Hindawi
Mathematical Problems in Engineering
Volume 2021, Article ID 5263989, 12 pages
https://doi.org/10.1155/2021/5263989

Research Article

Fast and Accurate Numerical Solution of Allen–Cahn Equation

Yongho Kim,[1] Gilnam Ryu,[2] and Yongho Choi [2,3]

[1]*Faculty of Mathematics, Otto Von Guericke University Magdeburg, Universitatsplatz 2, Magdeburg 39106, Germany*
[2]*Department of IT Convergence Engineering, Daegu University, Gyeongsan-si, Gyeongsangbuk-do 38453, Republic of Korea*
[3]*Department of Mathematics and Big Data, Daegu University, Gyeongsan-si, Gyeongsangbuk-do 38453, Republic of Korea*

Correspondence should be addressed to Yongho Choi; yongho_choi@daegu.ac.kr

Received 20 May 2021; Revised 3 September 2021; Accepted 21 October 2021; Published 6 December 2021

Academic Editor: José M. Domínguez

Simulation speed depends on code structures. Hence, it is crucial how to build a fast algorithm. We solve the Allen–Cahn equation by an explicit finite difference method, so it requires grid calculations implemented by many for-loops in the simulation code. In terms of programming, many for-loops make the simulation speed slow. We propose a model architecture containing a pad and a convolution operation on the Allen–Cahn equation for fast computation while maintaining accuracy. Also, the GPU operation is used to boost up the speed more. In this way, the simulation of other differential equations can be improved. In this paper, various numerical simulations are conducted to confirm that the Allen–Cahn equation follows motion by mean curvature and phase separation in two-dimensional and three-dimensional spaces. Finally, we demonstrate that our algorithm is much faster than an unoptimized code and the CPU operation.

1. Introduction

The Allen–Cahn (AC) equation is a reaction-diffusion equation composed of the reaction term $-F'(\phi(\mathbf{x},t))/\epsilon^2$ and the diffusion term $\Delta\phi(\mathbf{x},t)$:

$$\phi_t(\mathbf{x},t) = -\frac{F'(\phi(\mathbf{x},t))}{\epsilon^2} + \Delta\phi(\mathbf{x},t), \quad \mathbf{x} \in \Omega, t > 0, \tag{1}$$

where $\phi(\mathbf{x},t)$ is the order parameter which is defined as the difference in concentration of the two components in a mixture; $F(\phi)$ is double well potential energy function with minimum values at -1 and 1, and its form is $F(\phi) = 0.25(\phi^2 - 1)^2$. ϵ is the thickness of the transition layer which is a small positive constant value. The AC equation is first introduced in a research on the phase separation of binary iron alloys [1]. The AC equation was studied and applied to various fields such as image inpainting [2–4], image segmentation [5, 6], and crystal growth [7–9]. Also, there are various methods for numerically solving the reaction-diffusion type equations. In [10–12], the authors use the fourth-order exponential time differencing Runge–Kutta method to solve reaction-diffusion type equation as Burgers–Huxley and Gray–Scott model, respectively. There is a study of solving equations using an adaptive method [13–15]. The authors of [16, 17] considered the Fourier spectral method to solve system equations.

With the development of computer hardware such as GPU (graphics processing unit) and memory cards, neural networks are applied in a wide range of research areas such as computer vision, natural language processing, and numerical analysis. As GPU operations outperform CPU (central processing unit) performance in multitasks and high-dimensional problems, open source machine learning libraries such as Pytorch provide a variety of neural networks using GPU and are useful to build the architecture combined with neural networks and numerical methods. Thus, many research studies use machine learning libraries. Raissi et al. [18] proposed physics-informed neural networks combined by multilayer perceptrons and numerical methods to solve nonlinear partial differential equations. Karumuri et al. [19] introduced a solver-free approach for stochastic partial differential equations, and Yang et al. [20] proposed a Bayesian physics-informed neural network. In this paper, we propose a structure using padding and convolution operation for the GPU calculation of the AC equation and

demonstrate the validity of the proposed structure by verifying the result with the Python code that has the same mathematical meaning.

This paper is organized as follows. In Section 2, we present an explicit finite difference method to solve the AC equation which is implemented by CPU and GPU algorithms. In Section 3, the numerical simulations including a motion by mean curvature, phase separation, and temporal evolutions of various initial shapes are introduced as well as the runtime results between CPU and GPU operations are compared. Finally, conclusions are drawn in Section 4.

2. Numerical Solutions

In this section, we present an explicit finite difference method to solve AC equation (1). Also, we give an explanation of each algorithm for CPU and GPU computing. For simplicity of expression, we describe a numerical scheme for the AC equation in two dimensions (2D), and the definition in the three-dimensional (3D) space can be easily extended and considered. A computational domain is defined using a uniform grid of size $h = 1/N_x$ and $\Omega_h = \{(x_i, y_j) = (a + (i - 0.5)h, c + (j - 0.5)h)\}$ for $1 \le i \le N_x$, $1 \le j \le N_y$ is the set of cell centers. Here, N_x and N_y are mesh sizes on computational domain $(a, b) \times (c, d)$. For the definition of the boundary condition, we define the extended computational domain as follows:

$$\Omega_h = \left\{\left(x_i, y_j\right) = (a + (i - 1.5)h, c + (j - 1.5)h)\right\}, \tag{2}$$

for $1 \le i \le N_x + 2$ and $1 \le j \le N_y + 2$. Let ϕ_{ij}^n be approximations of $\phi(x_i, y_j, n\Delta t)$, where $\Delta t = 0.1h^2$ is the temporal step size, T is a final time, and N_t is the total number of time steps. The boundary condition is zero Neumann boundary condition:

$$\begin{aligned} &\phi_{i,1}^n = \phi_{i,2}^n, \quad \phi_{i,N_y+2}^n = \phi_{i,N_y+1}^n, 1 \le i \le N_x, \\ &\phi_{1,j}^n = \phi_{2,j}^n, \quad \phi_{N_x+2,j}^n = \phi_{N_x+1,j}^n, 1 \le j \le N_y. \end{aligned} \tag{3}$$

We define the thickness of the transition layer ϵ in equation (1) as ϵ_m [21]:

$$\epsilon_m = \frac{hm}{2\sqrt{2}\tanh^{-1}(0.9)}, \tag{4}$$

where m is the number of grids representing the thickness.

2.1. Numerical Solutions on CPU (Baseline). The AC equation (1) is discretized using the explicit finite difference method as

$$\begin{aligned} \frac{\phi_{ij}^{n+1} - \phi_{ij}^n}{\Delta t} &= \frac{\phi_{ij}^n - \left(\phi_{ij}^n\right)^3}{\epsilon^2} + \Delta_h \phi_{ij}^n, \\ \phi_{ij}^{n+1} &= \phi_{ij}^n + \Delta t\left(\frac{\phi_{ij}^n - \left(\phi_{ij}^n\right)^3}{\epsilon^2} + \Delta_h \phi_{ij}^n\right), \\ \phi_{ij}^{n+1} &= (1 + \alpha)\phi_{ij}^n - \alpha\left(\phi_{ij}^n\right)^3 + \Delta t \Delta_h \phi_{ij}^n, \end{aligned} \tag{5}$$

where $\alpha = \Delta t/\epsilon^2$ and $\Delta_h \phi_{ij}^n = (\phi_{i-1,j}^n + \phi_{i+1,j}^n + \phi_{i,j-1}^n + \phi_{i,j+1}^n - 4\phi_{ij}^n)/h^2$. The baseline algorithm is implemented in equation (5) using Numpy (CPU array).

2.2. Numerical Solutions on GPU (Pytorch). For GPU computing, the AC equation (1) can be expressed using Pytorch, and the algorithm can be represented as Figure 1.where f is a Pytorch model. The main Algorithm 1 is a set of steps in equation (6) using Pytorch.

$$\begin{aligned} \frac{\phi^{n+1} - \phi^n}{\Delta t} &= \frac{\phi^n - \left(\phi^n\right)^3}{\epsilon^2} + \Delta_h \phi^n, \\ \phi^{n+1} &= \phi^n + \Delta t\left(\frac{\phi^n - \left(\phi^n\right)^3}{\epsilon^2} + \Delta_h \phi^n\right), \\ \phi^{n+1} &= (1 + \alpha)\phi^n - \alpha\left(\phi^n\right)^3 \\ &\quad + \operatorname{conv}\left(\phi^n\right)\left(\alpha = \frac{\Delta t}{\epsilon^2}, \operatorname{conv}\left(\phi^n\right) = \Delta t \Delta_h \phi^n\right), \\ \phi^{n+1} &= f\left(\phi^n\right), \end{aligned} \tag{6}$$

In this Algorithm 1, *nn.ReplicationPad2d* (or *nn.ReplicationPad3d*) is applied to satisfy the boundary condition by padding the ϕ^n input using replication of the input boundary. Also, the convolutional operator *F.conv2d* (or *F.conv3d*) with the 2nd order differencing filter is used to calculate the diffusion term $\Delta\phi^n$. The model can choose an operation mode between GPU and CPU by *to (device)* in Algorithm 1. If device = cuda:0, the model is implemented on GPU, otherwise on CPU. All the codes are available from the first author's GitHub web page (https://github.com/kimy-de/gpuallencahn) and the corresponding author's web page (https://sites.google.com/view/yh-choi/code).

3. Numerical Experiments

In this section, we perform the following numerical tests in 2D and 3D: ϵ_m effect, the motion by mean curvature effect with various initial shapes (circle (sphere in 3D), dumbbell, star, torus, maze), and phase separation. In the ϵ_m effect test, we compare the numerical solutions to analytic values to find a proper ϵ_m value. We simulate a phenomenon that follows motion by mean curvature in various initial shapes and phase separation with random initial condition. We perform the simulation on the following specifications: Intel (R) Core (TM) i9-9900K CPU @3.60 GHz, 32 GB RAM/NVIDIA GeForce RTX 2080 Super.

3.1. Convergence Tests. We perform convergence tests for time and space. We define the discrete L^2 norm as

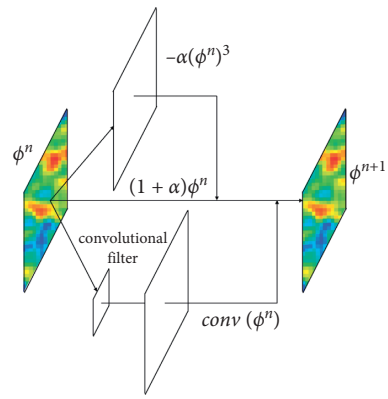


Figure 1: Schematic of the process to obtain f.

```
class Net (nn.Module):
    def __init__(self, h2, dt, eps, device):
        super (Net, self).__init__()
        # 2nd order differencing filter
        self.lap = torch.Tensor ([[[[0., 1., 0.], [1., −4., 1], [0., 1., 0.]]]]).to (device)
        self.pad = nn.ReplicationPad2d (1) #Replication pad for boundary condition.
        self.alpha = dt/eps ** 2
        self.beta = dt/h2
    def forward (self, x):
        u_pad = self.pad (x) #boundary condition
        reaction = F.conv2d (u_pad, self.lap) #reaction term
        x = (1 + self.alpha) * x-self.alpha * x ** 3 + self.beta * reaction
        return x
```

Algorithm 1: Our CNN model for solving AC equation (6).

$$\|e\|_{L^2}^d = \sqrt{h^2 \sum_{i=1}^{N_x} \sum_{j=1}^{N_y} \left(e_{ij}^n\right)^2}, \tag{7}$$

where $e_{ij}^n = \phi_{ij}^n - \phi_{ref}^n$ and ϕ_{ref}^n is reference solution. We use the following initial condition:

$$\phi(x, y, 0) = 0.1 \cos(2\pi x)\cos(2\pi y), \tag{8}$$

with zero Neumann boundary condition on the computational domain $(0, 1) \times (0, 1)$.

First, to show the convergence rate with respect to time, the space step size $h = 0.0025$ (i.e., $N_x = N_y = 400$) is fixed, and we use the reference solution with $N_x = N_y = 400$ and $\Delta t = 0.001h^2$. Then, the test proceeds to time $t = 0.0001$ for time steps $\Delta t = 0.2h^2, 0.1h^2$, and $0.05h^2$. The temporal convergence rate is defined as $\log_2(\|e^{\Delta t}\|_{L^2}^d / \|e^{\Delta t/2}\|_{L^2}^d)$. Table 1 shows that our proposed algorithm has first-order accuracy for time.

Next, we show the spatial accuracy of our proposed algorithm. To show the spatial convergence rate, the time step was fixed to $\Delta t = 0.1h^2$ for each mesh size $h = 0.01, 0.005$, and 0.0025. For the reference solution, the time step was set to $0.001h^2$ for each mesh size. The spatial convergence rate is defined as $\log_2(\|e^h\|_{L^2}^d / \|e^{h/2}\|_{L^2}^d)$. Table 2 shows that our proposed algorithm has between second- and third-order for the spatial accuracy.

3.2. Energy Dissipation and Maximum Principle. AC equation (1) follows the energy dissipation law, and the equation is derived from

$$\mathscr{E}(\phi) = \int_\Omega \left(\frac{F(\phi)}{\epsilon^2} + \frac{1}{2}|\nabla\phi|^2\right) d\mathbf{x}. \tag{9}$$

$\mathscr{E}(\phi)$ is decreasing with time

$$\frac{\partial}{\partial t}\mathscr{E}(\phi) = -\int_\Omega \left|\frac{\partial\phi}{\partial t}\right|^2 d\mathbf{x} \le 0. \tag{10}$$

To check a discrete energy, we can rewrite equation (9) as $\mathscr{E}^d(\phi)$. Figure 2 shows mesh plots and a discrete energy graph when time evolution is performed with random initial condition. The tested initial condition is used as equation (17a) and other parameters are used $N_x = N_y = 200$, space step size $h = 1/N_x$, time step size $\Delta t = 0.1h^2$, and thickness of transition layer ϵ_{10} on the computational domain $\Omega = (0, 1)^2$.

Also, we can obtain that the AC equation does not conserve the initial mass.

$$\begin{aligned} \frac{d}{dt}\int_\Omega \phi d\mathbf{x} &= \int_\Omega \phi_t d\mathbf{x} = \int_\Omega \left(-\frac{F'(\phi)}{\epsilon^2} + \Delta\phi\right) d\mathbf{x} \\ &= -\int_\Omega \frac{F'(\phi)}{\epsilon^2} d\mathbf{x} \le 0, \end{aligned} \tag{11}$$

with zero Neumann boundary condition.

Table 1: Temporal L^2 errors and convergence rates at $t = 0.0001$.

Δt	$0.2\ h^2$		$0.1\ h^2$		$0.05\ h^2$
L^2 error	$9.282e-3$		$4.759e-3$		$2.487e-3$
Rate		0.963		0.936	

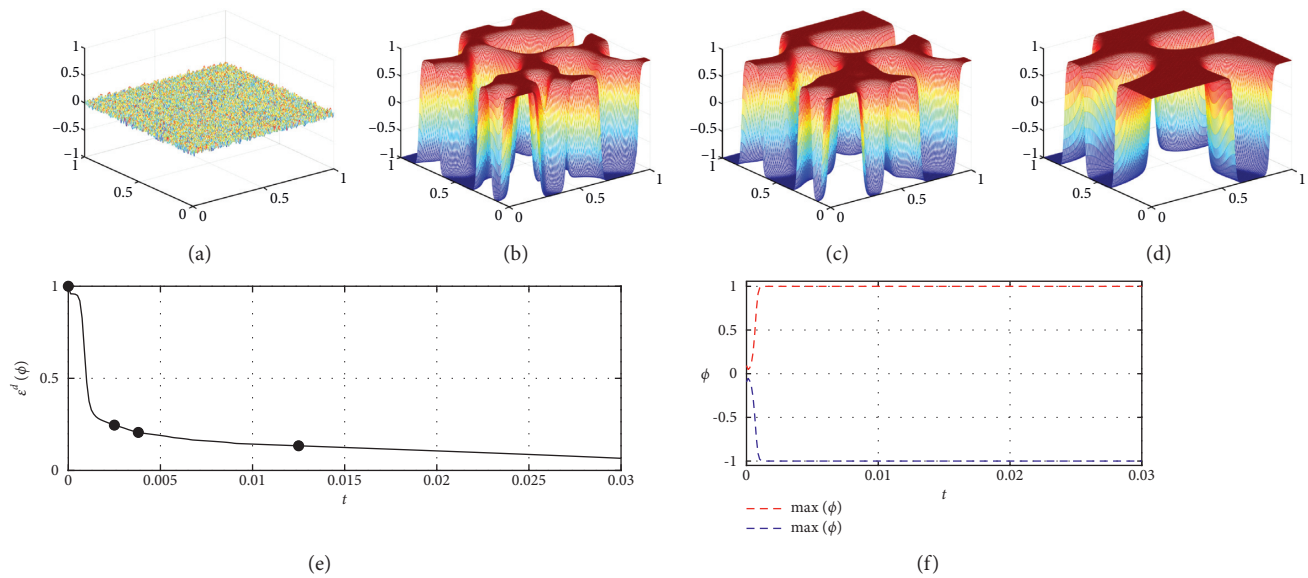


Figure 2: (a–d) Mesh plots at the marked points from left to right of the energy graph (e). (e) Time-dependent normalized discrete total energy $\mathscr{E}^d(\phi(t))/\mathscr{E}^d(\phi(0))$. (f) Maximum and minimum values of ϕ over time evolution.

3.3. Initial Conditions. We consider the 2D and 3D initial conditions introduced in this section: to check the ϵ_m effect, we measure the circle's (sphere in 3D) radius that changes with time and take the initial conditions:

$$t\phi(x, y, 0) = \tanh\left(\frac{R_0 - \sqrt{(x-0.5)^2 + (y-0.5)^2}}{\sqrt{2}\,\epsilon}\right), \quad \text{in 2D}, \tag{12a}$$

$$\phi(x, y, z, 0) = \tanh\left(\frac{R_0 - \sqrt{(x-0.5)^2 + (y-0.5)^2 + (z-0.5)^2}}{\sqrt{2}\,\epsilon}\right), \quad \text{in 3D}, \tag{12b}$$

where R_0 is the initial radius of a circle (sphere in 3D).

In various tests to check the motion by the mean curvature, the initial conditions for the circle and sphere refer to equations (12a) and (12b), respectively. A dumbbell is constructed by following the initial conditions (o/w: otherwise)

$$\phi(x,y,0)=\begin{cases}1.0, & \text{if } (0.4<x<1.6) \text{ and } (0.4<y<0.6),\\ 1+\tanh\left(\dfrac{R_0-\sqrt{(x-0.3)^2+Y}}{\sqrt{2}\,\epsilon}\right)+\tanh\left(\dfrac{R_0-\sqrt{(x-1.7)^2+Y}}{\sqrt{2}\,\epsilon}\right), & \text{otherwise,}\end{cases} \quad \text{in 2D,} \tag{13a}$$

$$\phi(x,y,z,0)=\begin{cases}1.0, & \text{if } (0.4<x<1.6) \text{ and } (0.4<y,z<0.6),\\ 1+\tanh\left(\dfrac{R_0-\sqrt{(x-0.3)^2+YZ}}{\sqrt{2}\,\epsilon}\right)+\tanh\left(\dfrac{R_0-\sqrt{(x-1.7)^2+YZ}}{\sqrt{2}\,\epsilon}\right), & \text{otherwise,}\end{cases} \quad \text{in 3D.} \tag{13b}$$

where R_0 is the initial radius of both sides of dumbbell's circle (sphere in 3D) and for simplicity of expression, $Y=(y-0.5)^2$ and $YZ=(y-0.5)^2+(z-0.5)^2$.

The initial conditions of a star shape are defined as

$$\phi(x,y,0)=\tanh\left(\frac{0.25+0.1\cos(6\theta)-\sqrt{(x-0.5)^2+(y-0.5)^2}}{\sqrt{2}\,\epsilon}\right), \quad \text{in 2D,} \tag{14a}$$

$$\phi(x,y,z,0)=\tanh\left(\frac{0.7+0.2\cos(6\theta)-\sqrt{x^2+y^2+z^2}}{\sqrt{2}\,\epsilon}\right), \quad \text{in 3D,} \tag{14b}$$

where

$$\theta=\begin{cases}\tan^{-1}\left(\dfrac{y-0.5}{x-0.5}\right), & \text{if } (x>0.5),\\ \pi+\tan^{-1}\left(\dfrac{y-0.5}{x-0.5}\right), & \text{otherwise,}\end{cases} \quad \text{in 2D,}$$

$$\theta=\begin{cases}\tan^{-1}\left(\dfrac{z}{x}\right), & \text{if } (x>0.5),\\ \pi+\tan^{-1}\left(\dfrac{z}{x}\right), & \text{otherwise.}\end{cases} \quad \text{in 3D.} \tag{15}$$

and we use different domain sizes in 2D and 3D, so the center of the star depends on the dimensions.

And, a torus shape is given by

$$\phi(x,y,0)=-1+\tanh\left(\frac{R_1-\sqrt{XY}}{\sqrt{2}\,\epsilon}\right)-\tanh\left(\frac{R_2-\sqrt{XY}}{\sqrt{2}\,\epsilon}\right), \quad \text{in 2D,} \tag{16a}$$

$$\phi(x,y,z,0)=\sqrt{z^2+\left(\sqrt{x^2+y^2}-R_1\right)^2}-R_2, \quad \text{in 3D,} \tag{16b}$$

where R_1 and R_2 are the radius of major (outside) and minor (inside) circles, respectively. And, for simplicity of expression, $XY=(x-0.5)^2+(y-0.5)^2$.

The initial conditions of a maze shape are complicated to describe its equation, so refer to the code (https://github.com/kimy-de/gpuallencahn, https://sites.google.com/view/yh-choi/code).

And, the last initial condition is random for confirming the phase separation of the AC equation:

$$\phi(x,y,0)=0.1\,\text{rand}(x,y), \quad \text{in 2D,} \tag{17a}$$

$$\phi(x,y,z,0)=0.1\,\text{rand}(x,y,z), \quad \text{in 3D.} \tag{17b}$$

Here, the function rand (x,y) has a random value between -1 and 1.

3.4. Simulations in the 2-Dimensional Space. Unless otherwise stated, we use the following parameters: mesh size $N_x=N_y=200$, space step size $h=1/N_x$, time step size $\Delta t=0.1h^2$, and computational domain $\Omega=(0,1)\times(0,1)$. We should find a proper thickness of the transition layer as defined in equation (4). First, we find an appropriate ϵ_m by comparing the numerical solution with the exact solution for the radius that decreases due to the motion by mean curvature in the circle initial condition in equation (12a).

Figures 3(a)–3(d) show a circle shrinking with motion by mean curvature based on ϵ_{10}. The exact solution of the radius r decreasing with time evolution can be calculated [22]:

$$r(t)=\sqrt{r_0^2+2(1-d)t}, \tag{18}$$

where r_0 is the initial radius of the circle, d is the dimension, and t is time. We simulate various ϵ_m until the final time $T=0.03$. As shown in Figure 3, when ϵ_{10} is used, it is found that the exact solution and the numerical solution are most similar in the experiment. Therefore, the other tests are

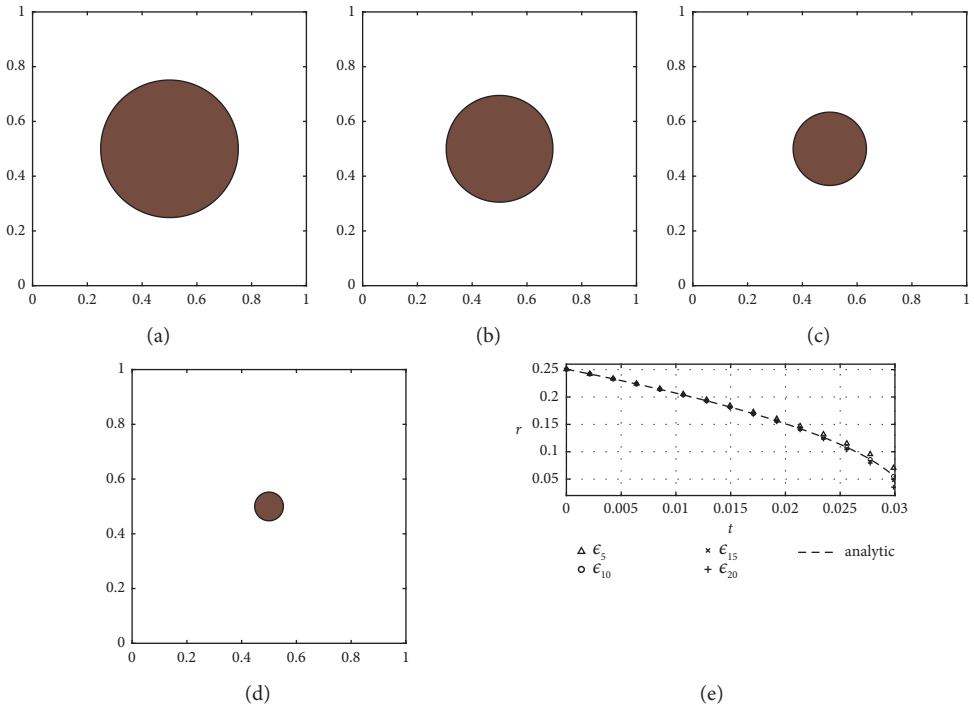


Figure 3: (a–d) The changes in the circle over time are shown, and each time is described in the figures. (e) Change of radius for various ϵ_m.

performed using ϵ_{10} except for the case of changing the grid in 2D.

Figure 4 shows the temporal evolution of the dumbbell shape, and the initial condition is shown in equation (13a). For resolution, we use $N_x = 400$ and $N_y = 200$ on the computational domain $\Omega = (0, 2) \times (0, 1)$. The final time T of the simulation is 0.0094 and $R_0 = 0.2$. In Figure 4(e), the changing direction by the motion by mean curvature is indicated by arrows.

Figure 5 shows the evolution of the star shape created by equation (14a). The parameters are used as mentioned at the beginning of this section and $T = 0.0325$. As shown in Figure 5, the tips of the star move inward and the gaps between the tips move outward. When it changes to the shape of a circle, the change of the radius can be predicted as shown in Figure 3.

Figure 6 shows the evolution of the torus shape of equation (16a) with $T = 0.0575$, $R_1 = 0.4$, and $R_2 = 0.3$. Because the inner circle has a larger curvature, it shrinks faster than the outer circle, and after the inner circle disappears, the change of radius over time can be measured as shown in Figure 3.

Figure 7 shows the evolution of a maze shape. We use $N_x = N_y = 100$, ϵ_5, and $T = 0.04$. As shown in Figure 7, we obtain the results of shrinking while maintaining its initial shape.

The last simulation in 2D is phase separation with a random initial condition equation (17a). In Figure 8, starting with random values with 0.1 amplitude, but over time, phase separation occurs with values -1 to 1.

According to the results in Table 3, the speed gap between CPU and GPU is significant. In 2D, it is up to 251.6 times the difference between the Python:CPU and the Pytorch:GPU codes. Also, Pytorch:GPU tensors make the model up to 4.73 times faster than Pytorch:CPU tensors in the same code.

3.5. Simulations in the 3-Dimensional Space. Three-dimensional simulations can be considered as an extension of two-dimensional tests. Unless otherwise stated, we use the following parameters: mesh size $N_x = N_y = N_z = 100$, space step size $h = 1/N_x$, time step size $\Delta t = 0.1h^2$, and computational domain $\Omega = (0, 1) \times (0, 1) \times (0, 1)$. As in the 2D simulation, we find an appropriate ϵ_m by comparing the numerical solution with the exact solution (using in equation (18)) for the radius of the sphere initial condition in equation (12b).

Figures 9(a)–9(d) show the time evolution results for ϵ_{12}, and (e) presents the comparison of the numerical and exact solutions of radius for various epsilons. As shown in Figure 9, when ϵ_{12} is used, it is found that the exact solution and the numerical solution are most similar. Therefore, the other tests are performed using ϵ_{12}.

Figure 10 shows the temporal evolution of the dumbbell shape, and the initial condition is shown in equation (13b). For resolution, we use $N_x = 200$ and $N_y = N_z = 100$ on the computational domain $\Omega = (0, 2) \times (0, 1) \times (0, 1)$. The final time T of the simulation is $T = 0.0025$ and $R_0 = 0.25$. The tendency of the 3D dumbbell motion is different from the result of 2D dumbbell. The reason is that, in the initial shape in Figure 10(a), the radius of the handle is much smaller than the radius of the spheres at both ends. Therefore, the handle part shrinks faster than the end of spheres and breaks as in Figure 10.

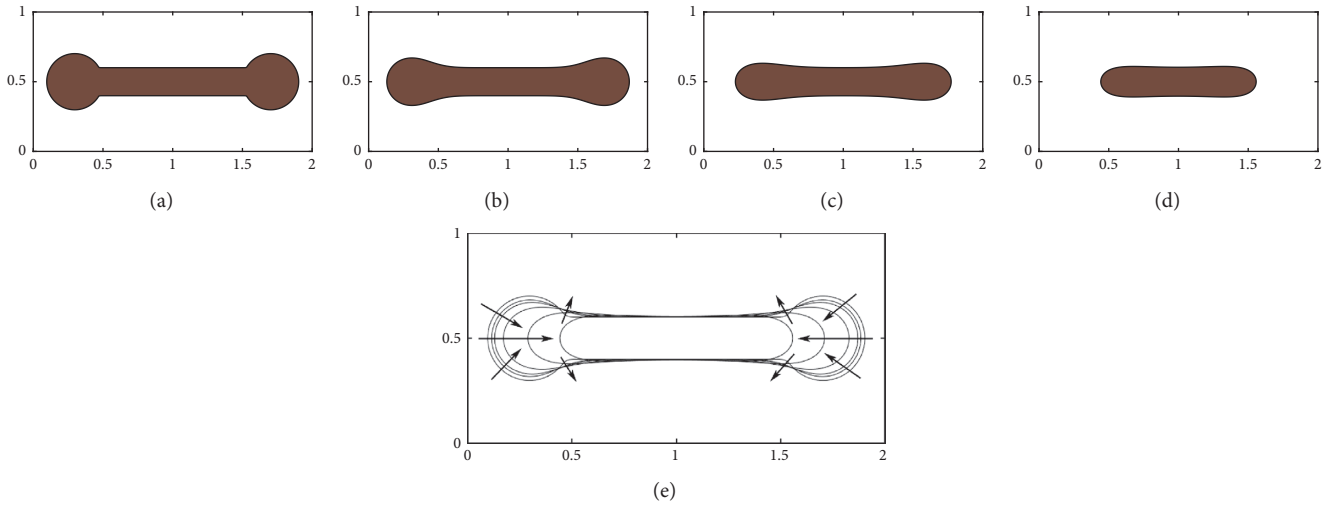


Figure 4: (a–d) Time evolution of dumbbell shape, and each time is described in the figures. (e) Contour lines over time are shown by overlapping. Changing by the mean curvature flow can be seen.

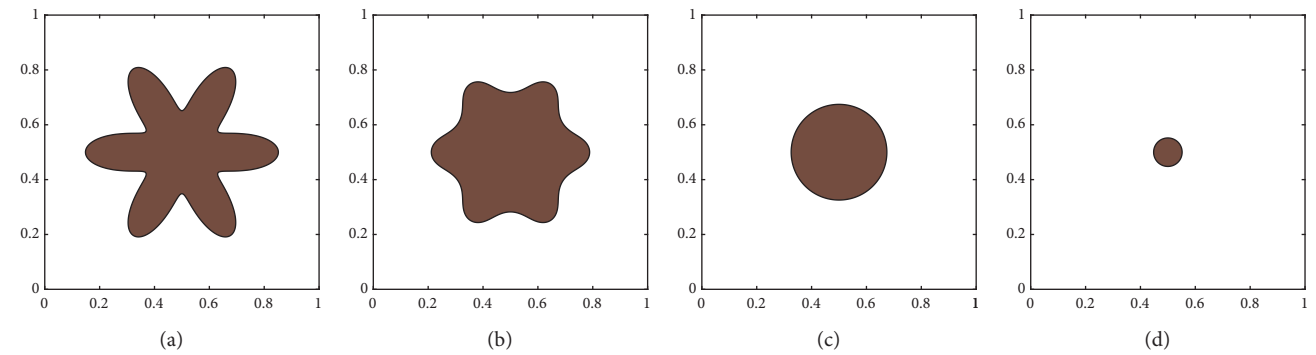


Figure 5: Time evolution of a star shape. As in the shape of a dumbbell, it can be seen that it changes with the mean curvature flow. (a) $t = 0$, (b) $t = 0.0025$, (c) $t = 0.0188$, (d) $t = 0.0325$.

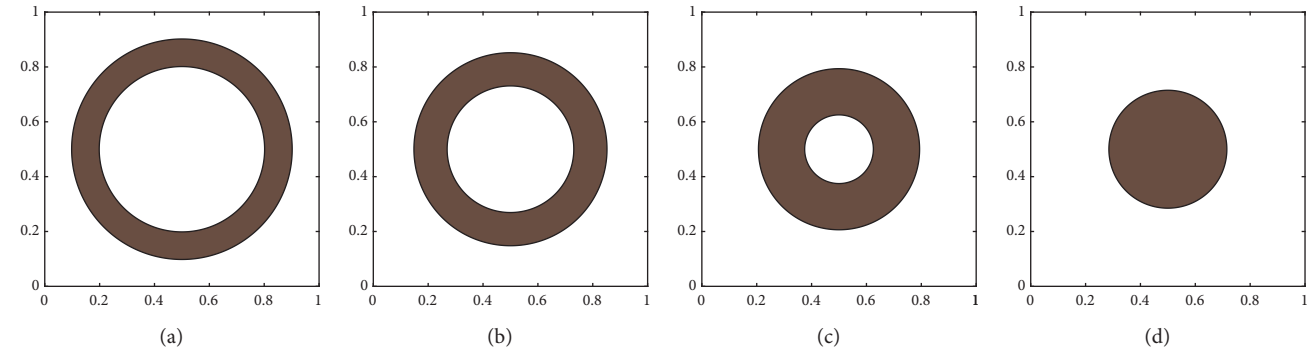


Figure 6: Time evolution of a torus shape, and each time is described in the figures. The inner circle has larger curvature than the outer circle, so the inner circle shrinks faster than the outer circle. (a) $t = 0$, (b) $t = 0.0188$, (c) $t = 0.0375$, (d) $t = 0.0575$.

Figure 11 shows the evolution of the star shape shown in equation (14b) on the computational domain $\Omega = (-1, 1) \times (-1, 1) \times (-1, 1)$. The parameters are used as mentioned at the beginning of this section and $T = 0.02$. As shown in Figure 11, similar to the 2D result, the tips of the star move inward, and the gaps between the tips move outward.

Figure 12 shows the evolution of the torus shape with initial condition in equation (16b) using $R_1 = 0.3$ and $R_2 = 0.3$ and $T = 0.01$ on the computational domain

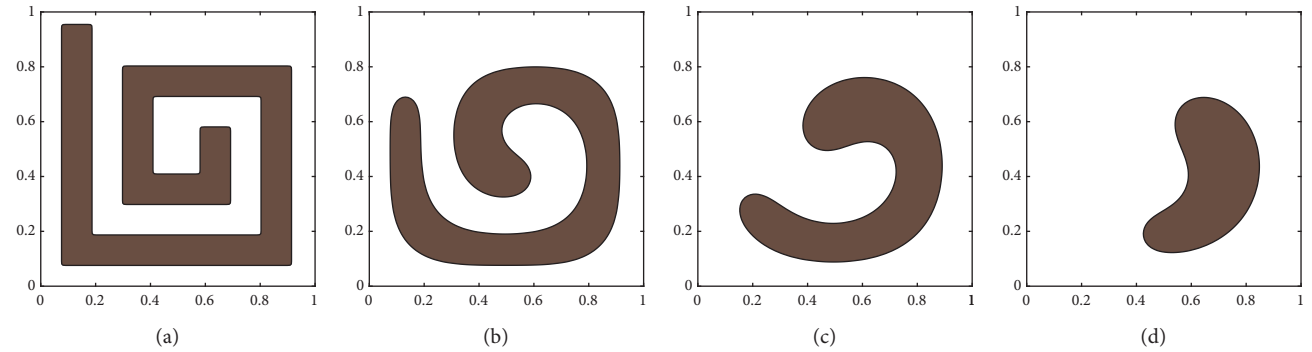


Figure 7: Time evolution of a maze shape on 100×100 mesh size with ϵ_5, and $T = 0.04$. (a) $t = 0$, (b) $t = 0.01$, (c) $t = 0.025$, (d) $t = 0.04$.

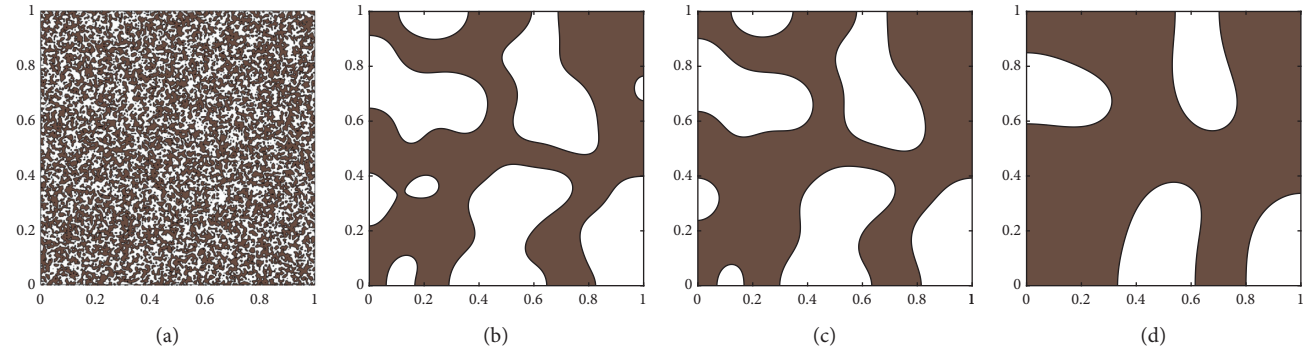


Figure 8: Time evolution of phase separation with a random initial condition. (a) $t = 0$, (b) $t = 0.0025$, (c) $t = 0.0038$, (d) $t = 0.0125$.

Table 2: Spatial L^2 errors and convergence rates at $t = 0.0001$.

h	0.01	0.005	0.0025
L^2 error	$1.011e-4$	$6.722e-04$	$4.760e-3$
Rate	2.733	2.824	

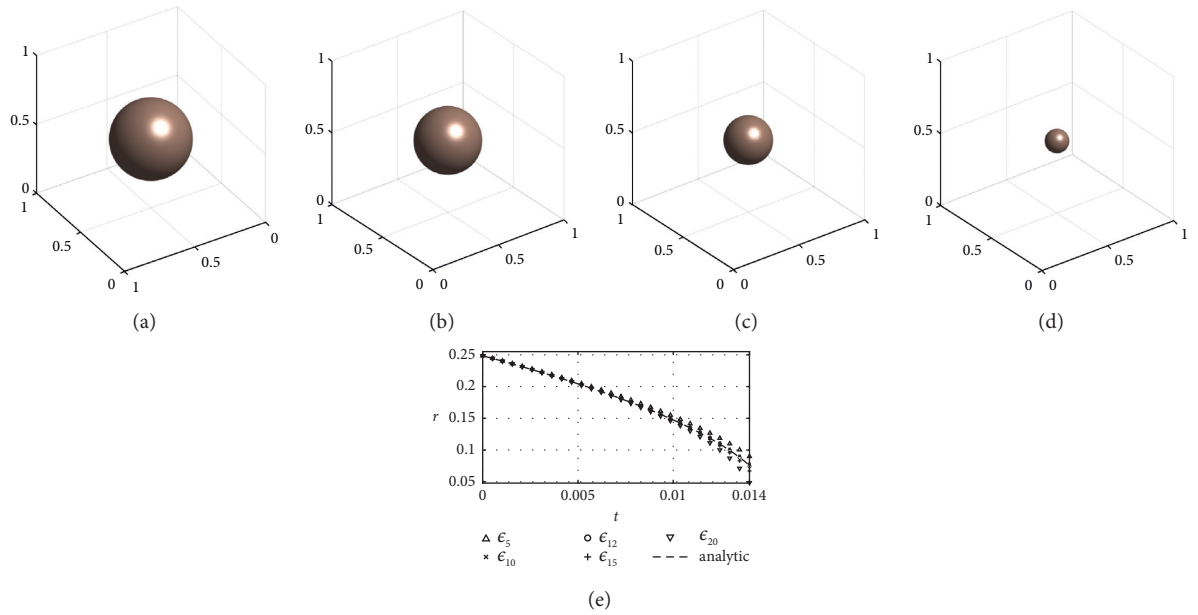


Figure 9: (a–d) The changes in the circle over time are shown, and each time is described in the figures. (e) Change of radius for various ϵ_m.

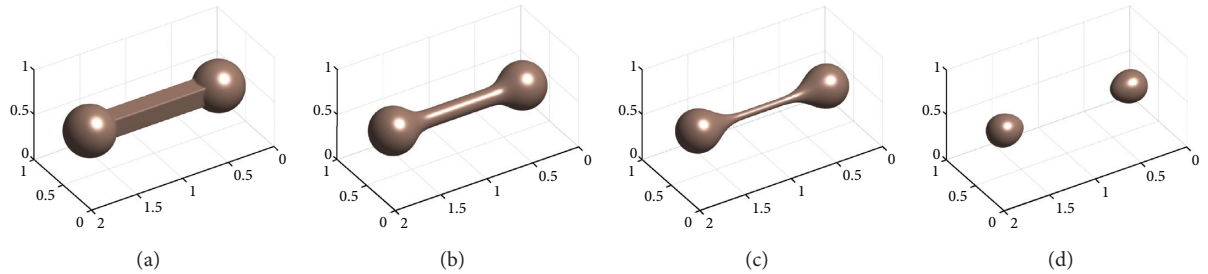


Figure 10: (a–d) Time evolution of the dumbbell shape. The handle shrinks quickly and breaks occur because the curvature of the handle is larger than that of both spheres. (a) $t = 0$, (b) $t = 0.0006$, (c) $t = 0.0013$, (d) $t = 0.0025$.

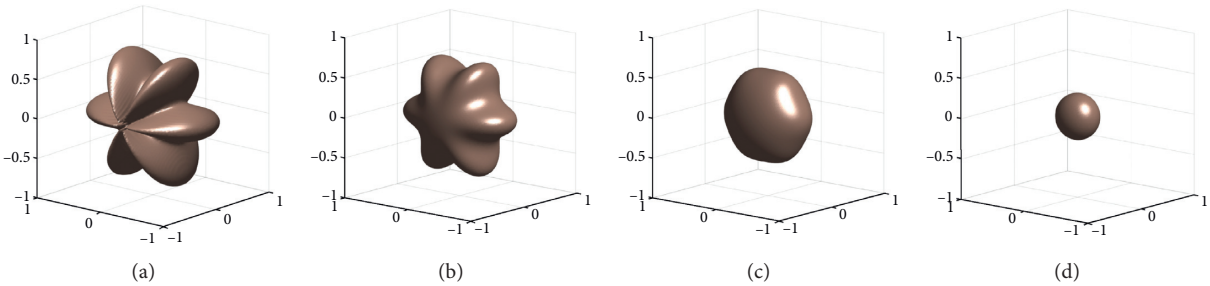


Figure 11: Time evolution of a star shape. The shape changes with the mean curvature flow. (a) $t = 0$, (b) $t = 0.0025$, (c) $t = 0.0075$, (d) $t = 0.02$.

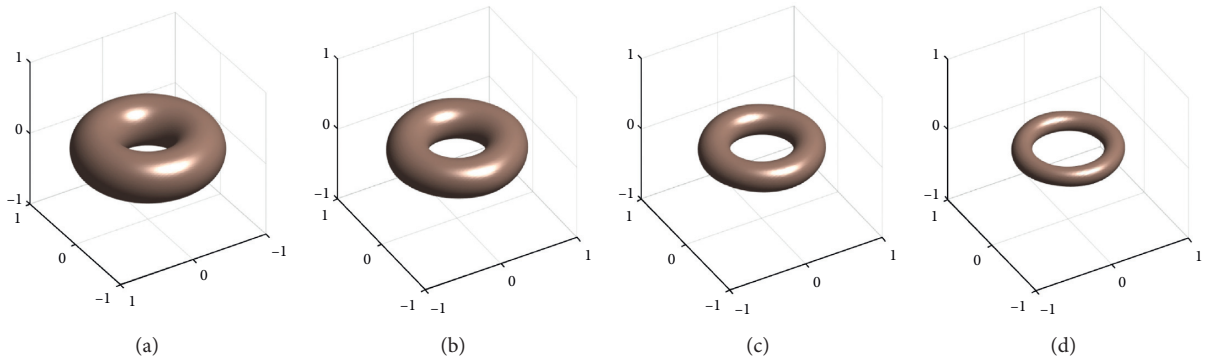


Figure 12: Time evolution of a torus shape. Contrary to the 2D results, the inner circle increases due to the effect of the mean curvature. (a) $t = 0$, (b) $t = 0.005$, (c) $t = 0.008$, (d) $t = 0.01$.

$\Omega = (-1, 1) \times (-1, 1) \times (-1, 1)$. Contrary to the 2D result, the inner circle becomes larger because the radius of the minor circle exists. Therefore, the radius of the major circle and the minor circle each has different curvature. Since the radius of the minor circle is smaller than major circle, the mean curvature drives the motion into the inside of the torus as shown in Figure 12.

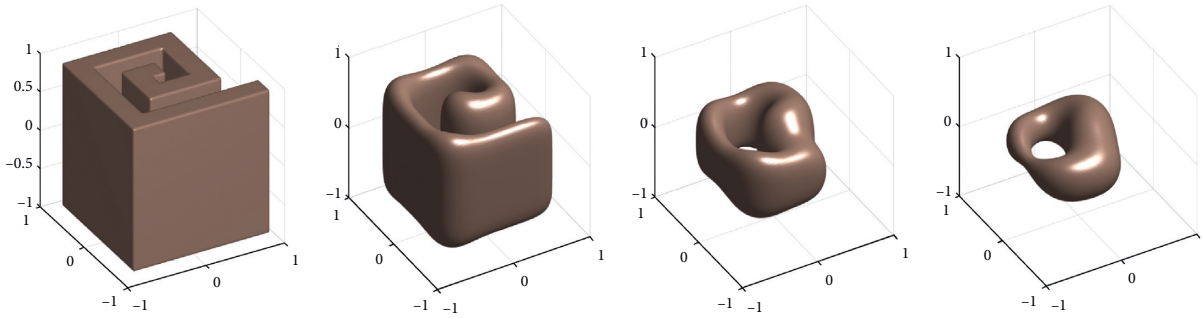
Figure 13 shows the evolution of a maze shape. The initial condition is described in code (https://github.com/kimy-de/gpuallencahn, https://sites.google.com/view/yh-choi/code). In this simulation, we use $T = 0.0175$ and computational domain $\Omega = (-1, 1) \times (-1, 1) \times (-1, 1)$. As shown in Figure 13, we obtain the results of shrinking while maintaining its initial shape in Figure 13(b), and then merges and shrinks in Figures 13(c) and 13(d). If we use different ϵ_m, it can shrink while preserving its initial shape.

The last simulation in 3D is phase separation with random initial condition in equation (17b). In Figure 14, starting with random values with 0.1 amplitude, but over time, phase separation occurs with values −1 to 1.

To estimate the error of the CPU and GPU codes, we use the following defined Err:

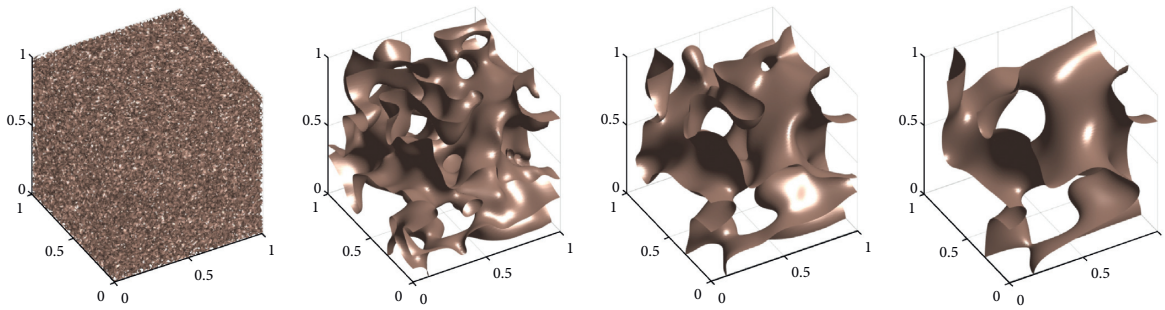
$$\text{Err} = \frac{1}{N} \sum_{t=1}^{N} \sqrt{\text{avg}\left(\left(\phi_{gpu}^{t} - \phi_{cpu}^{t} \right)^2 \right)}, \tag{19}$$

where N is the total number of iterations, $\text{avg}(X)$ is the average of elements in an array X, and t means evolution time (i.e., $t = n\Delta t$, $n = 1, 2, \ldots, N$). We show the error



(a) (b) (c) (d)

Figure 13: Time evolution of a maze shape on $\Omega = (-1, 1) \times (-1, 1) \times (-1, 1)$, and $T = 0.0175$. (a) $t = 0$, (b) $t = 0.005$, (c) $t = 0.0125$, (d) $t = 0.0175$.



(a) (b) (c) (d)

Figure 14: Time evolution of phase separation with a random initial condition. (a) $t = 0$, (b) $t = 0.0025$, (c) $t = 0.005$, (d) $t = 0.01$.

Table 3: Runtime Result in 2D(sec).

Iterations	Initial value					
	Circle 12001	Dumbbell 15001	Star 13001	Torus 23001	Maze 4001	Random 12001
CPU:Python	680.09 (185.31)	1736.05 (251.6)	729.74 (128.47)	1326.82 (165.65)	60.15 (52.3)	674.63 (119.62)
CPU:Pytorch	14.73 (4.01)	30.67 (4.44)	17.60 (3.1)	37.85 (4.73)	2.98 (2.59)	17.79 (3.15)
GPU:Pytorch	3.67	6.90	5.68	8.01	1.15	5.64

The values in parentheses describe how many times the difference is based on GPU:Pytorch time for each test.

Table 4: Errors of various numerical simulations with the baseline and ours.

Dimension	Initial value					
	Random	Dumbbell	Circle/sphere	Maze	Star	Torus
2D	1.75×10^{-6}	7.03×10^{-7}	5.51×10^{-7}	2.22×10^{-7}	6.55×10^{-7}	7.52×10^{-7}
3D	3.01×10^{-6}	1.20×10^{-6}	1.11×10^{-6}	3.36×10^{-6}	1.56×10^{-6}	1.91×10^{-6}

Table 5: Runtime result in 3D (sec).

Iterations	Initial value					
	Sphere 2001	Dumbbell 2001	Star 2001	Torus 1201	Maze 2401	Random 2001
CPU:Python	4022.38 (3944)	8050.45 (4087)	4014.83 (4015)	2478.22 (4766)	4844.24 (3814)	4163.17 (4042)
CPU:Pytorch	65.95 (65)	79.23 (40)	65.22 (65)	39.56 (76)	78.72 (62)	65.80 (64)
GPU:Pytorch	1.02	1.97	1.00	0.52	1.27	1.03

The values in parentheses describe how many times the difference is based on GPU:Pytorch time for each test.

results obtained by equation (19) to check the difference between the numerical results of CPU:Python (baseline) and GPU:Pytorch. In Table 4, all the errors for any cases are less than $1.0e-6$.

According to the results in Table 5, the speed gap between CPU and GPU operations is significant. Of course, it will be faster by performing GPU calculations, but we proposed a structure using padding and convolution operation in performing GPU calculations on AC equations. And, the results are demonstrated by verifying the results (Table 4) with Python code which has the same mathematical meaning. In 3D, the GPU performance is much more overwhelming than the CPUs. For instance, the GPU code is 4766 times faster than the Python code in the torus problem. Also, GPU tensors make the model up to 76 times faster than CPU tensors in the same code. The values in parentheses describe how many times the difference is based on GPU:Pytorch time for each test.

4. Conclusions

In this paper, we proposed a structure using padding and convolution operation for the GPU calculation of the Allen–Cahn equation. We increased the simulation speed and demonstrated the validity of the proposed structure by verifying the result with the Python code that has the same mathematical meaning. We solved the Allen–Cahn equation by the explicit finite difference method and compared the runtime results between CPU and GPU algorithms. The errors of CPU:Python and GPU:Pytorch are less than $1.0e-6$ for the given initial conditions. Also, we showed that our GPU code is up to 251.6 and 4765.81 times faster than the CPU codes in 2D and 3D, respectively. By showing these results, accuracy and efficiency have been demonstrated. Various numerical simulations were presented to confirm that the Allen–Cahn equation follows the motion by mean curvature and phase separation in the 2D and 3D space. In this way, we can build a fast algorithm for any differential equations using the finite difference method by efficient programmatic code structures.

Data Availability

All the codes are available from the first author's GitHub web page (https://github.com/kimy-de/gpuallencahn) and the corresponding author's web page (https://sites.google.com/view/yh-choi/code).

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the Daegu University Research Grant, 2019 (to Y. Choi).

References

[1] S. M. Allen and J. W. Cahn, "A microscopic theory for antiphase boundary motion and its application to antiphase domain coarsening," *Acta Metallurgica*, vol. 27, no. 6, pp. 1085–1095, 1979.

[2] Y. Li, D. Jeong, J.-i. Choi, S. Lee, and J. Kim, "Fast local image inpainting based on the Allen-Cahn model," *Digital Signal Processing*, vol. 37, pp. 65–74, 2015.

[3] A. L. Brkić and A. Novak, "A nonlocal image inpainting problem using the linear Allen–Cahn equation," in *Proceedings of the Conference on Non-integer Order Calculus and its Applications*, Springer, ŁÓDŹ, Poland, October 2018.

[4] Z. Feng, J. Yin, and J. Zhou, "Inpainting algorithm for Jacquared image based on phase-field model," in *Proceedings of the 2008 3rd International Conference on Intelligent System and Knowledge Engineering*, vol. 1, November 2008.

[5] M. Beneš, V. Chalupecký, and K. Mikula, "Geometrical image segmentation by the Allen–Cahn equation," *Applied Numerical Mathematics*, vol. 51, no. 2-3, pp. 187–205, 2004.

[6] D. A. Kay and A. Tomasi, "Color image segmentation by the vector-valued Allen-Cahn phase-field model: a multigrid solution," *IEEE Transactions on Image Processing*, vol. 18, no. 10, pp. 2330–2339, 2009.

[7] J. Zhang, C. Chen, and X. Yang, "A novel decoupled and stable scheme for an anisotropic phase-field dendritic crystal growth model," *Applied Mathematics Letters*, vol. 95, pp. 122–129, 2019.

[8] X. Yang, "Efficient linear, stabilized, second-order time marching schemes for an anisotropic phase field dendritic crystal growth model," *Computer Methods in Applied Mechanics and Engineering*, vol. 347, pp. 316–339, 2019.

[9] X. Jing and Q. Wang, "Linear second order energy stable schemes for phase field crystal growth models with nonlocal constraints," *Computers & Mathematics with Applications*, vol. 79, no. 3, pp. 764–788, 2020.

[10] K. M. Furati, M. Yousuf, and A. Q. M. Khaliq, "Fourth-order methods for space fractional reaction–diffusion equations with non-smooth data," *International Journal of Computer Mathematics*, vol. 95, no. 6-7, pp. 1240–1256, 2018.

[11] K. M. Owolabi, "Numerical solution of the generalized Burgers-Huxley equation by exponential time differencing scheme," *International Journal of Biomedical Engineering and Science*, vol. 1, pp. 43–52, 2015.

[12] K. M. Owolabi and K. C. Patidar, "Numerical solution of singular patterns in one-dimensional Gray-Scott-like models," *International Journal of Nonlinear Sciences and Numerical Stimulation*, vol. 15, no. 7-8, pp. 437–462, 2014.

[13] L. Ferm, A. Hellander, and P. Lötstedt, "An adaptive algorithm for simulation of stochastic reaction-diffusion processes," *Journal of Computational Physics*, vol. 229, no. 2, pp. 343–360, 2010.

[14] K. M. Owolabi and K. C. Patidar, "Solution of pattern waves for diffusive Fisher-like non-linear equations with adaptive methods," *International Journal of Nonlinear Sciences and Numerical Stimulation*, vol. 17, no. 6, pp. 291–304, 2016.

[15] K. M. Owolabi and K. C. Patidar, "Numerical simulations of multicomponent ecological models with adaptive methods," *Theoretical Biology and Medical Modelling*, vol. 13, no. 1, pp. 1–25, 2016.

[16] K. M. Owolabi and A. Atangana, "Mathematical analysis and numerical simulation of two-component system with non-integer-order derivative in high dimensions," *Advances in Difference Equations*, vol. 2017, no. 1, 24 pages, 2017.

[17] H. G. Lee, "A second-order operator splitting Fourier spectral method for fractional-in-space reaction-diffusion equations," *Journal of Computational and Applied Mathematics*, vol. 333, pp. 395–403, 2018.

[18] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[19] S. Karumuri, R. Tripathy, I. Bilionis, and J. Panchal, "Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks," *Journal of Computational Physics*, vol. 404, p. 109120, 2020.

[20] L. Yang, X. Meng, and G. E. Karniadakis, "B-PINNs: Bayesian physics-informed neural networks for forward and inverse PDE problems with noisy data," *Journal of Computational Physics*, vol. 425, p. 109913, 2021.

[21] J.-W. Choi, H. G. Lee, D. Jeong, and J. Kim, "An unconditionally gradient stable numerical method for solving the Allen-Cahn equation," *Physica A: Statistical Mechanics and its Applications*, vol. 388, no. 9, pp. 1791–1803, 2009.

[22] Y. Li, H. G. Lee, D. Jeong, and J. Kim, "An unconditionally stable hybrid numerical method for solving the Allen-Cahn equation," *Computers & Mathematics with Applications*, vol. 60, no. 6, pp. 1591–1606, 2010.