S}_{\mathbf{u}}^h$ and $p^h \in \mathcal{S}_p^h$ such that $\forall \mathbf{w}^h \in \mathcal{V}_{\mathbf{u}}^h$ and $q^h \in \mathcal{V}_p^h$:

$$\begin{aligned}
&\int_\Omega \mathbf{w}^h \cdot \rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{u}^h - \mathbf{f}^h \right) d\Omega + \int_\Omega \boldsymbol{\varepsilon}(\mathbf{w}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h) d\Omega - \int_{\Gamma_{\text{h}}} \mathbf{w}^h \cdot \mathbf{h}^h d\Gamma \\
+ &\int_\Omega q^h \boldsymbol{\nabla} \cdot \mathbf{u}^h d\Omega + \sum_{e=1}^{n_{el}} \int_{\Omega^e} \frac{1}{\rho} [\tau_{\text{SUPG}} \rho \mathbf{u}^h \cdot \nabla \mathbf{w}^h + \tau_{\text{PSPG}} \nabla q^h] \cdot \left[\text{Ł}(p^h, \mathbf{u}^h) - \rho \mathbf{f}^h \right] d\Omega \\
+ &\sum_{e=1}^{n_{el}} \int_{\Omega^e} \nu_{\text{LSIC}} \boldsymbol{\nabla} \cdot \mathbf{w}^h \rho \boldsymbol{\nabla} \cdot \mathbf{u}^h d\Omega \;=\; 0,
\end{aligned} \tag{9}$$

where

$$\text{Ł}(q^h, \mathbf{w}^h) = \rho \left(\frac{\partial \mathbf{w}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{w}^h \right) - \boldsymbol{\nabla} \cdot \boldsymbol{\sigma}(q^h, \mathbf{w}^h). \tag{10}$$

Here τ_{PSPG} and ν_{LSIC} are the PSPG and LSIC (least-squares on incompressibility constraint) stabilization parameters. For various ways of calculating τ_{PSPG} and ν_{LSIC}, see [15], [16], [17], [18].

IV. DSD/SST FINITE ELEMENT FORMULATION

In the DSD/SST method [1], the finite element formulation of the governing equations is written over a sequence of N space–time slabs Q_n, where Q_n is the slice of the space–time domain between the time levels t_n and t_{n+1}. At each time step, the integrations are performed over Q_n. The space–time finite element interpolation functions are continuous within a space–time slab, but discontinuous from one space–time slab to another. The notation $(\cdot)_n^-$ and $(\cdot)_n^+$ denotes the function values at t_n as approached from below and above. Each Q_n is decomposed into elements Q_n^e, where $e = 1, 2, \ldots, (n_{el})_n$. The subscript n used with n_{el} is for the general case in which the number of space–time elements may change from one space–time slab to another. The Dirichlet- and Neumann-type boundary conditions are enforced over $(P_n)_{\text{g}}$ and $(P_n)_{\text{h}}$, the complementary subsets of the lateral boundary of the space–time slab. The finite element trial function spaces $(\mathcal{S}_{\mathbf{u}}^h)_n$ for velocity and $(\mathcal{S}_p^h)_n$ for pressure, and the test function spaces $(\mathcal{V}_{\mathbf{u}}^h)_n$ and $(\mathcal{V}_p^h)_n = (\mathcal{S}_p^h)_n$ are defined by using, over Q_n, first-order polynomials in both space and time. The DSD/SST formulation is written as follows: given $(\mathbf{u}^h)_n^-$, find $\mathbf{u}^h \in (\mathcal{S}_{\mathbf{u}}^h)_n$ and $p^h \in (\mathcal{S}_p^h)_n$ such that $\forall \mathbf{w}^h \in (\mathcal{V}_{\mathbf{u}}^h)_n$ and $q^h \in (\mathcal{V}_p^h)_n$:

$$\begin{aligned}
&\int_{Q_n} \mathbf{w}^h \cdot \rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{u}^h - \mathbf{f}^h \right) dQ + \int_{Q_n} \boldsymbol{\varepsilon}(\mathbf{w}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h) dQ \\
- &\int_{(P_n)_{\text{h}}} \mathbf{w}^h \cdot \mathbf{h}^h dP + \int_{Q_n} q^h \boldsymbol{\nabla} \cdot \mathbf{u}^h dQ + \int_{\Omega_n} (\mathbf{w}^h)_n^+ \cdot \rho \left((\mathbf{u}^h)_n^+ - (\mathbf{u}^h)_n^- \right) d\Omega \\
+ &\sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \frac{1}{\rho} \left[\tau_{\text{SUPG}} \rho \left(\frac{\partial \mathbf{w}^h}{\partial t} + \mathbf{u}^h \cdot \nabla \mathbf{w}^h \right) + \tau_{\text{PSPG}} \nabla q^h \right] \cdot \left[\text{Ł}(p^h, \mathbf{u}^h) - \rho \mathbf{f}^h \right] dQ \\
+ &\sum_{e=1}^{n_{el}} \int_{Q_n^e} \nu_{\text{LSIC}} \boldsymbol{\nabla} \cdot \mathbf{w}^h \rho \boldsymbol{\nabla} \cdot \mathbf{u}^h dQ = 0.
\end{aligned} \tag{11}$$

This formulation is applied to all space–time slabs $Q_0, Q_1, Q_2, \ldots, Q_{N-1}$, starting with $(\mathbf{u}^h)_0^- = \mathbf{u}_0$. For an earlier, detailed reference on the DSD/SST formulation see [1].

Similarly, the DSD/SST formulation of Eq. (6) can be written as follows:

$$\begin{aligned} & \int_{Q_n} w^h \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \phi^h \right) dQ + \int_{Q_n} \boldsymbol{\nabla} w^h \cdot \nu \boldsymbol{\nabla} \phi^h dQ \\ - & \int_{(P_n)_{\mathrm{h}}} w^h \mathrm{h}^h dP + \int_{\Omega_n} (w^h)_n^+ \left((\phi^h)_n^+ - (\phi^h)_n^- \right) d\Omega \\ + & \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \tau_{\mathrm{SUPG}} \left(\frac{\partial w^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} w^h \right) \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \phi^h - \boldsymbol{\nabla} \cdot \left(\nu \boldsymbol{\nabla} \phi^h \right) \right) dQ = 0. \end{aligned} \tag{12}$$

V. CALCULATION OF THE STABILIZATION PARAMETERS FOR INCOMPRESSIBLE FLOWS

Various ways of calculating the stabilization parameters for incompressible flows were covered earlier in detail in [17], [18]. In this section we focus on the versions of the stabilization parameters (τs) denoted by the subscript UGN, namely the UGN/RGN-based stabilization parameters. For this purpose, we first define the unit vectors $\mathbf{s}$ and $\mathbf{r}$:

$$\mathbf{s} = \frac{\mathbf{u}^h}{\|\mathbf{u}^h\|} , \tag{13}$$

$$\mathbf{r} = \frac{\boldsymbol{\nabla} \|\mathbf{u}^h\|}{\| \boldsymbol{\nabla} \|\mathbf{u}^h\| \|} , \tag{14}$$

where, for the advection–diffusion equation, in Eq. (14) we replace $\|\mathbf{u}^h\|$ with $|\phi^h|$.

We define the components of $(\tau_{\mathrm{SUPG}})_{\mathrm{UGN}}$ corresponding to the advection-, transient- and diffusion-dominated limits as follows:

$$\tau_{\mathrm{SUGN1}} = \left(\sum_{a=1}^{n_{en}} |\mathbf{u}^h \cdot \boldsymbol{\nabla} N_a| \right)^{-1} , \tag{15}$$

$$\tau_{\mathrm{SUGN2}} = \frac{\Delta t}{2}, \tag{16}$$

$$\tau_{\mathrm{SUGN3}} = \frac{h_{\mathrm{RGN}}^2}{4\nu} , \tag{17}$$

where n_{en} is the number of element nodes and N_a is the interpolation function associated with node a, and the "element length" h_{RGN} is defined as

$$h_{\mathrm{RGN}} = 2 \left(\sum_{a=1}^{n_{en}} |\mathbf{r} \cdot \boldsymbol{\nabla} N_a| \right)^{-1} . \tag{18}$$

Based on Eq. (15), we define the "element length" h_{UGN} as

$$h_{\mathrm{UGN}} = 2 \|\mathbf{u}^h\| \ \tau_{\mathrm{SUGN1}} . \tag{19}$$

Although writing a direct expression for τ_{SUGN1} as given by Eq. (15) was pointed out in [17], [18], the element length definition one obtains by combining Eq. (15) and Eq. (19) was first introduced (as a direct expression for h_{UGN}) in [12]. The expression for h_{RGN} as given by Eq. (18) was first introduced

in [16]. We note that $h_{\rm UGN}$ and $h_{\rm RGN}$ can be viewed as the local length scales corresponding to the advection- and diffusion-dominated limits, respectively.

We now define $(\tau_{\rm SUPG})_{\rm UGN}$, $(\tau_{\rm PSPG})_{\rm UGN}$, and $(\nu_{\rm LSIC})_{\rm UGN}$ as follows:

$$(\tau_{\rm SUPG})_{\rm UGN} = \left(\frac{1}{\tau_{\rm SUGN1}^r} + \frac{1}{\tau_{\rm SUGN2}^r} + \frac{1}{\tau_{\rm SUGN3}^r} \right)^{-\frac{1}{r}}, \tag{20}$$

$$(\tau_{\rm PSPG})_{\rm UGN} = (\tau_{\rm SUPG})_{\rm UGN}, \tag{21}$$

$$(\nu_{\rm LSIC})_{\rm UGN} = (\tau_{\rm SUPG})_{\rm UGN} \left\| \mathbf{u}^h \right\|^2 . \tag{22}$$

Eq. (20) is based on the inverse of $(\tau_{\rm SUPG})_{\rm UGN}$ being defined as the r-norm of the vector with components $\frac{1}{\tau_{\rm SUGN1}}$, $\frac{1}{\tau_{\rm SUGN2}}$ and $\frac{1}{\tau_{\rm SUGN3}}$. We note that the higher the integer r is, the sharper the switching between $\tau_{\rm SUGN1}$, $\tau_{\rm SUGN2}$ and $\tau_{\rm SUGN3}$ becomes. This "*r-switch*" was introduced in [15]. Typically, we set $r = 2$. The expressions for $\tau_{\rm SUGN3}$ and $(\nu_{\rm LSIC})_{\rm UGN}$, given respectively by Eqs. (17) and (22), were proposed in [17], [18]. We define the "SUPG viscosity" $\nu_{\rm SUPG}$ as

$$\nu_{\rm SUPG} = \tau_{\rm SUPG} \|\mathbf{u}^h\|^2. \tag{23}$$

The space–time versions of $\tau_{\rm SUGN1}$, $\tau_{\rm SUGN2}$, $\tau_{\rm SUGN3}$, $(\tau_{\rm SUPG})_{\rm UGN}$, $(\tau_{\rm PSPG})_{\rm UGN}$, and $(\nu_{\rm LSIC})_{\rm UGN}$, given respectively by Eqs. (15), (16), (17), (20), (21), and (22), were defined in [17], [18] as follows:

$$\tau_{\rm SUGN12} = \left(\sum_{a=1}^{n_{en}} \left| \frac{\partial N_a}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} N_a \right| \right)^{-1}, \tag{24}$$

$$\tau_{\rm SUGN3} = \frac{h_{\rm RGN}^2}{4\nu} , \tag{25}$$

$$(\tau_{\rm SUPG})_{\rm UGN} = \left(\frac{1}{\tau_{\rm SUGN12}^r} + \frac{1}{\tau_{\rm SUGN3}^r} \right)^{-\frac{1}{r}}, \tag{26}$$

$$(\tau_{\rm PSPG})_{\rm UGN} = (\tau_{\rm SUPG})_{\rm UGN} , \tag{27}$$

$$(\nu_{\rm LSIC})_{\rm UGN} = (\tau_{\rm SUPG})_{\rm UGN} \left\| \mathbf{u}^h \right\|^2 . \tag{28}$$

Here, n_{en} is the number of nodes for the space–time element, and N_a is the space–time interpolation function associated with node a.

Remark 1: It was remarked in [15], [16], [17], [18] that in marching from time level n to $n+1$, there are advantages in calculating the τs from the flow field at time level n. That is

$$\tau \leftarrow \tau_n , \tag{29}$$

where τ is the stabilization parameter to be used in marching from time level n to $n+1$, and τ_n is the stabilization parameter calculated from the flow field at time level n. One of the main advantages in doing that, as it was pointed out in [16], [17], [18], is avoiding another level of nonlinearity coming from the way τs are defined. In general, we suggest making τs less dependent on short-term variations in the flow field. For this purpose, we propose a recursive time-averaging approach in determining the τs to be used in marching from time level n to $n+1$:

$$\tau \leftarrow z_1 \, \tau_n + z_2 \, \tau_{n-1} + (1 - z_1 - z_2) \, \tau , \tag{30}$$

where τ_n and τ_{n-1} are the stabilization parameters calculated from the flow field at time levels n and $n-1$, and the τ on the right-hand-side is the stabilization parameter that was used in marching from time level $n-1$ to n. The magnitudes and the number of the "averaging parameters" z_1, z_2, ... can be adjusted to create the desired outcome in terms of giving more weight to recently calculated τs or making the averaging closer to being a trailing average.

VI. DISCONTINUITY-CAPTURING DIRECTIONAL DISSIPATION (DCDD)

As an alternative to the LSIC stabilization, we proposed in [16], [17], [18] the Discontinuity-Capturing Directional Dissipation (DCDD) stabilization. In describing the DCDD stabilization, we first define the "DCDD viscosity" $\nu_{\rm DCDD}$ and the DCDD stabilization parameter $\tau_{\rm DCDD}$:

$$\nu_{\rm DCDD} = \tau_{\rm DCDD} \|\mathbf{u}^h\|^2, \tag{31}$$

$$\tau_{\rm DCDD} = \frac{h_{\rm DCDD}}{2u_{\rm cha}} \frac{\| \boldsymbol{\nabla} \|\mathbf{u}^h\| \| h_{\rm DCDD}}{u_{\rm ref}} , \tag{32}$$

where

$$h_{\rm DCDD} = h_{\rm RGN} . \tag{33}$$

Here $u_{\rm ref}$ is a reference velocity (such as $\|\mathbf{u}^h\|$ at the inflow, or the difference between the estimated maximum and minimum values of $\|\mathbf{u}^h\|$), and $u_{\rm cha}$ is a characteristic velocity (such as $u_{\rm ref}$ or $\|\mathbf{u}^h\|$). We propose to set $u_{\rm cha} = u_{\rm ref}$.

Then the DCDD stabilization is defined as

$$S_{\rm DCDD} = \sum_{e=1}^{n_{el}} \int_{\Omega^e} \rho \boldsymbol{\nabla} \mathbf{w}^h : \left([\nu_{\rm DCDD} \mathbf{rr} - \boldsymbol{\kappa}_{\rm CORR}] \cdot \boldsymbol{\nabla} \mathbf{u}^h \right) d\Omega , \tag{34}$$

where $\boldsymbol{\kappa}_{\rm CORR}$ was defined in [16], [17], [18] as

$$\boldsymbol{\kappa}_{\rm CORR} = \nu_{\rm DCDD} (\mathbf{r} \cdot \mathbf{s})^2 \mathbf{ss} . \tag{35}$$

As a possible alternative, we propose

$$\boldsymbol{\kappa}_{\rm CORR} = \nu_{\rm SUPG} (\mathbf{r} \cdot \mathbf{s})^2 \mathbf{rr} . \tag{36}$$

As two other possible alternatives, we propose

$$\boldsymbol{\kappa}_{\rm CORR} = switch \left(\nu_{\rm SUPG} , \nu_{\rm DCDD} (\mathbf{r} \cdot \mathbf{s})^2 \right) \mathbf{ss} , \tag{37}$$

$$\boldsymbol{\kappa}_{\rm CORR} = switch \left(\nu_{\rm DCDD} , \nu_{\rm SUPG} (\mathbf{r} \cdot \mathbf{s})^2 \right) \mathbf{rr} , \tag{38}$$

where the "*switch*" function is defined as the "*min*" function:

$$switch\,(\alpha, \beta) = min\,(\alpha, \beta) \tag{39}$$

or as the "*r-switch*" given in Section V :

$$switch\,(\alpha, \beta) = \left(\frac{1}{\alpha^r} + \frac{1}{\beta^r} \right)^{-\frac{1}{r}} . \tag{40}$$

Remark 2: Remark 1 applies also to the calculation of $\nu_{\rm DCDD}$.

VII. CALCULATION OF THE STABILIZATION PARAMETERS FOR COMPRESSIBLE FLOWS AND SHOCK-CAPTURING

The SUPG formulation for compressible flows was first introduced, in the context of conservation variables, in [8] and [9]. Here we will call that formulation "$(SUPG)_{82}$". In this section, in the context of the $(SUPG)_{82}$ formulation and based on the ideas we discussed in Sections V and VI, we propose alternative ways of calculating the stabilization parameters and defining the shock-capturing terms. For this purpose, we first define the conservation variables vector as $\mathbf{U} = (\rho, \rho u_1, \rho u_2, \rho u_3, \rho e)$ (where e is the total energy per unit volume), associate to it a test vector-function $\mathbf{W}$, define the acoustic speed as c, and define the unit vector $\mathbf{j}$ as

$$\mathbf{j} = \frac{\boldsymbol{\nabla}\rho^h}{\parallel \boldsymbol{\nabla}\rho^h \parallel} \ . \tag{41}$$

As the first alternative in computing τ_{SUGN1} for each component of the test vector-function $\mathbf{W}$, we propose to define ${\tau^\rho}_{\text{SUGN1}}$, ${\tau^u}_{\text{SUGN1}}$ and ${\tau^e}_{\text{SUGN1}}$ (associated with ρ, $\rho\mathbf{u}$ and ρe, respectively) by using the expression given by Eq. (15):

$${\tau^\rho}_{\text{SUGN1}} = {\tau^u}_{\text{SUGN1}} = {\tau^e}_{\text{SUGN1}} = \left(\sum_{a=1}^{n_{en}} |\mathbf{u}^h \cdot \boldsymbol{\nabla} N_a| \right)^{-1} \ . \tag{42}$$

As the second alternative, we propose to use the following definition:

$${\tau^\rho}_{\text{SUGN1}} = {\tau^u}_{\text{SUGN1}} = {\tau^e}_{\text{SUGN1}} = \left(\sum_{a=1}^{n_{en}} \left(c \ |\mathbf{j} \cdot \boldsymbol{\nabla} N_a| + |\mathbf{u}^h \cdot \boldsymbol{\nabla} N_a| \right) \right)^{-1} \ . \tag{43}$$

In computing τ_{SUGN2}, we propose to use the expression given by Eq. (16):

$${\tau^\rho}_{\text{SUGN2}} = {\tau^u}_{\text{SUGN2}} = {\tau^e}_{\text{SUGN2}} = \frac{\Delta t}{2} \ . \tag{44}$$

In computing τ_{SUGN3}, we propose to define ${\tau^u}_{\text{SUGN3}}$ by using the expression given by Eq. (17):

$${\tau^u}_{\text{SUGN3}} = \frac{h^2_{\text{RGN}}}{4\nu} \ . \tag{45}$$

We propose to define ${\tau^e}_{\text{SUGN3}}$ as

$${\tau^e}_{\text{SUGN3}} = \frac{\left({h^e}_{\text{RGN}}\right)^2}{4\nu^e} \ , \tag{46}$$

where ν^e is the "kinematic viscosity" for the energy equation,

$${h^e}_{\text{RGN}} = 2 \left(\sum_{a=1}^{n_{en}} |\mathbf{r}^e \cdot \boldsymbol{\nabla} N_a| \right)^{-1} , \tag{47}$$

$$\mathbf{r}^e = \frac{\boldsymbol{\nabla}\theta^h}{\parallel \boldsymbol{\nabla}\theta^h \parallel} \ , \tag{48}$$

and θ is the temperature. We define $(\tau^{\rho}_{\text{SUPG}})_{\text{UGN}}$, $(\tau^{u}_{\text{SUPG}})_{\text{UGN}}$ and $(\tau^{e}_{\text{SUPG}})_{\text{UGN}}$ by using the "*r-switch*" given in Section V :

$$(\tau^{\rho}_{\text{SUPG}})_{\text{UGN}} = \left(\frac{1}{(\tau^{\rho}{}_{\text{SUGN1}})^r} + \frac{1}{(\tau^{\rho}{}_{\text{SUGN2}})^r} \right)^{-\frac{1}{r}}, \tag{49}$$

$$(\tau^{u}_{\text{SUPG}})_{\text{UGN}} = \left(\frac{1}{(\tau^{u}{}_{\text{SUGN1}})^r} + \frac{1}{(\tau^{u}{}_{\text{SUGN2}})^r} + \frac{1}{(\tau^{u}{}_{\text{SUGN3}})^r} \right)^{-\frac{1}{r}}, \tag{50}$$

$$(\tau^{e}_{\text{SUPG}})_{\text{UGN}} = \left(\frac{1}{(\tau^{e}{}_{\text{SUGN1}})^r} + \frac{1}{(\tau^{e}{}_{\text{SUGN2}})^r} + \frac{1}{(\tau^{e}{}_{\text{SUGN3}})^r} \right)^{-\frac{1}{r}}. \tag{51}$$

In defining the shock-capturing term, we first define the "shock-capturing viscosity" ν_{SHOC}:

$$\nu_{\text{SHOC}} = \tau_{\text{SHOC}} (u_{\text{int}})^2, \tag{52}$$

where

$$\tau_{\text{SHOC}} = \frac{h_{\text{SHOC}}}{2u_{\text{cha}}} \left(\frac{\parallel \boldsymbol{\nabla}\rho^h \parallel h_{\text{SHOC}}}{\rho_{\text{ref}}} \right)^{\beta}, \tag{53}$$

$$h_{\text{SHOC}} = h_{\text{JGN}} \ , \tag{54}$$

$$h_{\text{JGN}} = 2 \left(\sum_{a=1}^{n_{en}} |\mathbf{j} \cdot \boldsymbol{\nabla} N_a| \right)^{-1}. \tag{55}$$

Here ρ_{ref} is a reference density (such as ρ^h at the inflow, or the difference between the estimated maximum and minimum values of ρ^h), u_{cha} is a characteristic velocity (such as u_{ref} or $\|\mathbf{u}^h\|$ or acoustic speed c), and u_{int} is an intrinsic velocity (such as u_{cha} or $\|\mathbf{u}^h\|$ or acoustic speed c). We propose to set $u_{\text{int}} = u_{\text{cha}} = u_{\text{ref}}$. The parameter β influences the smoothness of the shock-front. We set $\beta = 1$ for smoother shocks and $\beta = 2$ for sharper shocks (in return for tolerating possible overshoots and undershoots). As a compromise between the $\beta = 1$ and $\beta = 2$ selections, we propose the following averaged expression for τ_{SHOC} :

$$\tau_{\text{SHOC}} = \frac{1}{2} \Big((\tau_{\text{SHOC}})_{\beta=1} + (\tau_{\text{SHOC}})_{\beta=2} \Big) \ . \tag{56}$$

As an alternate way, we also propose to calculate ν_{SHOC} by using the following expression:

$$\nu_{\text{SHOC}} = \left\| \mathbf{Y}^{-1}\mathbf{Z} \right\| \left(\sum_{i=1}^{n_{sd}} \left\| \mathbf{Y}^{-1} \frac{\partial \mathbf{U}^h}{\partial x_i} \right\|^2 \right)^{\beta/2-1} \left(\frac{h_{\text{SHOC}}}{2} \right)^{\beta}, \tag{57}$$

where $\mathbf{Y}$ is a diagonal scaling matrix constructed from the reference values of the components of $\mathbf{U}$:

$$\mathbf{Y} = \begin{bmatrix} (U_1)_{\text{ref}} & 0 & 0 & 0 & 0 \\ 0 & (U_2)_{\text{ref}} & 0 & 0 & 0 \\ 0 & 0 & (U_3)_{\text{ref}} & 0 & 0 \\ 0 & 0 & 0 & (U_4)_{\text{ref}} & 0 \\ 0 & 0 & 0 & 0 & (U_5)_{\text{ref}} \end{bmatrix}, \tag{58}$$

$$\mathbf{Z} = \frac{\partial \mathbf{U}^h}{\partial t} + \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} \tag{59}$$

$$OR$$

$$\mathbf{Z} = \mathbf{A}_i^h \frac{\partial \mathbf{U}^h}{\partial x_i} \ , \tag{60}$$

and we set $\beta = 1$ or $\beta = 2$. Here

$$\mathbf{A}_i = \frac{\partial \mathbf{F}_i}{\partial \mathbf{U}} \ , \tag{61}$$

where $\mathbf{F}_i$ is the Euler flux vector corresponding to the ith spatial dimension. As a variation of the expression given by Eq. 57, we propose for ν_{SHOC} the following expression:

$$\nu_{\text{SHOC}} = \left\| \mathbf{Y}^{-1}\mathbf{Z} \right\| \left(\sum_{i=1}^{n_{sd}} \left\| \mathbf{Y}^{-1} \frac{\partial \mathbf{U}^h}{\partial x_i} \right\|^2 \right)^{\beta/2-1} \left\| \mathbf{Y}^{-1}\mathbf{U}^h \right\|^{1-\beta} \left(\frac{h_{\text{SHOC}}}{2} \right)^{\beta} . \tag{62}$$

As a compromise between the $\beta = 1$ and $\beta = 2$ selections, we propose the following averaged expression for ν_{SHOC} :

$$\nu_{\text{SHOC}} = \frac{1}{2} \Big((\nu_{\text{SHOC}})_{\beta=1} + (\nu_{\text{SHOC}})_{\beta=2} \Big) \ . \tag{63}$$

We can also calculate, based on Eq. 57, a separate ν_{SHOC} for each component of the test vector-function $\mathbf{W}$:

$$(\nu_{\text{SHOC}})_{\text{I}} = \left| \left(\mathbf{Y}^{-1}\mathbf{Z} \right)_{\text{I}} \right| \left(\sum_{i=1}^{n_{sd}} \left| \left(\mathbf{Y}^{-1} \frac{\partial \mathbf{U}^h}{\partial x_i} \right)_{\text{I}} \right|^2 \right)^{\beta/2-1} \left(\frac{h_{\text{SHOC}}}{2} \right)^{\beta}, \quad I = 1, 2, \ldots n_{sd} + 2 \ . \tag{64}$$

Similarly, a separate ν_{SHOC} for each component of $\mathbf{W}$ can be calculated based on Eq. 62 :

$$(\nu_{\text{SHOC}})_{\text{I}} = \left| \left(\mathbf{Y}^{-1}\mathbf{Z} \right)_{\text{I}} \right| \left(\sum_{i=1}^{n_{sd}} \left| \left(\mathbf{Y}^{-1} \frac{\partial \mathbf{U}^h}{\partial x_i} \right)_{\text{I}} \right|^2 \right)^{\beta/2-1} \left| \left(\mathbf{Y}^{-1}\mathbf{U}^h \right)_{\text{I}} \right|^{1-\beta} \left(\frac{h_{\text{SHOC}}}{2} \right)^{\beta}, \quad I = 1, 2, \ldots n_{sd} + 2 \ . \tag{65}$$

Given ν_{SHOC}, the shock-capturing term is defined as

$$S_{\text{SHOC}} = \sum_{e=1}^{n_{el}} \int_{\Omega^e} \boldsymbol{\nabla} \mathbf{W}^h : \left(\boldsymbol{\kappa}_{\text{SHOC}} \cdot \boldsymbol{\nabla} \mathbf{U}^h \right) d\Omega \ , \tag{66}$$

where $\boldsymbol{\kappa}_{\text{SHOC}}$ is defined as

$$\boldsymbol{\kappa}_{\text{SHOC}} = \nu_{\text{SHOC}} \ \mathbf{I} \ . \tag{67}$$

As a possible alternative, we propose

$$\boldsymbol{\kappa}_{\text{SHOC}} = \nu_{\text{SHOC}} \ \mathbf{jj} \ . \tag{68}$$

If the option given by Eq. 64 or Eq. 65 is exercised, then ν_{SHOC} becomes an $(n_{sd}+2)\times(n_{sd}+2)$ diagonal matrix, and the matrix $\boldsymbol{\kappa}_{\text{SHOC}}$ becomes augmented from an $n_{sd}\times n_{sd}$ matrix to an $(n_{sd}\times(n_{sd}+2))\times((n_{sd}+2)\times n_{sd})$ matrix.

In an attempt to preclude compounding, we propose to modify ν_{SHOC} as follows:

$$\nu_{\text{SHOC}} \;\leftarrow\; \nu_{\text{SHOC}} - switch\left(\tau_{\text{SUPG}}\,(\mathbf{j}\cdot\mathbf{u})^2,\ \tau_{\text{SUPG}}\,(|\mathbf{j}\cdot\mathbf{u}|-c)^2,\ \nu_{\text{SHOC}}\right)\ , \tag{69}$$

where the "*switch*" function is defined as the "*min*" function or as the "*r-switch*" given in Section V. For viscous flows, the above modification would be made separately with each of $\tau^{\rho}_{\text{SUPG}}$, τ^{u}_{SUPG} and τ^{e}_{SUPG}, and this would result in ν_{SHOC} becoming a diagonal matrix even if the option given by Eq. 64 or Eq. 65 is not exercised.

Remark 3: Remark 1 applies also to the calculation of $\tau^{\rho}_{\text{SUPG}}$, τ^{u}_{SUPG} and τ^{e}_{SUPG}, and ν_{SHOC}.

VIII. MESH UPDATE METHODS

How the mesh should be updated depends on several factors, such as the complexity of the interface and overall geometry, how unsteady the interface is, and how the starting mesh was generated. In general, the mesh update could have two components: moving the mesh for as long as it is possible, and full or partial remeshing (i.e., generating a new set of elements, and sometimes also a new set of nodes) when the element distortion becomes too high. In mesh moving strategies, the only rule the mesh motion needs to follow is that at the interface the normal velocity of the mesh has to match the normal velocity of the fluid. Beyond that, the mesh can be moved in any way desired, with the main objective being to reduce the frequency of remeshing. In 3D simulations, if the remeshing requires calling an automatic mesh generator, reducing the cost of automatic mesh generation becomes a major incentive for trying to reduce the frequency of remeshing. Furthermore, when we remesh, we need to project the solution from the old mesh to the new one. This introduces projection errors. Also, in 3D, the computing time consumed by this projection step is not a trivial one. All these factors constitute a strong motivation for designing mesh update strategies which minimize the frequency of remeshing. In some cases where the changes in the shape of the computational domain allow it, a special-purpose mesh moving method can be used in conjunction with a special-purpose mesh generator. In such cases, simulations can be carried out without calling an automatic mesh generator and without solving any additional equations to determine the motion of the mesh. One of the earliest examples of that, 2D computation of sloshing in a laterally vibrating container, can be found in [1]. Extension of that concept to 3D parallel computation of sloshing in a vertically vibrating container can be found in [11].

In general, however, we use an automatic mesh moving scheme. In the automatic mesh moving technique introduced in [22], the motion of the internal nodes is determined by solving the equations of elasticity. As boundary condition, the motion of the nodes at the interfaces is specified to match the normal velocity of the fluid at the interface. Similar mesh moving techniques were used earlier by other researchers (see for example [23]). In [22] the mesh deformation is dealt with selectively based on the sizes of the elements and also the deformation modes in terms of shape and volume changes. Mesh moving techniques with comparable features were later introduced in [24].

In the technique introduced in [22], selective treatment of the mesh deformation based on shape and volume changes is attained by adjusting the relative values of the Lamé constants of the elasticity equations. The objective would be to stiffen the mesh against shape changes more than we stiffen it against volume changes. Selective treatment based on element sizes, on the other hand, is attained by altering the way we account for the Jacobian of the transformation from the element domain to the physical domain. In this case, the objective is to stiffen the smaller elements, which are typically placed near solid surfaces, more than the larger ones.

The method described in [22] was recently augmented in [25], [26] to a more extensive kind, by introducing a stiffening power that determines the degree by which the smaller elements are rendered

stiffer than the larger ones. When the stiffening power is 0.0, the method reduces back to an elasticity model with no Jacobian-based stiffening. When it is 1.0, the method is identical to the one introduced in [22]. In [25], [26] we investigated the optimum values of the stiffening power with the objective of reducing the deformation of the smaller elements. In that context, by varying the stiffening power, we generated a family of mesh moving techniques, and tested those techniques on fluid meshes where the structure underwent three different types of prescribed motion or deformation (translation, rotation, and bending).

In the mesh moving technique introduced in [22], the structured layers of elements generated around solid objects (to fully control the mesh resolution near solid objects and have more accurate representation of the boundary layers) move "glued" to these solid objects, undergoing a rigid-body motion. No equations are solved for the motion of the nodes in these layers, because these nodal motions are not governed by the equations of elasticity. This results in some cost reduction. But more importantly, the user has full control of the mesh resolution in these layers. For early examples of automatic mesh moving combined with structured layers of elements undergoing rigid-body motion with solid objects, see [11]. Earlier examples of element layers undergoing rigid-body motion, in combination with deforming structured meshes, can be found in [1].

In computation of flows with fluid–solid interfaces where the solid is deforming, the motion of the fluid mesh near the interface cannot be represented by a rigid-body motion. Depending on the deformation mode of the solid, we may have to use the automatic mesh moving technique described above. In such cases, presence of very thin fluid elements near the solid surface becomes a challenge for the automatic mesh moving technique. In the Solid-Extension Mesh Moving Technique (SEMMT) [4], [5], [6], we proposed treating those very thin fluid elements almost like an extension of the solid elements. In the SEMMT, in solving the equations of elasticity governing the motion of the fluid nodes, we assign higher rigidity to these thin elements compared to the other fluid elements. Two ways of accomplishing this were proposed in [4], [5], [6]; solving the elasticity equations for the nodes connected to the thin elements separate from the elasticity equations for the other nodes, or together. If we solve them separately, for the thin elements, as boundary conditions at the interface with the other elements, we would use traction-free conditions. In [26], we demonstrated how the SEMMT functions as part of our mesh update method. We employed both of the SEMMT options described above. In [26], we referred to the separate solution option as "SEMMT – Multiple Domain" and the unified solution option as "SEMMT – Single Domain".

IX. ENHANCED-DISCRETIZATION INTERFACE-CAPTURING TECHNIQUE (EDICT)

In the EDICT (Enhanced-Discretization Interface-Capturing Technique) [20], [21], we start with the basic approach of an interface-capturing technique such as the volume of fluid (VOF) method [27]. The Navier-Stokes equations are solved over a non-moving mesh together with the time-dependent advection equation governing the evolution of the interface function ϕ. The trial function spaces corresponding to velocity, pressure and interface function are denoted, respectively, by $(\mathcal{S}_{\mathbf{u}}^h)_n$, $(\mathcal{S}_p^h)_n$, and $(\mathcal{S}_\phi^h)_n$. The weighting function spaces corresponding to the momentum equation, incompressibility constraint and time-dependent advection equation are denoted by $(\mathcal{V}_{\mathbf{u}}^h)_n$, $(\mathcal{V}_p^h)_n$ $(= (\mathcal{S}_p^h)_n)$, and $(\mathcal{V}_\phi^h)_n$. The subscript n in this case allows us to use different spatial discretizations corresponding to different time levels.

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

$$
\begin{aligned}
& \int_\Omega \mathbf{w}_{n+1}^h \cdot \rho \left(\frac{\partial \mathbf{u}^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \mathbf{u}^h - \mathbf{f}^h \right) d\Omega + \int_\Omega \boldsymbol{\varepsilon}(\mathbf{w}_{n+1}^h) : \boldsymbol{\sigma}(p^h, \mathbf{u}^h) d\Omega \\
- & \int_{\Gamma_{\mathrm{h}}} \mathbf{w}_{n+1}^h \cdot \mathbf{h}^h d\Gamma + \int_\Omega q_{n+1}^h \boldsymbol{\nabla} \cdot \mathbf{u}^h d\Omega \\
+ & \sum_{e=1}^{n_{el}} \int_{\Omega^e} \frac{1}{\rho} [\tau_{\mathrm{SUPG}} \rho \mathbf{u}^h \cdot \nabla \mathbf{w}_{n+1}^h + \tau_{\mathrm{PSPG}} \nabla q_{n+1}^h] \cdot \left[\mathbf{Ł}(p^h, \mathbf{u}^h) - \rho \mathbf{f}^h \right] d\Omega \\
+ & \sum_{e=1}^{n_{el}} \int_{\Omega^e} \nu_{\mathrm{LSIC}} \boldsymbol{\nabla} \cdot \mathbf{w}_{n+1}^h \rho \boldsymbol{\nabla} \cdot \mathbf{u}^h d\Omega = 0,
\end{aligned} \tag{70}
$$

$$
\begin{aligned}
& \int_\Omega \psi_{n+1}^h \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \phi^h \right) d\Omega \\
+ & \sum_{e=1}^{n_{el}} \int_{\Omega^e} \tau_\phi \mathbf{u}^h \cdot \boldsymbol{\nabla} \psi_{n+1}^h \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \phi^h \right) d\Omega = 0.
\end{aligned} \tag{71}
$$

Here τ_ϕ is calculated by applying the definition of τ_{SUPG} to Eq. (71).

To increase the accuracy, we use function spaces corresponding to enhanced discretization at and near the interface. A subset of the elements in the base mesh, Mesh-1, are identified as those at and near the interface. A more refined mesh, Mesh-2, is constructed by patching together second-level meshes generated over each element in this subset. The interpolation 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, we construct a third-level mesh, Mesh-3, for the interface function only. The construction of Mesh-3 from Mesh-2 is very similar to the construction of Mesh-2 from Mesh-1. The interpolation functions for the interface function will have three components, each coming from one of these three meshes. We re-define the subsets over which we build Mesh-2 and Mesh-3 not every time step but with sufficient frequency to keep the interface enveloped in. We need to avoid this envelope being too wide or too narrow.

X. MIXED INTERFACE-TRACKING/INTERFACE-CAPTURING TECHNIQUE (MITICT)

In computation of flow problems with fluid–solid interfaces, an interface-tracking technique, where the fluid mesh moves to track the interface, would allow us to have full control of the resolution of the fluid mesh in the boundary layers. With an interface-capturing technique (or an interface representation technique in the more general case), on the other hand, independent of how well the interface geometry is represented, the resolution of the fluid mesh in the boundary layer will be limited by the resolution of the fluid mesh where the interface is. In computation of flow problems with fluid–fluid interfaces where the interface is too complex or unsteady to track while keeping the remeshing frequency under control, interface-capturing techniques, with enhanced-discretization as needed, could be used as more flexible alternatives. Sometimes we may need to solve flow problems with both fluid–solid interfaces and complex or unsteady fluid–fluid interfaces.

MITICT was introduced in [2], [3], [4], primarily for fluid–object interactions with multiple fluids. The class of applications we were targeting were fluid–particle–gas interactions and free-surface flow of fluid–particle mixtures. However, the MITICT can be applied to a larger class of problems, where it is more effective to use an interface-tracking technique to track the solid–fluid interfaces and an interface-capturing technique to capture the fluid–fluid interfaces. The interface-tracking technique is

the DSD/SST formulation (but could as well be the Arbitrary Lagrangian-Eulerian method or other moving mesh methods). The interface-capturing technique rides on this, and is based on solving over a moving mesh, in addition to the Navier–Stokes equations, the advection equation governing the time-evolution of the interface function. The additional DSD/SST formulation is for this advection equation:

$$\int_{Q_n} \psi^h \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \phi^h \right) dQ + \int_{\Omega_n} (\psi^h)_n^+ \left((\phi^h)_n^+ - (\phi^h)_n^- \right) d\Omega$$
$$+ \sum_{e=1}^{(n_{el})_n} \int_{Q_n^e} \tau_\phi \left(\frac{\partial \psi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \psi^h \right) \left(\frac{\partial \phi^h}{\partial t} + \mathbf{u}^h \cdot \boldsymbol{\nabla} \phi^h \right) dQ = 0. \tag{72}$$

This equation, together with Eq. (11), constitute a mixed interface-tracking/interface-capturing technique that would track the solid–fluid interfaces and capture the fluid–fluid interfaces that would be too complex or unsteady to track with a moving mesh. The interface-capturing part of MITICT can be upgraded to the EDICT formulation for more accurate representation of the interfaces captured.

The MITICT can also be used for computation of fluid–structure interactions with multiple fluids or for flows with mechanical components moving in a mixture of two fluids. In more general cases, the MITICT can be used for classes of problems that involve both interfaces that can be accurately tracked with a moving mesh method and interfaces that are too complex or unsteady to be tracked and therefore require an interface-capturing technique.

Remark 4: We propose the MITICT-L as a version of the MITICT with limited mesh moving. In the MITICT-L, mesh moving would be limited to the sufficiently wide regions of the computational domain enveloping the fluid-solid interfaces. These regions would be wide enough so that we can keep the mesh deformation, frequency of remeshing, and frequency of re-defining the mesh-moving regions at reasonable levels. By not moving the mesh outside of these regions, we would significantly reduce the computational cost associated with solving the mesh-moving equations. Furthermore, by using only semi- discrete formulations outside of the mesh-moving regions, we would reduce the computational cost associated with using space-time formulations. Appropriate interface (matching) conditions would be used where two differently treated regions meet.

Remark 5: If the Reynolds number is not high enough to be concerned about the mesh resolution near the solid surfaces, then there is no strong reason for moving the mesh for the purpose of controlling the mesh resolution where the fluid–solid interface is. Depending on the nature of the problem, such flow simulations can still be carried out with the MITICT or MITICT-L. This is because not having high-resolution meshes near the fluid–solid interfaces would reduce the mesh moving burden in terms of the number of equations to be solved and frequency of remeshing. As an alternative to this approach, we propose the Fluid–Solid Interface Locator Technique (FSILT). In the FSILT, we propose to carry out the fluid–solid interface computations over non-moving meshes. We propose to accomplish this by modifying the left-hand-side of Eq. (9) as follows:

$$(LHS\ of\ Eq.\ (9)) - \int_{\Gamma_{\mathrm{FS}}} \mathbf{w}^h \cdot \mathbf{J}_{FS}^h \, d\Gamma = 0, \tag{73}$$

where Γ_{FS} is the fluid–solid interface, which is discretized by the structural interface mesh, and $\mathbf{J}_{FS}^h$ represents the interface stresses acting on the fluid at the fluid–solid interface. This would require projection between the non-moving fluid mesh and the moving structural interface mesh. The structural motion would be governed by the applicable structural mechanics equations (examples: rigid- or deformable-body mechanics equations, and membrane or shell equations), taking into account the interface stresses, $-\mathbf{J}_{FS}^h$, acting on the structure. In conjunction with the additional unknown $\mathbf{J}_{FS}^h$,

we need to add on the following constraint equation:

$$\int_{\Gamma_{\rm FS}} \mathbf{K}^h_{FS} \cdot \left(\mathbf{u}^h_F - \mathbf{u}^h_S\right)\, d\Gamma = 0, \tag{74}$$

where $\mathbf{u}^h_F$ is the fluid velocity evaluated on $\Gamma_{\rm FS}$, $\mathbf{u}^h_S$ is the structural displacement rate, and $\mathbf{K}^h_{FS}$ is the test function (variation of $\mathbf{J}^h_{FS}$). We propose two possible ways to approach this constraint problem. In the penalty formulation approach, we write $\mathbf{J}^h_{FS}$ as

$$\mathbf{J}^h_{FS} = -\lambda_{FS}\big(\mathbf{u}^h_F - \mathbf{u}^h_S\big), \tag{75}$$

where λ_{FS} is a penalty parameter. In the stabilized formulation approach, we modify Eq. (74) as follows:

$$\int_{\Gamma_{\rm FS}} \mathbf{K}^h_{FS} \cdot \left(\mathbf{u}^h_F - \mathbf{u}^h_S\right)\, d\Gamma \;+\; \int_{\Gamma_{\rm FS}} \tau_{\rm FS}\, \frac{1}{\rho_M L_{FS}}\, \mathbf{K}^h_{FS} \cdot \left[\mathbf{J}^h_{FS} - \left(\mathbf{n}^h_A \cdot \boldsymbol{\sigma}^h_A + \mathbf{n}^h_B \cdot \boldsymbol{\sigma}^h_B\right)\right]\, d\Gamma = 0, \tag{76}$$

where we assume that the structure is membrane or shell and that we have two different fluids, Fluid A and Fluid B, on each side the structure. Here $\rho_M = max(\rho_A, \rho_B)$, L_{FS} is a global length scale for the fluid–solid interface, and the stabilization parameter is defined as $\tau_{\rm FS} = switch(\tau_{\rm FS1}, \tau_{\rm FS3})$, with

$$\tau_{\rm FS1} = \frac{h_{\rm FS1}}{2\ u_{\rm cha}}\ , \qquad \tau_{\rm FS3} = \frac{(h_{\rm FS3})^2}{4\ (\mu_M/\rho_M)}\ , \tag{77}$$

where $h_{\rm FS1}$ and $h_{\rm FS3}$ are appropriate local length scales, and $\mu_M = max(\mu_A, \mu_B)$.

XI. EDGE-TRACKED INTERFACE LOCATOR TECHNIQUE (ETILT)

The Edge-Tracked Interface Locator Technique (ETILT) was introduced in [2], [3], [4], to have an interface-capturing technique with better volume conservation properties and sharper representation of the interfaces. To this end, we first define a second finite-dimensional representation of the interface function, namely ϕ^{he}. The added superscript "e" indicates that this is an edge-based representation. With ϕ^{he}, interfaces are represented as collection of positions along element edges crossed by the interfaces (i.e., along the "interface edges"). Nodes belong to "chunks" of Fluid A or Fluid B. An edge either belongs to a chunk of Fluid A or Fluid B or is an interface edge. Each element is either filled fully by a chunk of Fluid A or Fluid B, or is shared by a chunk of Fluid A and a chunk of Fluid B. If an element is shared like that, the shares are determined by the position of the interface along the edges of that element. The base finite element formulation is essentially the one described by Eqs. (70) and (71). Although the ETILT can be used in combination with the EDICT, we assume that we are working here with the plain, non-EDICT versions of Eqs. (70) and (71).

At each time step, given $\mathbf{u}^h_n$ and ϕ^{he}_n, we determine $\mathbf{u}^h_{n+1}$, p^h_{n+1}, and ϕ^{he}_{n+1}. The definitions of ρ and μ are modified to use the edge-based representation of the interface function: $\rho^h = \phi^{he}\rho_A + (1-\phi^{he})\rho_B$, $\mu^h = \phi^{he}\mu_A + (1-\phi^{he})\mu_B$. In marching from time level n to $n+1$, we first calculate ϕ^h_n from ϕ^{he}_n by a least-squares projection:

$$\int_\Omega \psi^h \left(\phi^h_n - \phi^{he}_n\right) d\Omega = 0. \tag{78}$$

To calculate ϕ^h_{n+1}, we use Eq. (71). From ϕ^h_{n+1}, we calculate ϕ^{he}_{n+1} by a combination of a least-squares projection:

$$\int_\Omega (\psi^{he}_{n+1})_P \left((\phi^{he}_{n+1})_P - \phi^h_{n+1}\right) d\Omega = 0, \tag{79}$$

and corrections to enforce volume conservation for all chunks of Fluid A and Fluid B, taking into account the mergers between the chunks and the split of chunks. This volume conservation condition can symbolically be written as $VOL\ (\phi_{n+1}^{he}) = VOL\ (\phi_n^{he})$. Here the subscript P is used for representing the intermediate values following the projection, but prior to the corrections for volume conservation. It can be shown that the projection given by Eq. (79) can be interpreted as locating the interface along the interface edges at positions where $\phi_{n+1}^h = 1/2$.

As an alternative way for computing ϕ_n^h from ϕ_n^{he}, we propose to solve the following problem:

$$\int_{\Omega_{INT}} \psi_n^h \left(\phi_n^h - \phi_n^{he}\right) d\Omega \ + \ \sum_{k=1}^{n_{ie}} \psi_n^h(\mathbf{x}_k)\ \lambda_{PEN} \left(\phi_n^h(\mathbf{x}_k) - 1/2\right) = 0, \tag{80}$$

where n_{ie} is the number of interface edges, $\mathbf{x}_k$ is the coordinate of the interface location along the k^{th} interface edge, λ_{PEN} is a large penalty parameter, and Ω_{INT} is the solution domain. The solution domain is the union of all the elements containing at least one node where the value of ϕ_n^h is unknown. We can assume ϕ_n^h to be unknown only at the nodes of the interface edges, with known values $\phi_n^h = 1$ (for Fluid A) and $\phi_n^h = 0$ (for Fluid B) at all other nodes. We can also augment the number of nodes where ϕ_n^h is unknown and thus enlarge the solution domain. This can be done all the way to the point where $\Omega_{INT} = \Omega$. As another alternative, in Eq. (80) we can replace the least-squares projection term with a slope-minimization term:

$$\int_{\Omega_{INT}} \boldsymbol{\nabla}\psi_n^h \cdot \boldsymbol{\nabla}\phi_n^h \ d\Omega \ + \ \sum_{k=1}^{n_{ie}} \psi_n^h(\mathbf{x}_k)\ \lambda_{PEN} \left(\phi_n^h(\mathbf{x}_k) - 1/2\right) = 0. \tag{81}$$

A 1D version of the way of computing ϕ_n^h from ϕ_n^{he} can be formulated by minimizing $(\phi_n^h - \phi_n^{he})^2$ along "chains" of interface edges:

$$\int_{S_{INT}} \psi_n^h \left(\phi_n^h - \phi_n^{he}\right) ds \ + \ \sum_{k=1}^{n_{ie}} \psi_n^h(\mathbf{x}_k)\ \lambda_{PEN} \left(\phi_n^h(\mathbf{x}_k) - 1/2\right) = 0, \tag{82}$$

where S_{INT} is the collection of all chains of interface edges, and s is the integration coordinate along the interface edges. This is, of course, a simpler formulation, and much of the equations for the unknown nodal values will be uncoupled.

These projections and volume corrections are embedded in the iterative solution technique, and are carried out at each iteration. The iterative solution technique, which is based on the Newton–Raphson method, addresses both the nonlinear and coupled nature of the set of equations that need to be solved at each time step. We now provide more explanation of how the projections and volume corrections would be handled at an iteration step taking us from iterative solution i to $i+1$.

A) In determining $(\phi_{n+1}^{he})_P^{i+1}$ from $(\phi_{n+1}^h)^{i+1}$, in the first step of the projection, the position of the interface along each interface edge is calculated. The calculation for an edge might yield for the interface position a value that is not within the range of values representing that edge. This would imply the following consequences.

i) That interface edge does not remain as an interface edge after the projection.
ii) The node at the end of that edge (in the direction of the interface motion) changes from one fluid to another after the projection.
iii) Different edges among those connecting that node to other nodes might be identified as edges expected to be interface edges after the projection. An edge connecting that node to another node would be identified as an interface edge if the other node belongs to a different fluid. If not, it means

that a chunk of one of the fluids is merging with another chunk of the same fluid. It might also mean, as a special case, that a chunk of fluid is connecting with itself at another point.

In the second step of the projection, the interface positions would be calculated along the newly-identified interface edges and those predicted to remain as interface edges after the first step of the projection. If additional steps of the projection are required, the same procedure would be followed.

B) After the projection is complete, we need to detect the possible occurrence of mergers between chunks and split of chunks. The mergers can be detected as described earlier when we discussed the options related to identification of interface edges following a projection step. To detect the split of chunks, one way is to go through a sorting process. In this process, for each chunk, we start with one of the nodes belonging to that chunk, identify all the nodes connected to that node with edges belonging to that chunk, do the same for the newly-identified nodes, and continue this recursive process until all the connected nodes are identified.

After this sorting is complete, if we still have some nodes left in that chunk, this would mean that the chunk we are inspecting has been split. The recursive process needs to be repeated for the nodes and edges remaining in that chunk, so that any additional splits that chunk might have undergone are detected.

C) After the process of identifying all the Fluid A and Fluid B chunks is complete, we enforce the volume conservation. For each chunk, we compare the volumes corresponding interface locations denoted by $(\phi_{n+1}^{he})^i$ and $(\phi_{n+1}^{he})^{i+1}$. In the cases of mergers and splits, we compare the aggregate volume of a set of chunks corresponding to $(\phi_{n+1}^{he})^i$ and constituting a merger/split group to the aggregate volume of the set of chunks constituting the related merger/split group corresponding to $(\phi_{n+1}^{he})^{i+1}$.

D) The volume conservation for a chunk or a merger/split group would be enforced by inflating or deflating its volume. Let us suppose that multiplying the positive and negative increments along each interface edge by a factor $(1 + x)$ and $(1 - x)$, respectively, results in a volume correction by a factor $(1+y)$, where y and x are of the same sign. We need to determine the value of x, such that the corresponding value of y is sufficiently close to the volume correction needed. This would be done iteratively, and the convergence of these iterations can be accelerated by calculating the numerical derivative of y with respect to x and using that estimate in updating x at every iteration. In some cases, we may not be able to represent an increment along the edge where the interface was located prior to a projection or volume correction iteration. This would happen when an interface edge does not remain as an interface edge after the projection or volume correction iteration. For such cases, we propose to measure the increment along the "extended edge", where the edge is imagined to extend into the neighboring element. The imaginary location of the interface along the extension would be where the extension pierces the "interface plane" passing through the interface locations along the edges of the neighboring element. The increment would be measured as the distance between the imaginary location and the location prior to the projection or volume correction iteration. The new, corrected interface locations along the edges of the neighboring element would be determined by shifting the interface plane parallel to itself until it is pierced by the extended edge at the corrected imaginary location along the extended edge. If an interface location along an edge is affected by more than one shifting plane, then the resultant correction would be calculated by a weighted averaging. If the extension of an edge coincides with another edge, then the increment would simply be measured along the "edge pair", and the interface location would be corrected along the edge pair itself. Interface locations that are not along such edge pairs or riding on shifting interface planes would be corrected by projection from those that are.

XII. ITERATIVE SOLUTION METHODS

The finite element formulations reviewed in the earlier sections fall into two categories: a space–time formulation with moving meshes or a semi-discrete formulation with non-moving meshes. Full discretizations of these formulations lead to coupled, nonlinear equation systems that need to be solved at every time step of the simulation. Whether we are using a space–time formulation or a semi-discrete formulation, we can represent the equation system that needs to be solved as follows:

$$\mathbf{N}\left(\mathbf{d}_{n+1}\right) = \mathbf{F}. \tag{83}$$

Here $\mathbf{d}_{n+1}$ is the vector of nodal unknowns. In a semi-discrete formulation, this vector contains the unknowns associated with marching from time level n to $n+1$. In a space–time formulation, it contains the unknowns associated with the finite element formulation written for the space–time slab Q_n. The time-marching formulations described earlier can also be used for computing a steady-state flow. In such cases time does not have a physical significance, but is only used in time-marching to the steady-state solution.

We solve Eq. (83) with the Newton-Raphson method:

$$\left.\frac{\partial \mathbf{N}}{\partial \mathbf{d}}\right|_{\mathbf{d}_{n+1}^i} \left(\Delta \mathbf{d}_{n+1}^i\right) = \mathbf{F} - \mathbf{N}\left(\mathbf{d}_{n+1}^i\right), \tag{84}$$

where i is the step counter for the Newton-Raphson sequence, and $\Delta\mathbf{d}_{n+1}^i$ is the increment computed for $\mathbf{d}_{n+1}^i$. The linear equation system represented by Eq. (84) needs to be solved at every step of the Newton–Raphson sequence. We can represent Eq. (84) as a linear equation system of the form

$$\mathbf{A}\mathbf{x} \;=\; \mathbf{b}. \tag{85}$$

In the class of computations we typically carry out, this equation system would be too large to solve with a direct method. Therefore we solve it iteratively. At each iteration, we need to compute the residual of this system:

$$\mathbf{r} \;=\; \mathbf{b} - \mathbf{A}\mathbf{x}. \tag{86}$$

This can be achieved in several different ways. The computation can be based on a sparse-matrix storage of $\mathbf{A}$. It can also be based on storing just element-level matrices (element-matrix-based), or even just element-level vectors (element-vector-based). This last strategy is also called the matrix-free technique. After the residual computation, we compute a candidate correction to $\mathbf{x}$ as given by the expression

$$\Delta\mathbf{y} = \mathbf{P}^{-1}\mathbf{r}, \tag{87}$$

where $\mathbf{P}$, the preconditioning matrix, is an approximation to $\mathbf{A}$. $\mathbf{P}$ has to be simple enough to form and factorize efficiently. However, it also has to be sophisticated enough to yield a desirable convergence rate. How to update the solution vector $\mathbf{x}$ by using $\Delta\mathbf{y}$ is also a major subject in iterative solution techniques. Several update methods are available, and we use the GMRES [28] method. In preconditioning design, we developed some advanced preconditioners such as the Clustered-Element-by-Element (CEBE) preconditioner [29] and the mixed CEBE and Cluster Companion (CC) preconditioner [29]. However, our typical computations are based on diagonal and nodal-block-diagonal preconditioners. These are very simple preconditioners, but are also very simple to implement on parallel platforms. More on our parallel implementations can be found in [30].

XIII. MIXED ELEMENT-MATRIX-BASED/ELEMENT-VECTOR-BASED COMPUTATION TECHNIQUE (MMVCT)

The MMVCT was introduced in [2], and was also described in [6]. Consider a nonlinear equation system of the kind given by Eq. (83), re-written in the following form:

$$\begin{aligned} \mathbf{N}_1\left(\mathbf{d}_1,\mathbf{d}_2\right) &= \mathbf{F}_1, \\ \mathbf{N}_2\left(\mathbf{d}_1,\mathbf{d}_2\right) &= \mathbf{F}_2, \end{aligned} \tag{88}$$

where $\mathbf{d}_1$ and $\mathbf{d}_2$ are the vectors of nodal unknowns corresponding to unknown functions $\mathbf{u}_1$ and $\mathbf{u}_2$, respectively. Similarly, we re-write Eq. (85) in the form

$$\begin{aligned} \mathbf{A}_{11}\mathbf{x}_1+\mathbf{A}_{12}\mathbf{x}_2 &= \mathbf{b}_1, \\ \mathbf{A}_{21}\mathbf{x}_1+\mathbf{A}_{22}\mathbf{x}_2 &= \mathbf{b}_2, \end{aligned} \tag{89}$$

where

$$\mathbf{A}_{\beta\gamma}=\frac{\partial \mathbf{N}_\beta}{\partial \mathbf{d}_\gamma}. \tag{90}$$

Re-writing Eqs. (83) and (85) in this fashion would help us recognize or investigate the properties associated with the individual blocks of the equation system. It would also help us explore selective treatment of these blocks during the solution process. For example, in the context of a coupled fluid–structure interaction problem, $\mathbf{u}_1$ and $\mathbf{u}_2$ might be representing the fluid and structure unknowns, respectively. We would then recognize that computation of the coupling matrices $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$ might pose a significant challenge and therefore alternative approaches should be explored.

Iterative solution of Eq. (89) can be written as

$$\begin{aligned} \mathbf{P}_{11}\Delta\mathbf{y}_1+\mathbf{P}_{12}\Delta\mathbf{y}_2 &= \mathbf{b}_1-\left(\mathbf{A}_{11}\mathbf{x}_1+\mathbf{A}_{12}\mathbf{x}_2\right), \\ \mathbf{P}_{21}\Delta\mathbf{y}_1+\mathbf{P}_{22}\Delta\mathbf{y}_2 &= \mathbf{b}_2-\left(\mathbf{A}_{21}\mathbf{x}_1+\mathbf{A}_{22}\mathbf{x}_2\right), \end{aligned} \tag{91}$$

where $\mathbf{P}_{\beta\gamma}$'s represent the blocks of the preconditioning matrix $\mathbf{P}$. Here we focus our attention to computation of the residual vectors on the right hand side, and explore alternative ways for evaluating the matrix–vector products.

Let us suppose that we are able to compute, without a major difficulty, the element-level matrices $\mathbf{A}_{11}^e$ and $\mathbf{A}_{22}^e$ associated with the global matrices $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$, and that we prefer to evaluate $\mathbf{A}_{11}\mathbf{x}_1$ and $\mathbf{A}_{22}\mathbf{x}_2$ by using these element-level matrices. Let us also suppose that calculation of the element-level matrices $\mathbf{A}_{12}^e$ and $\mathbf{A}_{21}^e$ is exceedingly difficult. Reflecting these circumstances, the computations can be carried out by using a mixed element-matrix-based/element-vector-based computation technique [2], [6]:

$$\begin{aligned} \left(\mathbf{A}_{11}\mathbf{x}_1+\mathbf{A}_{12}\mathbf{x}_2\right) &= \mathop{\mathrm{A}}_{e=1}^{n_{el}}\left(\mathbf{A}_{11}^e\mathbf{x}_1\right)+\mathop{\mathrm{A}}_{e=1}^{n_{el}}\lim_{\epsilon_1\to 0}\left[\frac{\mathbf{N}_1^e(\mathbf{d}_1,\mathbf{d}_2+\epsilon_1\mathbf{x}_2)-\mathbf{N}_1^e(\mathbf{d}_1,\mathbf{d}_2)}{\epsilon_1}\right], \\ \left(\mathbf{A}_{21}\mathbf{x}_1+\mathbf{A}_{22}\mathbf{x}_2\right) &= \mathop{\mathrm{A}}_{e=1}^{n_{el}}\left(\mathbf{A}_{22}^e\mathbf{x}_2\right)+\mathop{\mathrm{A}}_{e=1}^{n_{el}}\lim_{\epsilon_2\to 0}\left[\frac{\mathbf{N}_2^e(\mathbf{d}_1+\epsilon_2\mathbf{x}_1,\mathbf{d}_2)-\mathbf{N}_2^e(\mathbf{d}_1,\mathbf{d}_2)}{\epsilon_2}\right], \end{aligned} \tag{92}$$

where ϵ_1 and ϵ_2 are small parameters used in numerical evaluation of the directional derivatives. Here, $\mathbf{A}_{11}\mathbf{x}_1$ and $\mathbf{A}_{22}\mathbf{x}_2$ are evaluated with an element-matrix-based computation technique, while $\mathbf{A}_{12}\mathbf{x}_2$ and $\mathbf{A}_{21}\mathbf{x}_1$ are evaluated with an element-vector-based computation technique.

In extending the mixed element-matrix-based/element-vector-based computation technique described above to a more general framework, evaluation of a matrix–vector product $\mathbf{A}_{\beta\gamma}\mathbf{x}_\gamma$ (for

$\beta, \gamma = 1, 2, \ldots, N$ and no sum) appearing in a residual vector can be formulated as an intentional choice between the following element-matrix-based and element-vector-based computation techniques:

$$\mathbf{A}_{\beta\gamma}\mathbf{x}_\gamma = \mathop{\mathrm{A}}_{e=1}^{n_{el}} (\mathbf{A}^e_{\beta\gamma}\mathbf{x}_\gamma), \tag{93}$$

$$\mathbf{A}_{\beta\gamma}\mathbf{x}_\gamma = \mathop{\mathrm{A}}_{e=1}^{n_{el}} \lim_{\epsilon_\beta \to 0} \left[\frac{\mathbf{N}^e_\beta(\ldots, \mathbf{d}_\gamma + \epsilon_\beta \mathbf{x}_\gamma, \ldots) - \mathbf{N}^e_\beta(\ldots, \mathbf{d}_\gamma, \ldots)}{\epsilon_\beta} \right]. \tag{94}$$

Sometimes, computation of the element-level matrices $\mathbf{A}^e_{\beta\gamma}$ might not be prohibitively difficult, but we might still prefer to evaluate $\mathbf{A}_{\beta\gamma}\mathbf{x}_\gamma$ with an element-vector-based computation technique. In such cases, instead of an element-vector-based computation technique requiring numerical evaluation of directional derivatives, we might want to use the element-vector-based computation technique described below.

Let us suppose that the nonlinear vector function $\mathbf{N}_\beta$ corresponds to a finite element integral form $\mathbf{B}_\beta(\mathbf{W}_\beta, \mathbf{u}_1, \ldots, \mathbf{u}_N)$. Here $\mathbf{W}_\beta$ represents the vector of nodal values associated with the weighting function $\mathbf{w}_\beta$, which generates the nonlinear equation block β. Let us also suppose that we are able to, without a major difficulty, derive the first-order terms in the expansion of $\mathbf{B}_\beta(\mathbf{W}_\beta, \mathbf{u}_1, \ldots, \mathbf{u}_N)$ in $\mathbf{u}_\gamma$. Let the finite element integral form $\mathbf{G}_{\beta\gamma}(\mathbf{W}_\beta, \mathbf{u}_1, \ldots, \mathbf{u}_N, \Delta\mathbf{u}_\gamma)$ represents those first-order terms in $\Delta\mathbf{u}_\gamma$. We note that this finite element integral form will generate $\frac{\partial \mathbf{N}_\beta}{\partial \mathbf{d}_\gamma}$. Consequently, the matrix-vector product $\mathbf{A}_{\beta\gamma}\mathbf{x}_\gamma$ can be evaluated as described in [2], [6]:

$$\mathbf{A}_{\beta\gamma}\mathbf{x}_\gamma = \frac{\partial \mathbf{N}_\beta}{\partial \mathbf{d}_\gamma}\mathbf{x}_\gamma \quad = \quad \mathop{\mathrm{A}}_{e=1}^{n_{el}} \mathbf{G}_{\beta\gamma}(\mathbf{W}_\beta, \mathbf{u}_1, \ldots, \mathbf{u}_N, \mathbf{v}_\gamma), \tag{95}$$

where, $\mathbf{v}_\gamma$ is a function interpolated from $\mathbf{x}_\gamma$ in the same way that $\mathbf{u}_\gamma$ is interpolated from $\mathbf{d}_\gamma$. This element-vector-based computation technique allows us to evaluate matrix–vector products without dealing with numerical evaluation of directional derivatives. To differentiate between the element-vector-based computation techniques defined by Eqs. (94) and (95), we will call them, respectively, numerical element-vector-based (NEVB) and analytical element-vector-based (AEVB) computation techniques.

Remark 6: In fluid–structure interaction computations where the structure is light, in the absence of taking into account the coupling blocks $\mathbf{A}_{12}$ and $\mathbf{A}_{21}$, we propose a short cut approach for improving the convergence of the coupling iterations. In this approach, to reduce "over-correcting" (i.e. "over-incrementing") the structural displacements during the coupling iterations, we artificially increase the structural mass contribution to the matrix block corresponding to the structural mechanics equations and unknowns. In the context of this section, with the understanding that subscript 2 denotes the structure, this would be equivalent to artificially increasing the mass matrix contribution to $\mathbf{A}_{22}$. This is achieved without altering $\mathbf{b}_1$ or $\mathbf{b}_2$ (i.e. $\mathbf{F}_1 - \mathbf{N}_1\ (\mathbf{d}_1, \mathbf{d}_2)$ or $\mathbf{F}_2 - \mathbf{N}_2\ (\mathbf{d}_1, \mathbf{d}_2)$), and therefore when the coupling iterations converge, they converge to the solution of the problem with the correct structural mass.

Remark 7: In fluid–structure interaction computations with light and thin structures (such as membranes), it might be desirable to eliminate the higher spatial modes of the structural response normal to the membrane. We propose to accomplish that by adding to the finite element formulation of the structural mechanics problem a "Directional-Inertia Stabilizing Mass (DISM)" term, which we define as

$$S_{\text{DISM}} = \sum_{e=1}^{n_{el}} \int_{(\Omega^s)^e} \mathbf{w} \cdot (\eta\ \rho^s\ \mathbf{nn}) \cdot \left(\frac{d^2 \mathbf{y}^h}{dt^2} \right) d\Omega^s\ , \tag{96}$$

where Ω^s is the membrane domain, $\mathbf{y}^h$ is the displacement, ρ^s is the material density, $\mathbf{n}$ is the unit vector normal to the membrane, and η is a non-dimensional measure of the curvature in $\left(\frac{d^2\mathbf{y}^h}{dt^2}\right)$. As a possible alternative to the DISM term, we propose a "Directional-Damping Stabilization (DDS)" term defined as

$$S_{\text{DDS}} = \sum_{e=1}^{n_{el}} \int_{(\Omega^s)^e} \mathbf{w} \cdot \left(\xi^h \ \omega_{\text{int}}^h \ \rho^s \ \mathbf{nn}\right) \cdot \left(\frac{d\mathbf{y}^h}{dt}\right) d\Omega^s \ , \tag{97}$$

where ω_{int}^h is an intrinsic frequency and ξ^h is a non-dimensional measure of the curvature in $\left(\frac{d\mathbf{y}^h}{dt}\right)$. While we view these admittedly ad hoc techniques as short cut stabilization approaches, we also propose the "Fluid–Structure Interactions Mixed Structural Modeling (FSIMSM)" as a more rigorous approach. In FSIMSM, a mixture of different models (such as membrane, shell and continuum elements) would be used for representing the structure, depending on the nature of its deformation modes. For example, parachute computations would normally be based on using membrane and cable elements to model the parachute structure. In the FSIMSM approach, in regions of the structure where wrinkling or other instabilities are experienced or expected, the model would convert to one that is based on using shell and beam elements. This would bring bending rigidity to where it is needed. Appropriate interface (matching) conditions would be used where two different models meet. The FSIMSM can be implemented in a static or dynamic way. In the static way, the model to be used for each structural element would be determined based on what we know about the FSI problem in advance. In the dynamic way, regions experiencing instabilities during the computations would convert to models based on shell and beam elements. Regional models would not be re-defined every time step. They would be re-defined frequently enough to have a safe coverage of the regions experiencing instabilities.

XIV. ENHANCED-DISCRETIZATION SUCCESSIVE UPDATE METHOD (EDSUM)

In this section, we describe a multi-level iteration method for computation of flow behavior at small scales. The Enhanced-Discretization Successive Update Method (EDSUM) [2], [6] is based on the Enhanced-Discretization Interface-Capturing Technique (EDICT). Although it might be possible to identify zones where the enhanced discretization could be limited to, we need to think about and develop methods required for cases where the enhanced discretization is needed everywhere in the problem domain to accurately compute flows at smaller scales. In that case the enhanced discretization would be more wide-spread than before, and possibly required for the entire domain. Therefore an efficient solution approach would be needed to solve, at every time step, a very large, coupled nonlinear equation system generated by the multi-level discretization approach.

Such large, coupled nonlinear equation systems involve four classes of nodes. Class-1 consists of all the Mesh-1 nodes. These nodes are connected to each other through the Mesh-1 elements. Class-2E consists of the Mesh-2 edge nodes (but excluding those coinciding with the Mesh-1 nodes). The edge nodes associated with different edges are not connected (except those at each side of an edge, but we could possibly neglect that a side node might be connected to the side nodes of the adjacent edges). Nodes within an edge are connected through Mesh-2 elements. Class-2F contains the Mesh-2 face nodes (but excluding those on the edges). The face nodes associated with different faces are not connected (except those at sides of a face, but we could possibly neglect that those side nodes might be connected to the side nodes of the adjacent face). Nodes within a face are connected through Mesh-2 elements. Class-2I nodes are the Mesh-2 interior nodes. The interior nodes associated with different clusters of Mesh-2 elements are not connected. Nodes within a cluster are connected through Mesh-2 elements.

Based on this multi-level decomposition concept, a nonlinear equation system of the kind given by Eq. (83) can be re-written as follows:

$$\begin{aligned}
\mathbf{N}_1\left(\mathbf{d}_1,\ \mathbf{d}_{2E},\ \mathbf{d}_{2F},\ \mathbf{d}_{2I}\right) &= \mathbf{F}_1, \\
\mathbf{N}_{2E}\left(\mathbf{d}_1,\ \mathbf{d}_{2E},\ \mathbf{d}_{2F},\ \mathbf{d}_{2I}\right) &= \mathbf{F}_{2E}, \\
\mathbf{N}_{2F}\left(\mathbf{d}_1,\ \mathbf{d}_{2E},\ \mathbf{d}_{2F},\ \mathbf{d}_{2I}\right) &= \mathbf{F}_{2F}, \\
\mathbf{N}_{2I}\left(\mathbf{d}_1,\ \mathbf{d}_{2E},\ \mathbf{d}_{2F},\ \mathbf{d}_{2I}\right) &= \mathbf{F}_{2I},
\end{aligned} \tag{98}$$

where the subscript "$n+1$" has been dropped to simplify the notation.

This equation system would be solved with an approximate Newton–Raphson method. At each nonlinear iteration step, we would successively update the solution vectors corresponding to each class. While updating each class, we would use the most recent values of the solution vectors in calculating the vectors $\mathbf{N}_1$, $\mathbf{N}_{2E}$, $\mathbf{N}_{2F}$, and $\mathbf{N}_{2I}$ and their derivatives with respect to the solution vectors. We would start with updating the Class-1 nodes, then update the Class-2E, Class-2F, and Class-2I nodes, respectively. The process is shown below, where each class of equations are solved in the order they are written.

$$\begin{aligned}
\left.\frac{\partial \mathbf{N}_1}{\partial \mathbf{d}_1}\right|_{\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right)} \left(\Delta \mathbf{d}_1^i\right) &= \mathbf{F}_1 - \mathbf{N}_1\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right), \\
\left.\frac{\partial \mathbf{N}_{2E}}{\partial \mathbf{d}_{2E}}\right|_{\left(\mathbf{d}_1^{i+1},\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right)} \left(\Delta \mathbf{d}_{2E}^i\right) &= \mathbf{F}_{2E} - \mathbf{N}_{2E}\left(\mathbf{d}_1^{i+1},\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right), \\
\left.\frac{\partial \mathbf{N}_{2F}}{\partial \mathbf{d}_{2F}}\right|_{\left(\mathbf{d}_1^{i+1},\ \mathbf{d}_{2E}^{i+1},\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right)} \left(\Delta \mathbf{d}_{2F}^i\right) &= \mathbf{F}_{2F} - \mathbf{N}_{2F}\left(\mathbf{d}_1^{i+1},\ \mathbf{d}_{2E}^{i+1},\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right), \\
\left.\frac{\partial \mathbf{N}_{2I}}{\partial \mathbf{d}_{2I}}\right|_{\left(\mathbf{d}_1^{i+1},\ \mathbf{d}_{2E}^{i+1},\ \mathbf{d}_{2F}^{i+1},\ \mathbf{d}_{2I}^i\right)} \left(\Delta \mathbf{d}_{2I}^i\right) &= \mathbf{F}_{2I} - \mathbf{N}_{2I}\left(\mathbf{d}_1^{i+1},\ \mathbf{d}_{2E}^{i+1},\ \mathbf{d}_{2F}^{i+1},\ \mathbf{d}_{2I}^i\right).
\end{aligned} \tag{99}$$

This sequence would be repeated as many times as needed, and, as an option, we could alternate between this sequence and its reverse sequence:

$$\begin{aligned}
\left.\frac{\partial \mathbf{N}_{2I}}{\partial \mathbf{d}_{2I}}\right|_{\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right)} \left(\Delta \mathbf{d}_{2I}^i\right) &= \mathbf{F}_{2I} - \mathbf{N}_{2I}\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^i\right), \\
\left.\frac{\partial \mathbf{N}_{2F}}{\partial \mathbf{d}_{2F}}\right|_{\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^{i+1}\right)} \left(\Delta \mathbf{d}_{2F}^i\right) &= \mathbf{F}_{2F} - \mathbf{N}_{2F}\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^i,\ \mathbf{d}_{2I}^{i+1}\right), \\
\left.\frac{\partial \mathbf{N}_{2E}}{\partial \mathbf{d}_{2E}}\right|_{\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^{i+1},\ \mathbf{d}_{2I}^{i+1}\right)} \left(\Delta \mathbf{d}_{2E}^i\right) &= \mathbf{F}_{2E} - \mathbf{N}_{2E}\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^i,\ \mathbf{d}_{2F}^{i+1},\ \mathbf{d}_{2I}^{i+1}\right), \\
\left.\frac{\partial \mathbf{N}_1}{\partial \mathbf{d}_1}\right|_{\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^{i+1},\ \mathbf{d}_{2F}^{i+1},\ \mathbf{d}_{2I}^{i+1}\right)} \left(\Delta \mathbf{d}_1^i\right) &= \mathbf{F}_1 - \mathbf{N}_1\left(\mathbf{d}_1^i,\ \mathbf{d}_{2E}^{i+1},\ \mathbf{d}_{2F}^{i+1},\ \mathbf{d}_{2I}^{i+1}\right).
\end{aligned} \tag{100}$$

Updating the solution vector corresponding to each class would also require solution of a large equation system. These equations systems would each be solved iteratively, with an effective preconditioner, a reliable search technique, and parallel implementation. It is important to note that the bulk of the computational cost would be for Class-1 and Class-2I. While the Class-1 nodes would be

partitioned to different processors of the parallel computer, for the remaining classes, nodes in each edge, face or interior cluster would be assigned to the same processor. Therefore, solution of each edge, face or interior cluster would be local. If the size of each interior cluster becomes too large, then nodes for a given cluster can also be distributed across different processors, or a third level of mesh refinement can be introduced to make the enhanced discretization a tri-level kind.

A variation of the EDSUM could be used for the iterative solution of the linear equation system that needs to be solved at every step of a (full) Newton–Raphson method applied to Eq. (98). To describe this variation, we first write, similar to the way we wrote Eqs. (89) and (90), the linear equation system that needs to be solved:

$$\begin{aligned}
\mathbf{A}_{11}\mathbf{x}_1 + \mathbf{A}_{12E}\mathbf{x}_{2E} + \mathbf{A}_{12F}\mathbf{x}_{2F} + \mathbf{A}_{12I}\mathbf{x}_{2I} &= \mathbf{b}_1, \\
\mathbf{A}_{2E1}\mathbf{x}_1 + \mathbf{A}_{2E2E}\mathbf{x}_{2E} + \mathbf{A}_{2E2F}\mathbf{x}_{2F} + \mathbf{A}_{2E2I}\mathbf{x}_{2I} &= \mathbf{b}_{2E}, \\
\mathbf{A}_{2F1}\mathbf{x}_1 + \mathbf{A}_{2F2E}\mathbf{x}_{2E} + \mathbf{A}_{2F2F}\mathbf{x}_{2F} + \mathbf{A}_{2F2I}\mathbf{x}_{2I} &= \mathbf{b}_{2F}, \\
\mathbf{A}_{2I1}\mathbf{x}_1 + \mathbf{A}_{2I2E}\mathbf{x}_{2E} + \mathbf{A}_{2I2F}\mathbf{x}_{2F} + \mathbf{A}_{2I2I}\mathbf{x}_{2I} &= \mathbf{b}_{2I},
\end{aligned} \tag{101}$$

where

$$\mathbf{A}_{\beta\gamma} = \frac{\partial \mathbf{N}_\beta}{\partial \mathbf{d}_\gamma}, \tag{102}$$

with $\beta, \gamma = 1, 2E, 2F, 2I$. Then, for the iterative solution of Eq. (101), we define two preconditioners:

$$\mathbf{P}_{LTOS} = \begin{bmatrix} \mathbf{A}_{11} & 0 & 0 & 0 \\ \mathbf{A}_{2E1} & \mathbf{A}_{2E2E} & 0 & 0 \\ \mathbf{A}_{2F1} & \mathbf{A}_{2F2E} & \mathbf{A}_{2F2F} & 0 \\ \mathbf{A}_{2I1} & \mathbf{A}_{2I2E} & \mathbf{A}_{2I2F} & \mathbf{A}_{2I2I} \end{bmatrix}, \tag{103}$$

$$\mathbf{P}_{STOL} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12E} & \mathbf{A}_{12F} & \mathbf{A}_{12I} \\ 0 & \mathbf{A}_{2E2E} & \mathbf{A}_{2E2F} & \mathbf{A}_{2E2I} \\ 0 & 0 & \mathbf{A}_{2F2F} & \mathbf{A}_{2F2I} \\ 0 & 0 & 0 & \mathbf{A}_{2I2I} \end{bmatrix}. \tag{104}$$

We propose that these two preconditioners are used alternatingly during the inner iterations of the GMRES search. We note that this mixed preconditioning technique with multi-level discretization is closely related to the mixed CEBE and CC preconditioning technique [29] we referred to in Section XII. Along these lines, as a mixed preconditioning technique that is more closely related to the mixed CEBE and CC technique, we propose that the following three preconditioners are used in sequence during the inner iterations of the GMRES search:

$$\mathbf{P}_L = \begin{bmatrix} \mathbf{A}_{11} & 0 & 0 & 0 \\ 0 & \mathrm{DIAG}\,(\mathbf{A}_{2E2E}) & 0 & 0 \\ 0 & 0 & \mathrm{DIAG}\,(\mathbf{A}_{2F2F}) & 0 \\ 0 & 0 & 0 & \mathrm{DIAG}\,(\mathbf{A}_{2I2I}) \end{bmatrix}, \tag{105}$$

$$\mathbf{P}_{SETOI} = \begin{bmatrix} \text{DIAG}\,(\mathbf{A}_{11}) & 0 & 0 & 0 \\ 0 & \mathbf{A}_{2E2E} & 0 & 0 \\ 0 & \mathbf{A}_{2F2E} & \mathbf{A}_{2F2F} & 0 \\ 0 & \mathbf{A}_{2I2E} & \mathbf{A}_{2I2F} & \mathbf{A}_{2I2I} \end{bmatrix}, \tag{106}$$

$$\mathbf{P}_{SITOE} = \begin{bmatrix} \text{DIAG}\,(\mathbf{A}_{11}) & 0 & 0 & 0 \\ 0 & \mathbf{A}_{2E2E} & \mathbf{A}_{2E2F} & \mathbf{A}_{2E2I} \\ 0 & 0 & \mathbf{A}_{2F2F} & \mathbf{A}_{2F2I} \\ 0 & 0 & 0 & \mathbf{A}_{2I2I} \end{bmatrix}. \tag{107}$$

As possible sequences, we propose ($\mathbf{P}_L$, $\mathbf{P}_{SETOI}$, $\mathbf{P}_{SITOE}$, ..., $\mathbf{P}_L$, $\mathbf{P}_{SETOI}$, $\mathbf{P}_{SITOE}$), as well as ($\mathbf{P}_L$, $\mathbf{P}_{SETOI}$, ..., $\mathbf{P}_L$, $\mathbf{P}_{SETOI}$) and ($\mathbf{P}_L$, $\mathbf{P}_{SITOE}$, ..., $\mathbf{P}_L$, $\mathbf{P}_{SITOE}$). As a somewhat downgraded version of $\mathbf{P}_L$, we can use a preconditioner that is equivalent to not updating $\mathbf{x}_{2E}$, $\mathbf{x}_{2F}$, and $\mathbf{x}_{2I}$, instead of updating them by using DIAG ($\mathbf{A}_{2E2E}$), DIAG ($\mathbf{A}_{2F2F}$), and DIAG ($\mathbf{A}_{2I2I}$). Similarly, as downgraded versions of $\mathbf{P}_{SETOI}$ and $\mathbf{P}_{SITOE}$, we can use preconditioners that are equivalent to not updating $\mathbf{x}_1$, instead of updating it by using DIAG ($\mathbf{A}_{11}$).

To differentiate between the two variations of the EDSUM we described in this section, we call the nonlinear version, described by Eqs. (99) and (100), EDSUM-N, and the linear version, described by Eqs. (101) – (107), EDSUM-L.

Remark 8: The EDSUM provides a natural framework for resourceful and selective application of stabilized formulations to multi-scale computations and subgrid-scale modeling. Along these lines we propose the "Enhanced-Discretization Selective Stabilization Procedure (EDSSP)". In the EDSSP, finite element equations generating different blocks of the nonlinear equation system given by Eq. (98) would be based on different stabilized formulations. Level-1 equations (generating the first block) would be based on a stabilized formulation more suitable for flow behavior at larger scales, and the Level-2 equations (generating the second, third and fourth blocks) would be based on a stabilized formulation more suitable for flow behavior at smaller scales. As a special version of the EDSSP, we propose to use with the Level-1 equations only the SUPG and PSPG stabilizations, and with the Level-2 equations use additionally the DCDD stabilization.

Remark 9: We propose the EDSUM-B as an approximate version of the EDSUM. In the EDSUM-B, we propose to solve the Level-2 equations less frequently than the Level-1 equations. For example, instead of performing the same number of iterations to solve all four blocks of Eq. (98), we can perform one iteration for the Level-2 blocks for every two iterations we perform for the Level-1 block. This would mean that the small-scale data used in solving the large-scale equations is updated at every other iteration. As another example of EDSUM-B, we can use for the Level-2 equations a more dissipative time-integration algorithm and a time-step size that is twice the time-step size we use for the Level-1 equations. This would mean that the small-scale data used in solving the large-scale equations is updated at every other time step. Although the EDSUM-B is a more approximate technique compared to the EDSUM, it will still be superior in accuracy compared to carrying out the computation with the Level-1 discretization alone.

XV. Concluding Remarks

We highlighted the stabilized finite element interface-tracking and interface-capturing techniques we developed in recent years for computation of flows with moving boundaries and interfaces. We also highlighted the calculation of the stabilization parameters used in these stabilized formulations. The interface-tracking techniques are based on the DSD/SST formulation, where the mesh moves to track the interface. The interface-capturing techniques, which were developed for two-fluid flows, are based on the stabilized formulation, over non-moving meshes, of both the flow equations and

an advection equation. The advection equation governs the time-evolution of the interface function marking the interface location. We also described some of the additional methods developed to increase the scope and accuracy of the interface-tracking and interface-capturing techniques.

Acknowledgments

This work was supported by the US Army Natick Soldier Center and NASA JSC.

References

[1] T.E. Tezduyar, "Stabilized finite element formulations for incompressible flow computations," *Advances in Applied Mechanics*, vol. 28, pp. 1–44, 1992.

[2] T.E. Tezduyar, "Finite element methods for flow problems with moving boundaries and interfaces," *Archives of Computational Methods in Engineering*, vol. 8, pp. 83–130, 2001.

[3] T.E. Tezduyar, "Interface-tracking and interface-capturing techniques for computation of two-fluid flows," in *Proceedings of the First MIT Conference on Computational Fluid and Solid Mechanics*, Boston, Massachusetts, 2001.

[4] T. Tezduyar, "Finite element interface-tracking and interface-capturing techniques for flows with moving boundaries and interfaces," in *Proceedings of the ASME Symposium on Fluid-Physics and Heat Transfer for Macro- and Micro-Scale Gas-Liquid and Phase-Change Flows (CD-ROM)*, New York, New York, 2001, ASME Paper IMECE2001/HTD-24206, ASME.

[5] T. Tezduyar, "Interface-tracking and interface-capturing techniques for computation of moving boundaries and interfaces," in *Proceedings of the Fifth World Congress on Computational Mechanics*, Vienna, Austria, 2002, On-line publication: http://wccm.tuwien.ac.at/, Paper-ID: 81513.

[6] T.E. Tezduyar, "Interface-tracking, interface-capturing and enhanced solution techniques," in *Proceedings of the First South-American Congress on Computational Mechanics (CD-ROM)*, Santa Fe–Parana, Argentina, 2002.

[7] T.J.R. Hughes and A.N. Brooks, "A multi-dimensional upwind scheme with no crosswind diffusion," in *Finite Element Methods for Convection Dominated Flows*, T.J.R. Hughes, Ed., AMD-Vol.34, pp. 19–35. ASME, New York, 1979.

[8] T.E. Tezduyar and T.J.R. Hughes, "Development of time-accurate finite element techniques for first-order hyperbolic systems with particular emphasis on the compressible Euler equations," NASA Technical Report NASA-CR-204772, NASA, 1982, http://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19970023187_1997034954.pdf.

[9] T.E. Tezduyar and T.J.R. Hughes, "Finite element formulations for convection dominated flows with particular emphasis on the compressible Euler equations," in *Proceedings of AIAA 21st Aerospace Sciences Meeting*, Reno, Nevada, 1983, AIAA Paper 83-0125.

[10] T.J.R. Hughes, L.P. Franca, and M. Balestra, "A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška–Brezzi condition: A stable Petrov–Galerkin formulation of the Stokes problem accommodating equal-order interpolations," *Computer Methods in Applied Mechanics and Engineering*, vol. 59, pp. 85–99, 1986.

[11] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, and S. Mittal, "Parallel finite-element computation of 3D flows," *Computer*, vol. 26, no. 10, pp. 27–36, 1993.

[12] T.E. Tezduyar and Y.J. Park, "Discontinuity capturing finite element formulations for nonlinear convection-diffusion-reaction equations," *Computer Methods in Applied Mechanics and Engineering*, vol. 59, pp. 307–325, 1986.

[13] T.E. Tezduyar and D.K. Ganjoo, "Petrov-Galerkin formulations with weighting functions dependent upon spatial and temporal discretization: Applications to transient convection-diffusion problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 59, pp. 49–71, 1986.

[14] L.P. Franca, S.L. Frey, and T.J.R. Hughes, "Stabilized finite element methods: I. Application to the advective-diffusive model," *Computer Methods in Applied Mechanics and Engineering*, vol. 95, pp. 253–276, 1992.

[15] T.E. Tezduyar and Y. Osawa, "Finite element stabilization parameters computed from element matrices and vectors," *Computer Methods in Applied Mechanics and Engineering*, vol. 190, pp. 411–430, 2000.

[16] T.E. Tezduyar, "Adaptive determination of the finite element stabilization parameters," in *Proceedings of the ECCOMAS Computational Fluid Dynamics Conference 2001 (CD-ROM)*, Swansea, Wales, United Kingdom, 2001.

[17] T. Tezduyar, "Stabilization parameters and local length scales in SUPG and PSPG formulations," in *Proceedings of the Fifth World Congress on Computational Mechanics*, Vienna, Austria, 2002, On-line publication: http://wccm.tuwien.ac.at/, Paper-ID: 81508.

[18] T.E. Tezduyar, "Calculation of the stabilization parameters in SUPG and PSPG formulations," in *Proceedings of the First South-American Congress on Computational Mechanics (CD-ROM)*, Santa Fe–Parana, Argentina, 2002.

[19] T.J.R. Hughes and G.M. Hulbert, "Space–time finite element methods for elastodynamics: formulations and error estimates," *Computer Methods in Applied Mechanics and Engineering*, vol. 66, pp. 339–363, 1988.

[20] T. Tezduyar, S. Aliabadi, and M. Behr, "Enhanced-Discretization Interface-Capturing Technique," in *ISAC '97 High Performance Computing on Multiphase Flows*, Y. Matsumoto and A. Prosperetti, Eds., pp. 1–6. Japan Society of Mechanical Engineers, 1997.

[21] T. Tezduyar, S. Aliabadi, and M. Behr, "Enhanced-Discretization Interface-Capturing Technique (EDICT) for computation of unsteady flows with interfaces," *Computer Methods in Applied Mechanics and Engineering*, vol. 155, pp. 235–248, 1998.

[22] T.E. Tezduyar, M. Behr, S. Mittal, and A.A. Johnson, "Computation of unsteady incompressible flows with the finite element methods – space–time formulations, iterative strategies and massively parallel implementations," in *New Methods in Transient Analysis*, New York, 1992, PVP-Vol.246/AMD-Vol.143, pp. 7–24, ASME.

[23] D.R. Lynch, "Wakes in liquid-liquid systems," *Journal of Computational Physics*, vol. 47, pp. 387–411, 1982.

[24] A. Masud and T.J.R. Hughes, "A space–time Galerkin/least-squares finite element formulation of the Navier-Stokes equations for moving domain problems," *Computer Methods in Applied Mechanics and Engineering*, vol. 146, pp. 91–126, 1997.

[25] K. Stein, T. Tezduyar, and R. Benney, "Mesh moving techniques for fluid–structure interactions with large displacements," *Journal of Applied Mechanics*, vol. 70, pp. 58–63, 2003.

[26] K. Stein and T. Tezduyar, "Advanced mesh update techniques for problems involving large displacements," in *Proceedings of the Fifth World Congress on Computational Mechanics*, Vienna, Austria, 2002, On-line publication: http://wccm.tuwien.ac.at/, Paper-ID: 81489.

[27] C. W. Hirt and B. D. Nichols, "Volume of fluid (VOF) method for the dynamics of free boundaries," *Journal of Computational Physics*, vol. 39, pp. 201–225, 1981.

[28] Y. Saad and M. Schultz, "GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems," *SIAM Journal of Scientific and Statistical Computing*, vol. 7, pp. 856–869, 1986.

[29] T.E. Tezduyar, M. Behr, S.K. Aliabadi, S. Mittal, and S.E. Ray, "A new mixed preconditioning method for finite element computations," *Computer Methods in Applied Mechanics and Engineering*, vol. 99, pp. 27–42, 1992.

[30] T. Tezduyar, S. Aliabadi, M. Behr, A. Johnson, V. Kalro, and M. Litke, "High performance computing techniques for flow simulations," in *Solving Large-Scale Problems in Mechanics:*

Parallel Solution Methods in Computational Mechanics, M. Papadrakakis, Ed., chapter 10, pp. 363–398. John Wiley & Sons, 1997.