Numerical Studies of Time Dependent Ginzburg-Landau Model by FEM with Moving Grid Deformation

Mingchao Cai * Stefan Turek †

Abstract

The time dependent Ginzburg-Landau (TDGL) equation is a typical model in phase field theory for many applications like two phase flow simulations and phase transitions. In this paper, we develop effective algorithms so that the solution of the TDGL model can be accurately approximated. Specifically, we adopt finite element methods for the spatial discretization and study different algorithms for handling the nonlinear and diffusion terms in the TDGL model. We show that the fully implicit Backward Euler scheme and the Crank-Nicolson scheme exhibit high accuracy and allow for large time step size. Since high resolution is needed in the interfacial region when the interfacial thickness is small, we apply newly developed moving grid deformation techniques [10] to improve both the accuracy and the efficiency. The advantages of the applied moving grid deformation algorithm over the existing works are highlighted.

Keywords: time dependent Ginzburg-Landau model, Allen-Cahn equation, phase field, moving grid deformation, multigrid solver, mean curvature, finite element.

1 Introduction.

Scientific works on phase field methods arouse increasing interest recently [4, 5, 7, 13, 15]. Many physical processes like microstructure evolution in material science [4], alloy melting and solidification, grain growth [4] and binary fluids dynamics [15], can be described by using phase field theory. In phase field models, an order parameter $\phi(\mathbf{x}, t) \in [-1, 1]$ is introduced to characterize different phases. Typically, a point $\mathbf{x}$ in the domain Ω belongs to one phase if $\phi(\mathbf{x}, t) = 1$ and it is in another phase if $\phi(\mathbf{x}, t) = -1$, the region where $\phi(\mathbf{x}, t)$ changes rapidly from 1 to -1 is the interfacial region. Generally speaking, phase field theory emphasizes on modeling the total free energy in terms of ϕ, and the governing equation for ϕ then follows from the minimization of the free energy. In this paper, we consider the following form of the total free energy

$$\mathcal{F} = \int_\Omega \left(\frac{1}{2} |\nabla \phi|^2 + F(\phi) \right) d\Omega.$$

Here, $F(\phi) = \frac{(\phi^2-1)^2}{4\epsilon^2}$ is the the double well potential with ϵ being the interfacial thickness. The time dependent Ginzburg-Landau model is obtained by the following energy minimization argument [2, 13]:

$$\frac{\partial \phi}{\partial t} = -\frac{\delta \mathcal{F}}{\delta \phi} = -\left(-\Delta \phi + \frac{1}{\epsilon^2} f(\phi) \right), \quad \forall \mathbf{x} \in \Omega,\ t \in [0, T], \tag{1}$$

*Institute of Applied Mathematics, Dortmund University of Technology, Vogelpothsweg 87, D-44227, Dortmund, Germany, cmchao2005@gmail.com. Work supported in part by DFG through the grant PAK178(Ku1530/5).

†Institute of Applied Mathematics, Dortmund University of Technology, Vogelpothsweg 87, D-44227, Dortmund, Germany, ture@featflow.de. Work supported in part by DFG through the grant SFB708.

where $\frac{1}{\epsilon^2} f(\phi) = \frac{1}{\epsilon^2}(\phi^3 - \phi)$ is the derivative of $F(\phi)$. Moreover, ϕ satisfies the Neumann boundary condition

$$\nabla\phi \cdot \mathbf{n} = 0, \quad \forall \mathbf{x} \in \partial\Omega, t \in [0, T].$$

The TDGL model and its derivative forms are fundamental in phase field theory [5, 7, 15]. For instance, if a convective term is added to the left hand side of (1), then it leads to the following Allen-Cahn equation:

$$\frac{\partial \phi}{\partial t} + \mathbf{u} \cdot \nabla\phi = -\left(-\Delta\phi + \frac{1}{\epsilon^2} f(\phi)\right). \tag{2}$$

Equation (2) has been used in many real applications like two phase flow dynamics and solidification processes for binary alloys [5, 13, 15]. In addition, both (1) and (2) are highly connected with geometric problems of moving surface (curves in two dimensional case). It was formally proved that, as ϵ tends to 0, the zero level set of ϕ (denoted by $\Gamma_t^\epsilon = \{\mathbf{x} \in \Omega | \phi(\mathbf{x}, t) = 0\}$) approaches to a surface Γ_t, which evolves with the normal velocity (denoted by $\mathbf{V}_n$) being equal to its mean curvature (denoted by κ) , i.e.,

$$\mathbf{V}_n = \kappa. \tag{3}$$

In general, it is not easy to solve the diffusive interface model (1) analytically. People usually compare the numerical results with the solution of the sharp interface model (3). However, it is difficult to analyze the numerical error because how to analyze the error from the model approximation, say e_ϵ, and the discretization error, say $e_{\epsilon,\Delta t,h}$, is a very delicate issue [19]. Existing rigorous analysis requires some strong assumptions on the time step size Δt and the mesh size h. For instance, it is proved in [14] that if the $P1$ conforming element is used for spatial discretization and the forward Euler scheme is used for time discretization, the zero level set of the diffusive model (1) converges to the zero level set of the sharp interface model, provided that $\Delta t, h^2 = o(\epsilon^4)$. For backward Euler scheme, the constraint relaxes to $\Delta t, h^2 = o(\epsilon^3)$. The purpose of this paper is to propose and compare different algorithms so that we have more information on the choices of the discretization parameters. Moreover, we use an adapted moving grid algorithm to improve both the accuracy and the efficiency.

From (1), we note that the governing equation consists of a time derivative, a diffusion term and a nonlinear term of polynomial type. Therefore, how to properly handle all these terms is crucial for designing numerical algorithms. It easy to see that the following explicit scheme [5]

$$\frac{\phi(t_{n+1}) - \phi(t_n)}{\Delta t} = -\left(-\Delta\phi(t_n) + \frac{1}{\epsilon^2} f(\phi(t_n))\right) \tag{4}$$

is time consuming because of the stability constraint from the diffusion term. Therefore, it is natural to treat the diffusion term implicitly. In the present work, we search for proper ways for handling the nonlinear term. Specifically, we examine the following 3 ways:

1. Explicitly treating the nonlinear term.
2. Linearizing the nonlinear term by using the solution of the previous time step.
3. Utilizing a fully nonlinear solver.

We try to understand all these algorithms and discuss the choices of the discretization parameters. We compare all these algorithms by testing benchmark problems, for instance, circular bubble shrinking under the mean curvature force. It is observed that the numerical shrinking speed of the bubble based on algorithm 1 and algorithm 2 is highly affected by the discretization parameters like the time step size, the finite element used and the mesh size. Numerical results show that algorithm 3 performs much better and allows for relative

large time step size. Furthermore, we highlight some points on how to choose various parameters based on the numerical experiments.

The obvious difficulty in numerical methods for phase field models is that high resolution is needed in the interfacial region to achieve high accuracy. For equation (1) and equation (2), it is notable that when ϵ is small, if a uniform mesh is used, the mesh has to be very fine. Therefore, an adaptive mesh, which is concentrated around the interfacial region, is a good remedy for dealing with these issues. Moreover, since the governing equation is a time dependent partial differential equation, the mesh should move according to the dynamics of the interface. This calls for moving grids in numerical implementation. In this paper, we adopt the moving grid deformation algorithm developed recently [10] to improve the numerical accuracy. There are two basic types of grid generation methods, one is calculating the new mesh coordinates $\mathbf{x}$ by minimizing a variational form [15, 18], the other is calculating the mesh velocity $\mathbf{V}_{mesh} = \mathbf{x}_t$ by using a Lagrangian like formulation [6, 10, 12]. The moving grid deformation method in this paper belongs to the velocity based methods [6, 12]. Numerical experiments are given to show that the applied moving grid deformation technique can greatly improve the accuracy and efficiency of the numerical algorithms. The applied moving grid algorithm has several advantages: only linear Poisson problems on fixed meshes are needed to be solved, monitor functions can be obtained directly from distance functions or error distributions, mesh tangling can be prevented, and the data structure for the mesh nodes is always the same as for the starting mesh.

The organization of this paper is as follows. In section 2, we present different algorithms for handling the nonlinear term, the finite element discretization, and the linear and nonlinear solvers for the deduced systems. In section 3, the moving grid deformation algorithm is introduced. In section 4, numerical experiments are conducted to compare the different algorithms. Conclusions are drawn in the last section.

2 Numerical Algorithms.

2.1 Time Stepping Schemes.

Since the explicit treatment of the diffusion term leads to stability problems, we will use implicit schemes for the diffusion term. Therefore, we mainly discuss on how to handle the nonlinear term. Note that $f(\phi)$ is a cubic polynomial, one may approximate it by the following ways.

Algorithm 2.1. Solve for $\phi(t_{n+1})$ by implicitly treating the Laplacian term and explicitly treating the nonlinear term:

$$\frac{\phi(t_{n+1}) - \phi(t_n)}{\Delta t} - \theta\Delta\phi(t_{n+1}) = (1-\theta)\Delta\phi(t_n) - \frac{1}{\epsilon^2}f(\phi(t_n)). \tag{5}$$

Here and thereafter $\theta \in [0, 1]$ is an adjustable parameter.

Algorithm 2.2. By using the solution of the previous time step and applying the linear approximation

$$f(\phi(t_{n+1})) = (\phi^2(t_{n+1}) - 1)\phi(t_{n+1}) \approx (\phi^2(t_n) - 1)\phi(t_{n+1}),$$

we have the following scheme:

$$\begin{aligned}\frac{\phi(t_{n+1}) - \phi(t_n)}{\Delta t} + \theta\left[-\Delta\phi(t_{n+1}) + \frac{1}{\epsilon^2}(\phi(t_n)^2 - 1)\phi(t_{n+1})\right] \\ = -(1-\theta)\left[-\Delta\phi(t_n) + \frac{1}{\epsilon^2}f(\phi(t_n))\right].\end{aligned} \tag{6}$$

Algorithm 2.3. Find $\phi(t_{n+1})$ by the following completely implicit and nonlinear solver:

$$\frac{\phi(t_{n+1})-\phi(t_n)}{\Delta t}+\theta\left[-\Delta\phi(t_{n+1})+\frac{1}{\epsilon^2}f(\phi(t_{n+1}))\right]=-(1-\theta)\left[-\Delta\phi(t_n)+\frac{1}{\epsilon^2}f(\phi(t_n))\right]. \quad (7)$$

It is easy to see that when $\theta=0$, (5) and (7) degenerate to the Forward Euler scheme (4); when $\theta=1.0$, (7) is the Backward Euler scheme; when $\theta=0.5$, (7) is the second order accurate Crank-Nicolson scheme. Here and thereafter, θ will be set as 0.5 or 1.0 for discussions and comparisons. We comment here that the parameter θ is included in all algorithms not only because it may improve the numerical accuracy, but also because θ is a flexible parameter and the above one step θ schemes can be naturally extended to fractional θ schemes [1], in particular when one needs to couple the TDGL model or the Allen-Cahn model (2) with the Navier-Stokes equations [1].

Actually, some of the above algorithms have been used as sub-solvers in several applications like moving contact line and two phase flow dynamics. However, it seems that there is no systematic comparison and benchmark test for all these algorithms. We note that the time error can be analyzed based on Taylor expansion [19]. For illustration, the local truncation error of the **Algorithm 2.3** with $\theta=0.5$ is

$$\begin{aligned} error &= \tfrac{\phi(t_{n+1})-\phi(t_n)}{\Delta t}-\tfrac{1}{2}\left(\tfrac{\partial\phi(t_{n+1})}{\partial t}+\tfrac{\partial\phi(t_n)}{\partial t}\right)\\ &= \mathcal{O}\left(\Delta t^2\tfrac{\partial^3\phi}{\partial t^3}\right). \end{aligned}$$

Moreover, it is reasonable to assume that, in the interfacial region [19],

$$\frac{\partial^m\phi}{\partial t^m}\sim\frac{1}{\epsilon^m},\quad m=1,2,3,...$$

Therefore, the leading order of the time error is

$$error=\mathcal{O}\left(\frac{\Delta t^2}{\epsilon^3}\right).$$

In comparison, if $\theta=1.0$ in **Algorithm 2.3**, the leading order of the time error is $\mathcal{O}\left(\frac{\Delta t}{\epsilon^2}\right)$. For **Algorithm 2.1** with $\theta=1.0$, the leading order time error analysis can be obtained by using the same way [19]:

$$\begin{aligned} error &= \tfrac{\partial\phi(t_{n+1})}{\partial t}-\tfrac{\phi(t_{n+1})-\phi(t_n)}{\Delta t}+\tfrac{1}{\epsilon^2}[f(\phi(t_{n+1}))-f(\phi(t_n))]\\ &= \tfrac{\Delta t}{2}\tfrac{\partial^2\phi(t_{n+1})}{\partial t^2}+\tfrac{\Delta t}{\epsilon^2}\tfrac{\partial\phi(t_{n+1})}{\partial t}f'(\phi(t_{n+1}))+\mathcal{O}\left(\Delta t^2\tfrac{\partial^3\phi}{\partial t^3}+\tfrac{\Delta t^2}{\epsilon^2}\tfrac{\partial^2\phi}{\partial t^2}\right)\\ &= \mathcal{O}\left(\tfrac{\Delta t}{\epsilon^3}\right). \end{aligned}$$

This implies that the error from the explicit treatment of the nonlinear term is dominant. Similarly, for **Algorithm 2.1** with $\theta=0.5$, the time error, to leading order, again comes from the nonlinear term $\frac{1}{\epsilon^2}[f(\phi(t_{n+1}))-f(\phi(t_n)))]$. More precisely,

$$\begin{aligned} error &= \tfrac{\phi(t_{n+1})-\phi(t_n)}{\Delta t}-\tfrac{1}{2}\left(\tfrac{\partial\phi(t_{n+1})}{\partial t}+\tfrac{\partial\phi(t_n)}{\partial t}\right)+\tfrac{1}{2\epsilon^2}[f(\phi(t_n))-f(\phi(t_{n+1}))]\\ &= \mathcal{O}\left(\Delta t^2\tfrac{\partial^3\phi}{\partial t^3}\right)+\mathcal{O}\left(\tfrac{\Delta t}{\epsilon^3}\right)\\ &= \mathcal{O}\left(\tfrac{\Delta t}{\epsilon^3}\right). \end{aligned}$$

From the analysis, we note that although the corresponding time errors (to the leading order) are of the same order for $\theta=1.0$ and $\theta=0.5$, but the magnitude of the error for the case $\theta=0.5$ is much smaller (since

there is a scaling factor 0.5 in front of the nonlinear term) than the case $\theta = 1.0$. This observation is further verified by the numerical experiments. For **Algorithm 2.2**, the time error can be analyzed in the same way. The leading order of time error for **Algorithm 2.2** is again $\mathcal{O}\left(\frac{\Delta t}{\epsilon^3}\right)$, no matter $\theta = 0.5$ or $\theta = 1.0$. The scaling factor θ will lead to different magnitude of the time error. One can expect that the algorithm with $\theta = 0.5$ outperforms better than the case $\theta = 1.0$.

By comparing the leading order of the time errors, we see that **Algorithm 2.3** is much better than **Algorithm 2.1** and **Algorithm 2.2**. **Algorithm 2.3** can achieve the same accuracy as **Algorithm 2.1**, even the time step size used is much larger than that for **Algorithm 2.1** and **Algorithm 2.2**. Furthermore, even for the same algorithm, the choices of θ will affect the total numerical errors. We further justify these points by the numerical experiments in Section 4.

2.2 Finite Element Approximation.

Let $L^2(\Omega)$ be the space of square integrable functions with norm denoted by $||\cdot||$ and inner product $(\cdot,\cdot)$. We denote T_h as a quasi-uniform quadrilateral decomposition of Ω and assume that $\bar{\Omega} = \cup_{K\in T_h}\bar{K}$. In this work, $Q1$ elements or $Q2$ elements are used for discretiazation and the corresponding finite element space is denoted as Φ_h.

With the notations defined as above, the full discrete scheme based on (7) reads as: Given the solution $\phi(t_n) \in \Phi_h$, find $\phi(t_{n+1}) \in \Phi_h$ such that

$$\begin{aligned}(\phi(t_{n+1}),\psi) + \theta\Delta t\left[(\nabla\phi(t_{n+1}),\nabla\psi) + \frac{1}{\epsilon^2}(f(\phi(t_{n+1})),\psi)\right] = (\phi(t_n),\psi)\\ -(1-\theta)\Delta t\left[(\nabla\phi(t_n),\nabla\psi) + \frac{1}{\epsilon^2}(f(\phi(t_n)),\psi)\right], \quad \forall\psi\in\Phi_h.\end{aligned} \tag{8}$$

Similarly, the full discrete schemes of (5) and (6) are

$$\begin{aligned}(\phi(t_{n+1}),\psi) + \theta\Delta t(\nabla\phi(t_{n+1}),\nabla\psi) = (\phi(t_n),\psi)\\ -(1-\theta)\Delta t(\nabla\phi(t_n),\nabla\psi) + \Delta t\frac{1}{\epsilon^2}(f(\phi(t_n)),\psi), \quad \forall\psi\in\Phi_h\end{aligned} \tag{9}$$

and

$$\begin{aligned}(\phi(t_{n+1}),\psi) + \theta\Delta t\left[(\nabla\phi(t_{n+1}),\nabla\psi) + \frac{1}{\epsilon^2}(\phi(t_n)^2-1)(\phi(t_{n+1}),\psi)\right] = (\phi(t_n),\psi)\\ -(1-\theta)\Delta t\left[(\nabla\phi(t_n),\nabla\psi) + \frac{1}{\epsilon^2}(f(\phi(t_n)),\psi)\right], \quad \forall\psi\in\Phi_h.\end{aligned} \tag{10}$$

For the full discrete schemes, the error analyses are not easy. Some convergence analyses of the full discrete scheme (8) are established based on an abstract argument (for instance, [7, 8]). We are interested in the results which can provide information for the choice of h in connection with ϵ. The authors of [7] consider a coupled Allen-Cahn-Navier-Stokes model. For the semi-discrete scheme (the discretization is only applied for spatial variable, not for the time variable) used in [7], it is proved that if $h = o(\epsilon^2)$, then the scheme with the first order spatial accuracy converges (Theorem 3.1 in [7]). However, it seems that the assumption $h = o(\epsilon^2)$ is still too strong. It is notable that Zhang and Du in [19] provide some tentative analysis based on a different approach. By conducting numerical experiments and the analysis of the error for the bending energy approximation, it is argued that one can choose h to be of order $\mathcal{O}(\epsilon^\alpha)$, where $\alpha > 1$ and depends on the order of the spatial accuracy (More clearly, α will be closer to 1 if higher order elements

are used). This means that the mesh should be refined as ϵ gets smaller and smaller. Indeed, all these investigations provide some arguments of the error analysis. However, it is still not easy to tell under what condition which error is dominant. The error from the model approximation, the spatial error and the time error all contribute to the total error. We conduct numerical experiments to provide some information.

Let N_ϕ be the dimension of the space Φ_h. The finite element solution at t_n can be expressed as

$$\phi(t_n) = \sum_{i=1}^{N_\phi} \phi_i^n b_i(\mathbf{x}),$$

where $\Phi^n = (\phi_1^n, \phi_2^n, ..., \phi_{N_\phi}^n)^T$ is the set of coefficients for the phase field variable, and $\mathbf{b} = (b_1, b_2, ..., b_{N_\phi})^T$ is the set of basis functions. We introduce the following matrices and vectors: mass matrix $\mathbf{M}$, stiffness matrix $\mathbf{L}$, a vector $\mathbf{N}(\Phi^{n+1})$, and the right hand side $\mathbf{F}$, with the following components:

$$\mathbf{M}(i,j) = \int_\Omega b_i b_j d\Omega, \qquad \mathbf{L}(i,j) = \int_\Omega \nabla b_i \cdot \nabla b_j d\Omega, \quad 1 \le i,j \le N_\phi,$$

$$\mathbf{N}(\Phi^{n+1})(i) = \int_\Omega f(\phi(t_{n+1})) b_i d\Omega, \quad 1 \le i \le N_\phi,$$

$$\mathbf{F}(i) = \int_\Omega \phi(t_n) b_i d\Omega - (1-\theta)\Delta t \int_\Omega \nabla \phi(t_n) \cdot \nabla b_i + \frac{1}{\epsilon^2} f(\phi(t_n)) b_i d\Omega, \quad 1 \le i \le N_\phi.$$

Then, the global system for (8) can be written as

$$[\mathbf{M} + \theta \Delta t \mathbf{L}]\Phi^{n+1} + \theta \Delta t \frac{1}{\epsilon^2} \mathbf{N}(\Phi^{n+1}) = \mathbf{F}. \tag{11}$$

Similarly, the algebraic systems for (9) and (10) are

$$[\mathbf{M} + \theta \Delta t \mathbf{L}]\Phi^{n+1} = \bar{\mathbf{F}} \tag{12}$$

with

$$\bar{\mathbf{F}}(i) = \int_\Omega \phi(t_n) b_i d\Omega - (1-\theta)\Delta t \int_\Omega \nabla \phi(t_n) \cdot \nabla b_i + \int_\Omega \frac{1}{\epsilon^2} f(\phi(t_n)) b_i d\Omega$$

and

$$[(1 + \theta \Delta t \frac{1}{\epsilon^2}(\phi(t_n)^2 - 1))\mathbf{M} + \theta \Delta t \mathbf{L}]\Phi^{n+1} = \tilde{\mathbf{F}} \tag{13}$$

with

$$\tilde{\mathbf{F}}(i) = \int_\Omega \phi(t_n) b_i d\Omega - (1-\theta)\Delta t \int_\Omega \nabla \phi(t_n) \cdot \nabla b_i + \frac{1}{\epsilon^2} f(\phi(t_n)) b_i d\Omega.$$

2.3 Linear and Nonlinear Solvers.

For the algebraic system (11), the nonlinear solver is a Newton iteration. We implement it as a special case of the defect correction method [16]. Specifically, for a given nonlinear system

$$\mathbf{T}(\mathbf{x})\mathbf{x} = \mathbf{g},$$

we apply the iteration

$$\mathbf{x}^{(n+1)} = \mathbf{x}^{(n)} + \omega^n \tilde{\mathbf{T}}^{-1}(\mathbf{x}^{(n)})(\mathbf{g} - T(\mathbf{x}^{(n)})\mathbf{x}^{(n)}),$$

where $\tilde{\mathbf{T}}^{-1}(\mathbf{x}^{(n)})$ can be an approximation of the Frechét-derivative of $\mathbf{T}$ to the last iterate $\mathbf{x}^{(n)}$, and ω^n is a relaxation parameter in each step of the iteration. More precisely, we take

$$\tilde{\mathbf{T}}(\Phi^{(n)}) = \mathbf{M} + \theta\Delta t\mathbf{L} + \theta\Delta t\frac{1}{\epsilon^2}\mathbf{N}'(\Phi^{(n)})$$

with

$$\mathbf{N}'(\Phi^{(n)})(i,j) = \int_\Omega ((3\sum_{k=1}^{N_\phi}\Phi_k^{(n)}b_k)^2 - 1)b_i b_j d\Omega,$$

where $\Phi^{(n)} = (\Phi_1^{(n)}, \Phi_2^{(n)}, ..., \Phi_{N_\phi}^{(n)})^T$ is the set of coefficients of the solution at the n-th step of the nonlinear iteration. We stop the Newton iteration if a maximum iteration counter is exhausted, $n \geq N_{MAX}$, or if the norm of the nonlinear residual is sufficiently small:

$$||\mathbf{T}(\mathbf{x}^{(n+1)})\mathbf{x}^{(n+1)} - \mathbf{g}||_{l^2} < 1.0 \times 10^{-8}.$$

The linear solvers for (12), (13), and each step of the Newton iteration are all based on a geometric multigrid algorithm [16]. In each multigrid iteration, several steps of incomplete LU decomposition are used for presmoothing and postsmoothing since Gauss-Seidel iteration or Jacobi iteration will not work for the TDGL model with small ϵ. Gauss elimination is used as the coarsest grid solver. We stop the multigrid iteration if the l^2- norm of the relative residual is smaller than 1.0×10^{-6}.

3 Moving Grid Deformation Based Algorithms.

In many PDE problems, adaptive grids are necessary. For some particular problems, locally adapted grids can improve both the accuracy and the efficiency. In this work, we adopt the grid deformation algorithm newly developed in [10, 11]. It is a generic approach for generating computational grids and has been successfully applied to both stationary and time dependent partial differential equations [6, 10, 12, 17]. In this section, we first introduce the detailed algorithm, then describe on how to combine the adaptive grid with the algorithms in Section 2.

3.1 Grid Deformation Algorithm.

Generally speaking, grid deformation is constructing a transformation B, from computational space Ω (with coordinate $\mathbf{x}_c$) to physical space $\hat{\Omega}$ (with coordinate $\mathbf{x}$), $\mathbf{x} = B(\mathbf{x}_c)$. For simplicity, we assume that $\hat{\Omega} = \Omega$ is a two dimensional domain, although the deformation algorithm [10, 11] is valid for both two dimensional and three dimensional problems. To introduce the algorithm, we import the notations: a weighting function $g(\mathbf{x}) > 0$, a monitor function $m(\mathbf{x}) > 0$, and the Jacobian matrix of the transformation $\frac{\partial \mathbf{x}}{\partial \mathbf{x}_c}$.

The proposed grid deformation algorithm in [10] is to construct a bijection B satisfying

$$g(\mathbf{x}_c)\left|\det\left(\frac{\partial \mathbf{x}}{\partial \mathbf{x}_c}\right)\right| = m(B(\mathbf{x}_c)), \quad \mathbf{x}_c \in \Omega, \tag{14}$$

and

$$B : \partial\Omega \to \partial\Omega. \tag{15}$$

It is easy to see that (15) means that the bijection B maps the boundary nodes again to boundary nodes. To have a deeper understanding of (14), we apply integration on an element $T \subset \Omega$

$$S(B(T)) := \int_{B(T)} 1 d\mathbf{x} = \int_T \left| \det \left(\frac{\partial \mathbf{x}}{\partial \mathbf{x}_c} \right) \right| d\mathbf{x}_c,$$

where $S(B(T))$ is the area of the mapped element. Using the 1×1 quadrature rule in the above integral and using (14), we see that

$$g(\mathbf{x}^*) \frac{S(B(T))}{S(T)} = m(B(\mathbf{x}^*)) + \mathcal{O}(h). \tag{16}$$

Here $S(T)$ is the area of T and $\mathbf{x}^*$ is the center of T. From (16), we observe that $m(\mathbf{x})$ actually describes the relative growth or shrinkage of the element with respect to the undeformed mesh if $g \equiv 1$ (This special case has been investigated by Liao in [12].). While, both g and m can be more general. In the following algorithm, we consider g to be the area distribution on the undeformed mesh and m shall be the absolute mesh size distribution of the target grid. The basic idea for grid deformation is calculating the mesh velocity by using a Lagrangian-like formulation [10, 12]. The construction of the transformation B is realized via the following four steps:

Algorithm 3.1.

1. Calculate the scaling factors c_m and c_g such that

$$c_m \int_\Omega \frac{1}{m(\mathbf{x})} d\mathbf{x} = c_g \int_\Omega \frac{1}{g(\mathbf{x})} d\mathbf{x} = |\Omega|.$$

   Denote the reciprocals of the scaled functions m and g by $\tilde{m}$ and $\tilde{g}$ respectively, i.e.,

$$\tilde{m} = \frac{c_m}{m}, \quad \tilde{g} = \frac{c_g}{g}.$$

2. Compute a grid-velocity vector field $\mathbf{v} : \Omega \to R^2$ by satisfying the following linear Poisson problem

$$-\nabla \cdot \mathbf{v}(\mathbf{x}) = \tilde{m}(\mathbf{x}) - \tilde{g}(\mathbf{x}) \quad \text{and} \quad \mathbf{v}(\mathbf{x}) \cdot \mathbf{n} = 0, \ \mathbf{x} \in \partial\Omega. \tag{17}$$

   Here $\mathbf{v} = \nabla w$ with w being a scalar function and $\mathbf{n}$ is the outer normal vector of the domain boundary.

3. For each grid point $\mathbf{x}$, solve the following ODE system

$$\frac{\partial G(\mathbf{x}, t)}{\partial t} = \eta(G(\mathbf{x}, t), t), \quad 0 \le t \le 1, \quad G(\mathbf{x}, 0) = \mathbf{x}, \tag{18}$$

   with

$$\eta(\mathbf{y}, s) := \frac{\mathbf{v}(\mathbf{y})}{s\tilde{m}(\mathbf{y}) + (1-s)\tilde{g}(\mathbf{y})}, \quad \mathbf{y} \in \Omega, s \in [0, 1].$$

4. Get the new grid points via

$$B(\mathbf{x}) := G(\mathbf{x}, 1).$$

The most time consuming part of the above algorithm is the second step. (17) defines a Poisson problem with pure Neumann boundary condition. We solve the Poisson problem by using a state of the art multigrid algorithm. Therefore the whole algorithm is very efficient. The third step of the above algorithm can be

solved by existing ODE solvers like Runge-Kutta method. We apply explicit Euler scheme to (18) and it will not affect the global efficiency too much [10, 11].

The grid deformation algorithms proposed in [10, 11] has several advantages over the existing works. Fisrstly, it is a generalization of Liao's and Moser's work [6, 12], and it prevents mesh tangling and offers precise control over the element volumes. Secondly, it can be applied for any kind of discretizations, finite element, finite difference, finite volume, or whatever. Thirdly, its numerical realization requires only a Poisson problem and an initial value problem. Many other deformation methods, in contrast, involve the solution of complicated nonlinear partial differential equations. Fourthly but not lastly, the data structure for the mesh nodes is always the same as for the starting mesh.

As an example, we show an adaptive mesh concentrated at an elliptic interface in Figure 1. The domain

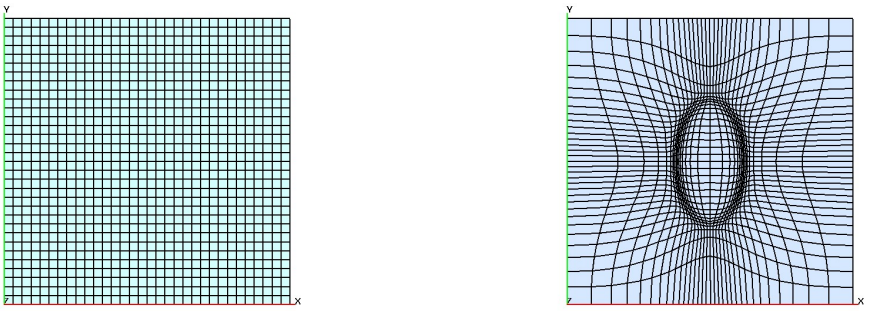

Figure 1: A 32×32 uniform grid of the computational domain is deformed into an adaptive grid with grid points concentrated near the elliptic interface.

$\Omega = [0,1] \times [0,1]$ is triangulated by an equidistant tensor product mesh. The ellipse is defined by the following parameterized form:

$$x(s) = 0.5 + 0.12\cos(s), \quad y(s) = 0.5 + 0.22\sin(s), \quad s \in [0, 2\pi].$$

Here, the monitor function is defined as

$$m(\mathbf{x}) = \min\left\{1, \max\left\{d, \delta\right\}\right\}, \quad d := \sqrt{(x - x(s))^2 + (y - y(s))^2}, \quad \mathbf{x} = (x, y) \in \Omega.$$

We set $\delta = 0.01$. It means that on the deformed grid, the largest cell has 100 times the area of the smallest one. In this example, the monitor function is chosen according to the distance to the given ellipse. Thus, the grid is concentrated around its contours. Actually, one may add more parameters to the distance function. For instance, one can use Cd^{β} to replace the distance function d, where C and β are used to control the magnitude of mesh concentration. It is true that the monitor function should be problem dependent. We try to give some discussions on the monitor function for the model studied in this paper. One may also try other choices of monitor functions, for instance, one can determine the monitor function according to the solution error or the boundary curvature [3, 18].

3.2 Moving Mesh PDE Formulation.

When time dependent problems are solved, in particular those problems with the evolution of singularities or large variations, moving grids are necessary for improving the accuracy. Basically, there are two ways in the implementations of moving grid techniques for time dependent problems. One is using moving mesh PDE (MMPDE) formulation, incorporating the mesh velocity into the original PDE [15]. The other is using the original governing equation, while interpolating the solutions between the old mesh and the new mesh in each time step of the computation [20]. The first one is equivalent to variable substitution. The second one is based on the mapping between different meshes. To avoid the interpolation between old mesh and new mesh, our algorithms are based on the MMPDE formulation.

To have a clear picture of the MMPDE formulation of (1), we write the governing equation in the new coordinate $(\mathbf{x}_c, t)$ through the transform $\mathbf{x} = \mathbf{x}(\mathbf{x}_c, t)$. Denote $\phi((\mathbf{x}(\mathbf{x}_c, t), t)$ as $\psi(\mathbf{x}_c, t)$, then

$$\frac{\partial \psi(\mathbf{x}_c, t)}{\partial t} = \frac{\partial \phi((\mathbf{x}(\mathbf{x}_c, t), t)}{\partial t} = \frac{\partial \phi((\mathbf{x}(\mathbf{x}_c, t))}{\partial t} + \dot{\mathbf{x}} \cdot \nabla \phi(\mathbf{x}, t),$$

where $\dot{\mathbf{x}} = \frac{\partial \mathbf{x}}{\partial t}$ represents the derivative of $\mathbf{x}$ with respect to t. To simplify the notation, we shall use the same notations to denote the functions ϕ, before and after the transformation. Therefore, the moving mesh PDE formulation for (1) is as follows [15],

$$\frac{\partial \phi}{\partial t} - \dot{\mathbf{x}} \cdot \nabla \phi = -(-\Delta \phi + \frac{1}{\epsilon^2} f(\phi)). \tag{19}$$

Now, let us design algorithms based on the MMPDE formulation (19). Note that $\dot{\mathbf{x}}$ is also a time derivative, we shall not scale θ or $1-\theta$ for the term $\dot{\mathbf{x}} \cdot \nabla \phi$. Thus, the algorithm based on the MMPDE formulation is as follows

$$\begin{aligned} \frac{\phi(t_{n+1}) - \phi(t_n)}{\Delta t} - \dot{\mathbf{x}}_{n+1} \cdot \nabla \phi(t_{n+1}) + \theta \left[-\Delta \phi(t_{n+1}) + \frac{1}{\epsilon^2} f(\phi(t_{n+1})) \right] = \\ -(1-\theta) \left[-\Delta \phi(t_n) + \frac{1}{\epsilon^2} f(\phi(t_n)) \right], \end{aligned} \tag{20}$$

where $\dot{\mathbf{x}}_{n+1}$ is an approximation of the velocity speed at time step t_{n+1}. More precisely, we calculate the mesh speed at t_{n+1} by

$$\dot{\mathbf{x}}_{n+1} \approx \frac{\mathbf{x}_{n+1} - \mathbf{x}_n}{\Delta t}. \tag{21}$$

Similarly, the corresponding MMPDE formulation based algorithms for **Algorithm 2.1** and **Algorithm 2.2** are

$$\frac{\phi(t_{n+1}) - \phi(t_n)}{\Delta t} - \dot{\mathbf{x}}_{n+1} \cdot \nabla \phi(t_{n+1}) - \theta \Delta \phi(t_{n+1}) = (1-\theta)\Delta \phi(t_n) - \frac{1}{\epsilon^2} f(\phi(t_n)) \tag{22}$$

and

$$\begin{aligned} \frac{\phi(t_{n+1}) - \phi(t_n)}{\Delta t} - \dot{\mathbf{x}}_{n+1} \cdot \nabla \phi(t_{n+1}) + \theta \left[-\Delta \phi(t_{n+1}) + \frac{1}{\epsilon^2} (\phi(t_n)^2 - 1)\phi(t_{n+1}) \right] \\ = -(1-\theta) \left[-\Delta \phi(t_n) + \frac{1}{\epsilon^2} f(\phi(t_n)) \right]. \end{aligned} \tag{23}$$

In sum, the complete algorithm is using **Algorithm 3.1** to generate a new mesh $\mathbf{x}_{n+1}$ and using (21) to calculate the mesh speed, then solving for $\phi(t_{n+1})$ by one of (20), (22) and (23).

4 Numerical Experiments.

In this section, we compare the accuracy and efficiency of the different algorithms. The benchmark problems concern the shrinkage of circular bubbles. Under the mean curvature force, the circular interface will shrink and eventually vanish. However, the numerical vanishing speed will be affected by the numerical scheme used, the time stepsize and the spatial approximations. Therefore, we will first check which algorithm can accurately capture the vanishing speed on the uniform mesh. Later, we show that the accuracy and the efficiency of the algorithms can be greatly improved by the moving grid deformation technique. Comparisons with the numerical results obtained by using the uniform grid are also reported to verify the advantages of the moving grid deformation based algorithm.

Example 1. This example is a well-known benchmark problem which has been tested in [5] and [15]. In [5], the computational domain is $[0, 256] \times [0, 256]$ and the interface thickness $\epsilon = 1$. The initial solution is given by

$$\phi(x, 0) = -\tanh\left(\frac{\sqrt{(x-128)^2 + (y-128)^2} - 100}{\epsilon}\right). \tag{24}$$

This initial solution represents a circular bubble centered at $(128, 128)$ with radius being equal to 100. The order parameter values inside the circle are $+1$ and -1 outside. Compared with the domain size, the interface thickness is very small. The solution of equation (1) is unstable and the bubble will shrink [2]. Note that the TDGL model is an approximation of the sharp interface model, by the theory of the sharp interface model, the shrinking speed of the sharp interface is given by

$$\frac{dR}{dt} = -\frac{1}{R}, \tag{25}$$

where R is the radius of the circle at a given time. The negative sign means that the circle shrinks towards its center. Furthermore, the vanishing speed (25) gives the area of the circle at the given time t,

$$A = \pi R_0^2 - 2\pi t,$$

where $R_0 = 100$ is the initial radius. Therefore, the circular bubble, with the initial solution given by (24), vanishes at $t = 5000$. After mapping the domain to $[-1, 1] \times [-1, 1]$ and scaling the time variable t with ϵ^2, we obtain equation (1) and the initial profile is given by $\phi(\mathbf{x}, 0) = -\tanh\left(\frac{\sqrt{x^2+y^2}-r_0}{\epsilon}\right)$, where $\epsilon = \frac{1}{128}$ and $r_0 = \frac{100}{128}$. Then the bubble will vanish at $t = 5000\epsilon^2$.

For comparing the different algorithms, we measure the numerical results by computing the accumulated error. More precisely, if the numerical bubble does not vanish at $t = 5000\epsilon^2$ and the area is A_{num}, then the accumulated error is defined as

$$error = \frac{A_{num}}{\pi r_0^2}.$$

We calculate A_{num} by

$$A_{num} = \frac{1}{2}\int_\Omega [\phi_h(\mathbf{x}, 5000\epsilon^2) + 1] d\Omega. \tag{26}$$

If the numerical results give that the bubble vanishes at a time $t = t_{num} < 5000\epsilon^2$, then the accumulated error is computed by

$$error = -\frac{\pi r_0^2 - 2\pi t_{num}}{\pi r_0^2}.$$

El	θ	$\Delta t(\epsilon^2)$	$h(\epsilon)$	Err(%)
$Q1$	1	1.0	1	-4.64
	0.5	1.0	1	-4.6
	1	0.25	1	-4.59
	1	0.5	0.5	-1.31
	0.5	0.5	0.5	-1.29
	0.5	0.25	0.5	-1.28
$Q2$	1	1.0	2	0.80
	0.5	1.0	2	0.93
	1	0.5	2	0.89
	0.5	1.0	1	-0.26
	1	0.25	1	-0.22
	0.5	0.25	1	-0.2

Table 1: Numerical results based on **Algorithm 2.3**.

El	θ	$\Delta t(\epsilon^2)$	$h(\epsilon)$	Err(%)	Time Err (%)
$Q1$	1	0.25	1	-14.38	-9.79
	1	0.125	1	-9.48	-4.89
	0.5	0.25	1	-9.47	-4.88
	0.5	0.125	1	-7.03	-2.44
$Q2$	1	0.25	2	-9.4	-10.29
	0.5	0.25	2	-4.25	-5.14
	1	0.125	1	-5.2	-5.0
	0.5	0.125	1	-2.69	-2.49

Table 2: Numerical results based on **Algorithm 2.2**.

Here, the negative sign means that the numerical bubble shrinks faster than the sharp interface.

We first use uniform grids to compare different algorithms. Different time step sizes and mesh sizes are used for configurations. Recall that the leading order of the local truncation error is of order $\mathcal{O}(\frac{\Delta t}{\epsilon^3})$ for **Algorithm 2.1** and **Algorithm 2.2** and note that the quantity defined by (26) involves an integral over the domain. The time error based on these two algorithms, to leading order, should be $\mathcal{O}(\frac{\Delta t}{\epsilon^2})$. Although the leading order of time error for **Algorithm 2.3** is better, however we note that the linear system for 2D or 3D problem is too stiff to solve if the time step size is of order $\mathcal{O}(\epsilon)$. Therefore, the time step sizes are set to be proportional to ϵ^2 for all algorithms, and the mesh sizes are chosen to be proportional to ϵ.

We first analyze the results obtained by **Algorithm 2.3**. We note that, for fixed mesh size, the error will not improve too much by reducing the time step size. The error gets saturated because the time step size is very small and the spatial errors are dominant for all the configurations. This observation coincides with the numerical results obtained by Zhang and Du [19]. They test a similar one dimensional benchmark problem and note that the spatial error is dominant if the time step size is small enough. Another observation is that, $Q2$ element based results are much better than $Q1$ element based results. For instance, when $\Delta t = \epsilon^2$ and $\theta = 1.0$, the accumulated error is 0.81%. This clearly shows that high order discretization can reduce the

El	θ	$\Delta t(\epsilon^2)$	$h(\epsilon)$	Err(%)	Time Err (%)
$Q1$	1	0.5	1	12.2	16.79
	1	0.25	1	4.45	9.04
	1	0.125	1	0.03	4.62
	0.5	0.5	1	2.34	6.93
	0.5	0.25	1	-0.1	4.49
	0.5	0.125	1	-1.3	3.29
$Q2$	1	0.25	2	9.92	9.03
	1	0.125	2	5.65	4.76
	1	0.25	1	8.94	9.14
	1	0.125	1	4.62	4.82
	0.5	0.25	1	2.39	2.59
	0.5	0.125	1	0.93	1.13

Table 3: Numerical results based on **Algorithm 2.1**.

accumulated error. For **Algorithm 2.3**, there seems to be no significant difference for the results by using different θ (because time step size is very small, spatial error is dominant). However, we do find that when $\theta = 0.5$, it allows larger time step size than the case $\theta = 1.0$. We report here that if $Q1$ is used and $h = \epsilon$, Δt can be as large as $2.0\epsilon^2$ when $\theta = 0.5$, but this setting is unstable for the case $\theta = 1.0$. In addition, we mention here that the maximum $\Delta t = 0.5\epsilon^2$ is applied for the semi-implicit Fourier spectral method in [5].

For **Algorithm 2.2**, we observe that the linearization technique tends to make the bubble vanishing faster than expected. The numerical results reported here show that all the accumulated errors have negative sign. However, we do see that by setting $\theta = 0.5$, one can get better results than those based on $\theta = 1.0$. Given a time step size, if the spatial error is small (for instance, $Q2$ element is used), the numerical error based on $\theta = 0.5$ is almost half of that based on $\theta = 1.0$. We also see that the time error is reduced to be half if the time step size is refined once. Therefore, we conclude that one can improve the numerical accuracy by taking $\theta = 0.5$, using a higher order element and setting a smaller time step size for this algorithm.

For algorithm **Algorithm 2.1**, for relative large time step size, if the element and the mesh size are fixed, then the results based on $\theta = 0.5$ are much better than those based on $\theta = 1.0$. This clearly shows that the numerical results may be improved by choosing proper . Secondly, for fixed mesh size, by reducing the time step size, the numerical accuracy can be improved. For **Algorithm 2.1** and **Algorithm 2.2**, we calculate the approximate time error by deducting the corresponding spatial error from the total error. For instance, when $h =$ and $Q1$ element is used, the spatial error is around 4.59%. We therefore deduct this number for **Algorithm 2.1** and **Algorithm 2.2** if $h =$ and $Q1$ elements are used. By this way, we see that the approximate time errors based on these two algorithms seem to be proportional to $\frac{\Delta t}{\epsilon^2}$. We also note that if the time step size is properly selected, the total error by using **Algorithm 2.1** is very small. The total error is very small perhaps because the spatial error is well balanced with the time error.

We report here that it only takes 2 to 4 steps for the nonlinear iteration in **Algorithm 2.3** to converge, because the nonlinear term in TDGL model is of polynomial type. Moreover, **Algorithm 2.3** allows for large time step size. Therefore, the overall computational costs for **Algorithm 2.3** is smaller than those for **Algorithm 2.1** and **Algorithm 2.2**. In sum, we conclude that **Algorithm 2.3** outperforms the other two algorithms.

To improve the accuracy and efficiency of the above algorithms, it is better to use moving grid defor-

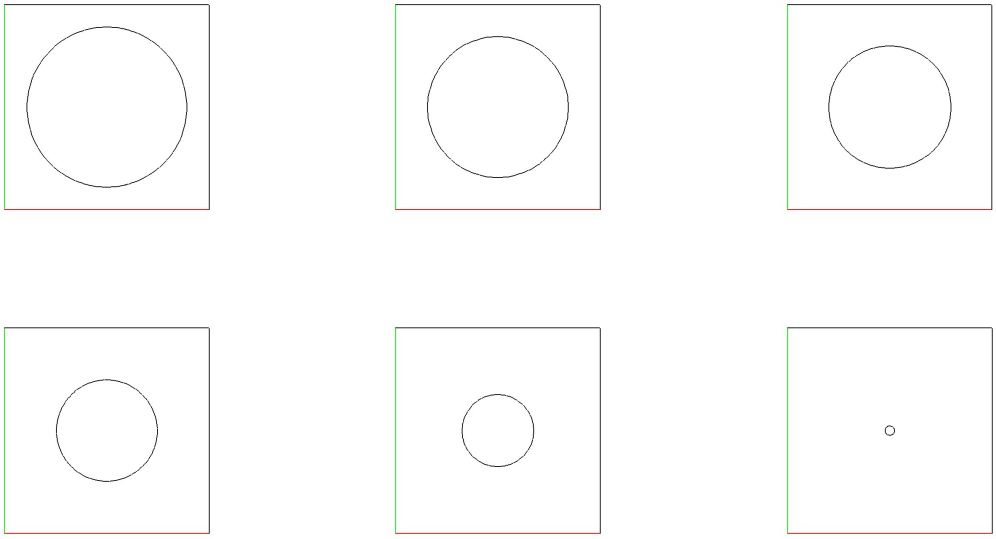

Figure 2: The snapshots of the numerical interface at $t = 0, 1000\epsilon^2, 2000\epsilon^2, 3000\epsilon^2, 4000\epsilon^2, 5000\epsilon^2$.

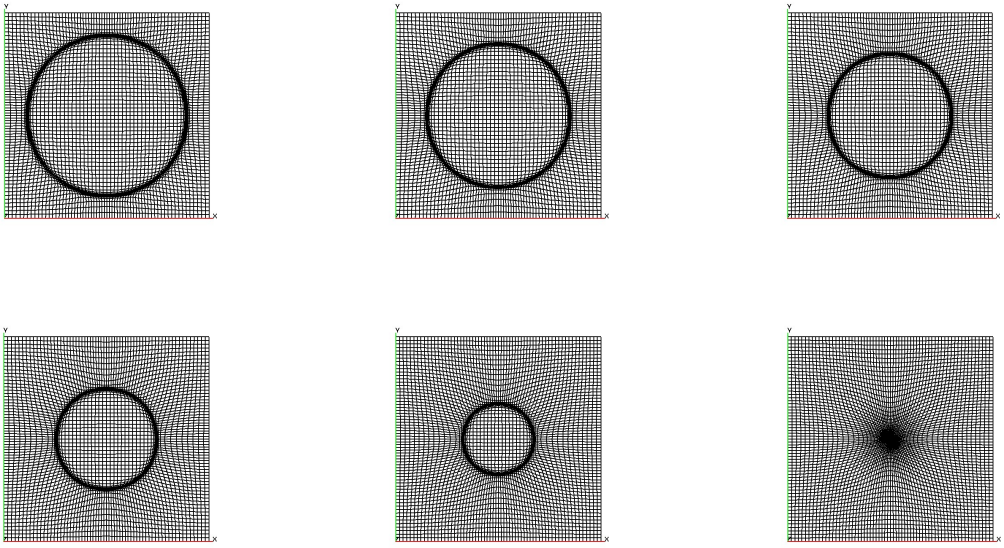

Figure 3: The snapshots of the mesh at $t = 0, 1000\epsilon^2, 2000\epsilon^2, 3000\epsilon^2, 4000\epsilon^2, 5000\epsilon^2$.

mation algorithms so that they can accurately track the interfacial dynamics. In Figure 2 and Figure 3, we show the snapshots of the numerical interface and the corresponding adapted meshes. Numerical results are obtained by using (20) and the parameters are: $Q1$ elements, $\theta = 1.0$ and $\Delta t = \epsilon^2$. The mesh size in the computational domain is $h = 4\epsilon$. The monitor function is $m(\mathbf{x}, t) = \min\{1, \max\{d(\mathbf{x}, \Gamma)/(10\epsilon), 0.05\}\}$, where $d(\mathbf{x}, \Gamma)$ is the distance to the interface and can be obtained by using the theory (25) or by reconstruction according to the numerical solution. With this setting, the accumulated error is 1.08%. Since the maximum degree of freedoms is much less than that used in the uniform mesh cases ($h = 4\epsilon$ versus $h = \epsilon$), the overall computation cost for moving grid deformation based algorithm is much less than that for uniform grid based algorithm. Moreover, by using moving grid deformation, we observe that the accuracy of the solution is greatly improved. We comment here that better monitor function can be designed to have further improvement of the accuracy.

We also combined the moving grid deformation algorithm with **Algorithm 2.1** and **Algorithm 2.2**. For comparing the efficiency, we list the total CPU time for the moving grid deformation based algorithms in Table 4. We report here that the total CPU time for **Algorithm 2.3** with the parameters setting $h = \epsilon$, $\Delta t = \epsilon^2$ and $Q1$ element is 31784 seconds. For moving grid based algorithms, we set $h = 4\epsilon$. in the computational domain. From the numerical experiments, we observe that when $\theta = 1.0$ both **Algorithm 2.1** and **Algorithm 2.2** are unstable if $\Delta t \geq 0.2\epsilon^2$. The instability perhaps is caused by the fact that the time step size is affected by the local mesh size. However, if we set $\theta = 0.5$ in these two algorithms, the time step size can be relative large and the computational costs are much less than the uniform grid based results. Moreover, for **Algorithm 2.3**, the time step size seems to be independent of the local mesh size. The total CPU time of **Algorithm 2.3** with moving grids is much less than that of uniform mesh based algorithm. Therefore, we conclude that the moving grid deformation technique can significantly improve the efficiency.

Algorithm	Element	θ	$\Delta t(\epsilon^2)$	$h(\epsilon)$	CPU time
Alg2.1	$Q1$	1.0	0.2	4	unstable
		1.0	0.2	4	23671
Alg2.2	$Q1$	0.5	0.2	4	unstable
		0.5	0.2	4	24634
Alg2.3	$Q1$	0.5	1.0	4	9919
	$Q2$	0.5	1.0	8	3273

Table 4: CPU time (in seconds) for moving grid deformation based algorithms.

Example 2. The second example is concerned with the shrinking of elliptic bubble. The domain is $\Omega = [-1, 1] \times [-1, 1]$. The interface thickness is $\epsilon = 0.05$. The initial solution is

$$\phi(x, y, 0) = \tanh\left(\frac{1}{\epsilon}\left(\frac{x^2}{0.09} + \frac{y^2}{0.025} - 1\right)\right). \tag{27}$$

For comparisons, we solve this problem by using both uniform mesh based algorithm and moving grid algorithm. For uniform mesh configuration, we set $\theta = 0.5$, $Q1$ element, 512×512 tensor mesh, $\Delta t = 5.0 \times 10^{-5}$. In Figure 4, we list the snapshots of the numerical interface at different time by using the uniform mesh. For deformed mesh configuration, we use 64×64 grids in the computational domain and $\Delta t = 1.0 \times 10^{-4}$, other settings are same as those used in the uniform mesh configuration. In Figure 5, we show the snapshots of the deformed meshes. The monitor function is $m(\mathbf{x}, t) = \min\{1, \max\{0.25d^{0.4}(\mathbf{x}, \Gamma), 0.02\}\}$. Since the interface is not exactly known, we reconstruct the interface by first marking all elements where the solution changes the sign, then taking the centers of these elements and using picewise linear splines to

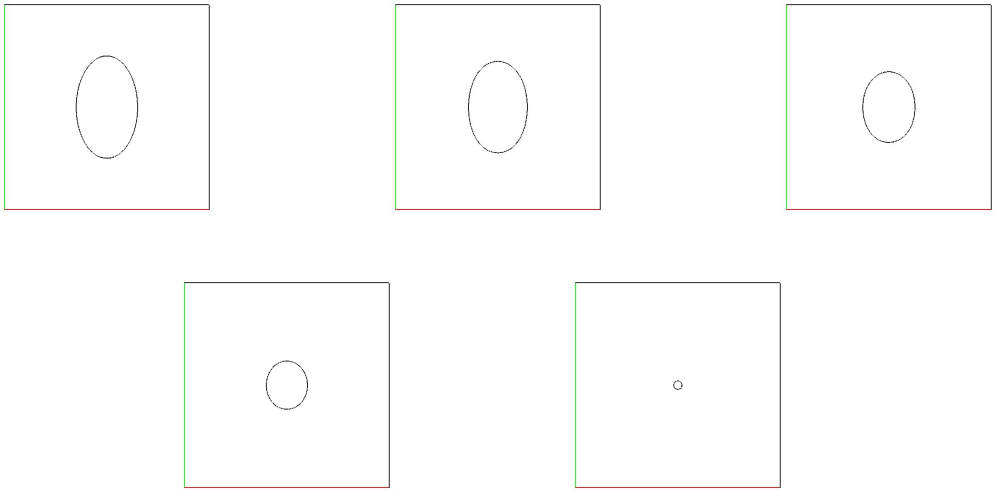

Figure 4: The snapshots of the numerical interface at $t = 0, 0.01, 0.03, 0.05, 0.07$.

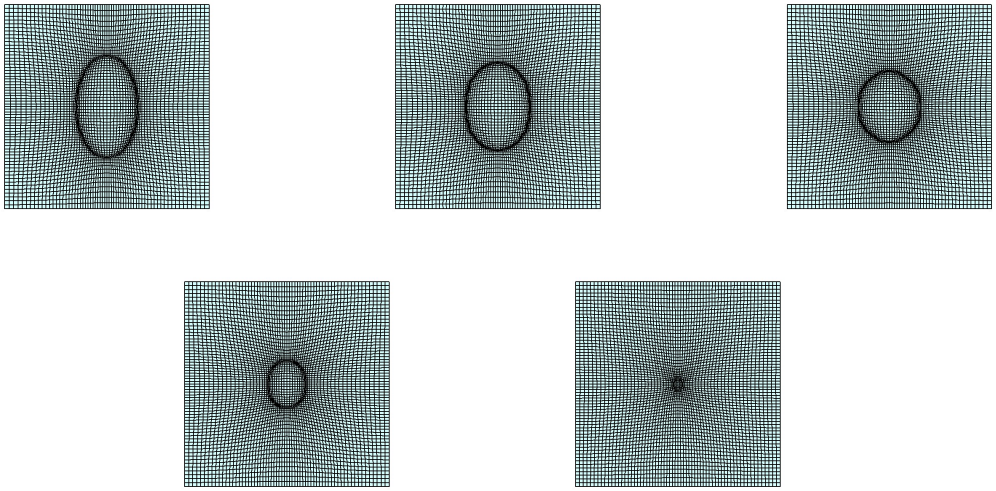

Figure 5: The adapted meshes at $t = 0, 0.01, 0.03, 0.05, 0.07$. (Numerical results obtained by using uniform mesh)

connect all centers. By this way, the error of the approximation of the interface is less than the smallest mesh size. The numerical results based on the uniform mesh show that the bubble vanishes at $t = 0.07045$. We

also observe that the algorithm based on deformed meshes gives a very accurate result. Most importantly, we see that the adapted grid is concentrated at the interfacial region.

5 Conclusions and Outlook.

Several numerical algorithms for solving the TDGL model have been proposed and tested. Moreover, a new moving grid deformation technique is applied for improving the accuracy and efficiency of all algorithms. Firstly, it is observed that the numerical scheme is crucial for providing accurate numerical solutions. From the numerical experiments, we conclude that the nonlinear scheme (7) allows for large time step size and gives more accurate results. Note that the nonlinear term is a polynomial of degree 3, therefore the nonlinear solver converges in few iterations. Secondly, by using the moving grid deformation technique, we see that numerical results are greatly improved for both the accuracy and efficiency. The applied grid deformation algorithm is efficient and robust because only linear Poisson problems on fixed meshes are needed to be solved and the mesh tangling can be prevented.

From the numerical studies, we also note some important problems: Firstly, more robust linear and nonlinear solvers are necessary. Although the local truncation error analysis shows that the time step size may be as large as $\mathcal{O}(\epsilon)$ for **Algorithm 2.3**, but the linear and nonlinear solver will break down for 2D or 3D problem if the time step size is of order $\mathcal{O}(\epsilon)$. Secondly, more studies are necessary to understand the moving grid algorithm. In particular, how to design monitor functions so that the moving grids are aligned with the solution of PDEs is not well understood. It is known that the monitor function should be problem dependent and is crucial for moving grid based simulations. In this paper, we have tested the distance based monitor function for the TDGL model. More works are necessary to understand the connections between the monitor function and the solution.

Acknowledgements. The authors would like to thank Michael Köster, Dmitri Kuzmin and Raphael Münster for many helpful discussions.

References

[1] M. O. Bristeau, R. Glowinski and J. Periaux, Numerical methods for the Navier-Stokes equations. Application to the simulation of compressible and incompressible flows, Comp. Phys., 6 (1987), pp.73-188.

[2] J. W. Cahn and S. M. Allen, A microscopic theory for domain wall motion and its experimental verification in Fe-Al alloy domain growth kinetics, J. Phys., 7 (1978) pp. 7-51.

[3] W. Cao,W. Huang, and R.D. Russell, A study of monitor functions for two dimensional adaptive mesh generation, SIAM J. Sci. Comput., 20 (1999), pp. 1978-1994.

[4] L. Chen, Phase-field models for microstructure evolution, Annu. Rev. Mater. Res., 32 (1) (2002), pp. 113-140.

[5] L. Q. Chen and J. Shen, Applications of semi-implicit Fourier-spectral method to phase-field equations, Comput. Phys. Commun., 108 (1998) pp. 147-158.

[6] B. Dacorogna and J. Moser, On a partial differential equation involving the Jacobian determinant, Ann. Inst. Henri Poincaré Anal. Non Linéaire, 7 (1990), pp. 1-26.

[7] X. Feng, Y. He, and C. Liu, Analysis of finite element approximations of a phase field model for two-phase fluids, Math. Comp., 76 (2007), pp. 539-571.

[8] X. Feng and A. Prohl, Numerical analysis of the Allen-Cahn equation and approximation for mean curvature flows, Numer. Math., 94 (2003), no. 1, pp. 33-65.

[9] X. Feng and H. Wu, A posteriori error estimates and an adaptive finite element method for the Allen-Cahn equationand the mean curvature flow, J. Sci. Comput., 24 (2005), no. 2, pp. 121-146.

[10] M. Grajewski, M. Köster and S. Turek, Mathematical and numerical analysis of a robust and efficient grid deformation method in the finite element context, SIAM J. Sci. Comput. 31 (2008/09), no. 2, pp. 1539-1557.

[11] M. Grajewski, M. Köster and S. Turek, Numerical analysis and implementational aspects of a new multilevel grid deformation method , Appl. Num. Math., in press.

[12] G. Liao and B. Semper, A moving grid finite element method using grid deformation, Numer. Methods for Partial Diff. Equat., 11 (1995) pp. 603-615.

[13] A phase field model for the mixture of two incompressible fluids and its approximation by a Fourier-spectral method, Physica D, 179 (2003), pp. 211-228.

[14] R. H. Nochetto and C. Verdi, Convergence past singularities for a fully discrete approximation of curvature-driven interfaces, SIAM J. Numer. Anal., 34 (1997), pp. 490-512.

[15] J. Shen and X. F. Yang, An efficient moving mesh spectral method for the phase field model of two phase flows, J. Comput. Phys., 228 (2009) pp. 2978-2992.

[16] S. Turek, Efficient solvers for incompressible flow problems: An algorithmic and computational approach. LNCSE 6, 1999, Springer Verlag.

[17] D. Wan and S. Turek, Fictitious boundary and moving mesh methods for the numerical simulation of rigid particulate flows, J. Comput. Phys., 222 (2007), no. 1, pp. 28-56.

[18] P.A. Zegeling, Theory and application of adaptive moving grid methods, published in the book "Adaptive Computations: Theory and Algorithms", edited by T. Tang and J. Xu (2007).

[19] J. Zhang and Q. Du, Numerical studies of discrete approximations to the Allen-Cahn equation in the sharp interface limit, SIAM J. Sci. Comput., 31 (2009), no. 4, pp. 3042-3063.

[20] Z. Zhang and H. Tang, An adaptive phase field method for the mixture of two incompressible fluids, Computers and Fluids, (2007), pp. 1307-1317.