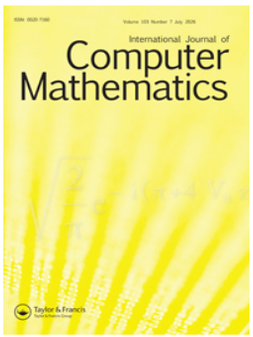


International Journal of Computer Mathematics

ISSN: 0020-7160 (Print) 1029-0265 (Online) Journal homepage: www.tandfonline.com/journals/gcom20

A cell structure implementation of the multigrid method for the three-dimensional diffusion equation

Youngjin Hwang , Soobin Kwak , Jyoti , Hyunho Shin , Xinpei Wu & Junseok Kim

To cite this article: Youngjin Hwang , Soobin Kwak , Jyoti , Hyunho Shin , Xinpei Wu & Junseok Kim (30 Apr 2026): A cell structure implementation of the multigrid method for the three-dimensional diffusion equation, International Journal of Computer Mathematics, DOI: 10.1080/00207160.2026.2664005

To link to this article: https://doi.org/10.1080/00207160.2026.2664005

INTERNATIONAL JOURNAL OF COMPUTER MATHEMATICS
https://doi.org/10.1080/00207160.2026.2664005

Taylor & Francis
by informa

RESEARCH ARTICLE

A cell structure implementation of the multigrid method for the three-dimensional diffusion equation

Youngjin Hwang[a,b], Soobin Kwak[a], Jyoti[c], Hyunho Shin[a], Xinpei Wu[a] and Junseok Kim[a]

[a]Department of Mathematics, Korea University, Seoul, Republic of Korea; [b]Department of Mathematics, Chonnam National University, Gwangju, Republic of Korea; [c]The Institute of Basic Science, Korea University, Seoul, Republic of Korea

ABSTRACT

We present a non-recursive implementation of the multigrid method for the three-dimensional diffusion equation discretized on cell-centred grids. The multigrid method is a well-established numerical method with solid mathematical foundations for solving large-scale linear and nonlinear systems efficiently. Recursive algorithms are relatively easy to understand because of their natural hierarchical structure. However, it is difficult to debug an incorrect implementation of a recursive algorithm at a specific grid level. We present a non-recursive multigrid algorithm that combines clarity, ease of development and debugging, and high computational efficiency by taking advantage of recent features of programming languages and their standard libraries for handling arrays with complex structures such as cell arrays. The method preserves the efficiency of standard multigrid solvers and has a structure that is easy to use. This non-recursive, level-wise organization works across programming environments. It helps researchers understand multigrid methods more clearly and supports both learning and research.

ARTICLE HISTORY
Received 16 October 2025
Revised 10 April 2026
Accepted 18 April 2026

KEYWORDS
Non-recursive multigrid method; diffusion equation; cell array structure; finite difference method; explicit level-wise management

2020 MATHEMATICS SUBJECT CLASSIFICATIONS
65N55; 65N06; 65M55; 35K05

1. Introduction

In this paper, we introduce a non-recursive formulation for the complex nature and challenges associated with the implementation of three-dimensional (3D) multigrid algorithms. We also provide a MATLAB programme in which level-wise data are stored in cell arrays. This approach provides a clear and simplified description of the multigrid algorithm, and the same level-wise organization can be adopted in other programming environments. Although multigrid algorithms are extensively used to solve large-scale linear systems with high efficiency, their application and implementation frequently involve intricate structures and considerable difficulties. We consider the diffusion equation with Neumann boundary conditions on $\Omega = (L_1, R_1) \times (L_2, R_2) \times (L_3, R_3)$ in three-dimensional space:

$$\frac{\partial u}{\partial t}(\mathbf{x}, t) = D\Delta u(\mathbf{x}, t), \quad \mathbf{x} \in \Omega,\ t > 0, \tag{1}$$

$$\mathbf{n} \cdot \nabla u(\mathbf{x}, t) = 0, \quad \mathbf{x} \in \partial\Omega,\ t \geq 0, \tag{2}$$

CONTACT Junseok Kim cfdkim@korea.ac.kr Department of Mathematics, Korea University, Seoul 02841, Republic of Korea

$$u(\mathbf{x}, 0) = u_0(\mathbf{x}), \quad \mathbf{x} \in \Omega, \tag{3}$$

where $u(\mathbf{x}, t)$ is the concentration at space $\mathbf{x}$ and time t, $u_0(\mathbf{x})$ is the initial condition, D is the diffusion coefficient, and $\mathbf{n}$ is the outward unit normal vector on the boundary $\partial\Omega$. For simplicity, we set $D = 1$ throughout this paper. The diffusion equation arises in numerous physical and engineering applications [11], including heat transfer [24], fluid dynamics [1,10,25], conservative phase transformation [17], biological transport phenomena [7,9,12], motion by mean curvature [23], and image processing [19]. Efficient numerical methods are essential for solving such partial differential equations (PDEs), especially in 3D domains where computational complexity increases significantly [15].

The multigrid method is a widely used numerical method for solving large-scale linear and nonlinear systems, particularly those arising from partial differential equations [2]. For instance, it is crucial in computational fluid dynamics for solving complex flow problems [13,27], in structural and mechanical engineering for analyzing stress and deformation [28,35,36], and in computer graphics and vision for tasks like image reconstruction and mesh processing [4,20,33]. This method operates on multiple grid levels, where corrections are computed at multiple resolutions, from coarse grids to fine grids [32]. This is achieved by using a combination of smoothing, restriction, and prolongation operations, which accelerates the convergence of iterative solvers. Unlike classical methods such as Gauss–Seidel and Jacobi iterations, which suffer from slow convergence for high-resolution problems [22], the multigrid approach significantly improves efficiency by rapidly eliminating both high- and low-frequency error components [3,34]. Previous studies have also shown the feasibility of applying the multigrid method in numerical experiments. Chen et al. [5] proposed a meta-learning-based multigrid method, the Meta-MgNet, to efficiently solve parameterized partial differential equations and demonstrated its efficiency through numerical experiments. Jiang and Bai [18] developed two efficient multigrid methods for solving time-fractional conservation laws. Their approaches were designed to accelerate the convergence of nonlinear spatial discrete systems arising from implicit finite-volume TVD schemes using global and local Lax–Friedrichs fluxes. Numerical experiments demonstrated that both methods exhibit good convergence for relatively large fractional orders, while the second multigrid method maintains stable convergence even as the fractional order approaches zero. Trofimov [31] investigated inverse problems for a layered elastic structure on an elastic foundation and developed a gradient computation method using the Adjoint State Method, in which the multigrid method plays a central role in efficiently solving the forward and adjoint problems. Dünnebacke et al. [8] devised a time-simultaneous multigrid method for efficiently solving large-scale partial differential equations and validated its effectiveness through numerical experiments. Ciaramella et al. [6] developed a collective multigrid algorithm. They tested the method on various problems and showed that it effectively solves large saddle-point systems arising in PDE-constrained optimization under uncertainty. Sondak et al. [29] developed a variational multiscale-based large eddy simulation method for Rayleigh–Bénard convection at high Rayleigh numbers, in which a fully-coupled algebraic multigrid method was employed to efficiently solve the large sparse linear systems arising from the finite element discretization. Huang et al. [16] optimized multigrid smoothers by training convolutional neural networks on small partial differential equation problems and applied these smoothers to solve larger problems more efficiently. Liu et al. [26] formulated a nonlinear multigrid method for parameter identification in constrained partial differential equations with and confirmed its effectiveness. The method reduced computational costs, suppressed noise, and improved inversion accuracy in simulations of porosity identification in fluid-saturated porous media. However, traditional implementations of the multigrid method frequently rely on recursive structures. Although these structures are mathematically elegant, they introduce computational challenges such as increased memory overhead, make it difficult to track errors at specific levels, and reduce accessibility for users who are unfamiliar with recursive algorithms [14,30].

To resolve these challenges, we propose a non-recursive implementation of the multigrid method for solving the 3D diffusion equation. Although recursive algorithms are relatively easy to

understand because of their natural hierarchical structure, debugging an incorrect recursive implementation at a specific grid level is difficult. Recent features of programming languages and their standard libraries for handling arrays with complex structures such as cell arrays make it possible to construct a non-recursive multigrid algorithm with clarity, ease of development and debugging, and high computational efficiency. The proposed approach removes recursion and manages level-dependent data systematically through explicit level structures across multigrid levels. As an illustration, we provide a MATLAB example in which cell arrays store the level-wise variables, but the non-recursive level management is independent of the programming language. This level-wise non-recursive structure makes the algorithm easier to follow and debug, while preserving the efficiency of standard multigrid solvers.

This article is organized as follows: Section 2 describes the non-recursive formulation, presents a reference implementation, and discusses the benefits compared with standard recursive multigrid implementations. Section 3 presents numerical experiments that demonstrate the convergence of the algorithm and illustrate the dynamic behaviour of the 3D diffusion equation. Finally, Section 4 concludes the findings and discusses potential extensions of this work.

2. Numerical solution algorithm

In this section, we describe a simple non-recursive implementation of the multigrid algorithm and present a MATLAB example. The same level-wise data management can be implemented in other programming environments as well. Before the algorithm is described in detail, we first introduce the concept and usage of the cell structure in MATLAB, which serves as a convenient data container in the proposed implementation. In MATLAB, a cell array is a flexible data type that allows the storage of elements with different types and sizes. Unlike regular arrays, which require uniform data types and consistent dimensions, cell arrays can contain scalars, vectors, matrices, or even other cell arrays. This property is especially advantageous in the multigrid method, where the solution arrays at each grid level typically have different sizes. The following provides a simple example of the cell structure in MATLAB.

```
>> u{1}=4*ones(4,4,4); u{2}=2*ones(2,2,2);
>> u
u = 1x2 cell array
    {4x4x4 double} {2x2x2 double}
>> u{2}
ans(:,:,1) =
     2  2


     2  2
ans(:,:,2) =
     2  2
     2  2
>> u{1}(2,3,4) = 1;
>> u{1}
ans(:,:,1) =
     4  4  4  4
     4  4  4  4
     4  4  4  4
     4  4  4  4
ans(:,:,2) =
     4  4  4  4
     4  4  4  4
     4  4  4  4
     4  4  4  4
ans(:,:,3) =
     4  4  4  4
     4  4  4  4
     4  4  4  4
     4  4  4  4
```

```
ans(:,:,4) =
     4   4   4   4
     4   4   1   4
     4   4   4   4
     4   4   4   4
```

As shown in the example, using a cell array allows data at each level to be systematically stored and accessed within a single variable, providing an efficient and organized structure for implementing the multigrid method in MATLAB. Let $N_x = 2^a$, $N_y = 2^b$, and $N_z = 2^c$, where a, b, and c are positive integers, and let $h = (R_1 - L_1)/N_x = (R_2 - L_2)/N_y = (R_3 - L_3)/N_z$ be the uniform spatial step size. We define the global discrete domain as

$$\Omega^d = \big\{\big(x_i = L_1 + (i-0.5)h,\ y_j = L_2 + (j-0.5)h,\ z_k = L_3 + (k-0.5)h\big) \mid i = 1, 2, \ldots, N_x,\ j = 1, 2, \ldots, N_y,\ k = 1, 2, \ldots, N_z\big\},$$

and define $M = \min(a, b, c)$. The approximate solution for $u(x_i, y_j, z_k, n\Delta t)$ is denoted by u^n_{ijk}, where Δt is the size of temporal step size. To solve Equation (1) on Ω^d, we apply the fully implicit method for the time discretization and use the 7-stencil finite difference method to discretize the Laplace operator.

$$\frac{u^{n+1}_{ijk} - u^n_{ijk}}{\Delta t} = \frac{1}{h^2}\left(u^{n+1}_{i+1,jk} + u^{n+1}_{i-1,jk} + u^{n+1}_{i,j+1,k} + u^{n+1}_{i,j-1,k} + u^{n+1}_{ij,k+1} + u^{n+1}_{ij,k-1} - 6u^{n+1}_{ijk}\right). \tag{4}$$

The fully implicit discretization (4) is unconditionally stable for any ratio $\alpha = \Delta t/h^2$ [21]. Therefore, stability does not restrict the choices of $\Delta t > 0$ and $h > 0$. The Neumann boundary conditions (2) are applied as follows:

$$\begin{aligned}
u^n_{0,jk} &= u^n_{1,jk},\ u^n_{N_x+1,jk} = u^n_{N_x,jk} \quad \text{for } j = 1, 2, \ldots, N_y,\ k = 1, 2, \ldots, N_z \\
u^n_{i,0,k} &= u^n_{i,1,k},\ u^n_{i,N_y+1,k} = u^n_{i,N_y,k} \quad \text{for } i = 1, 2, \ldots, N_x,\ k = 1, 2, \ldots, N_z \\
u^n_{ij,0} &= u^n_{ij,1},\ u^n_{ij,N_z+1} = u^n_{ij,N_z} \quad \text{for } i = 1, 2, \ldots, N_x,\ j = 1, 2, \ldots, N_y.
\end{aligned}$$

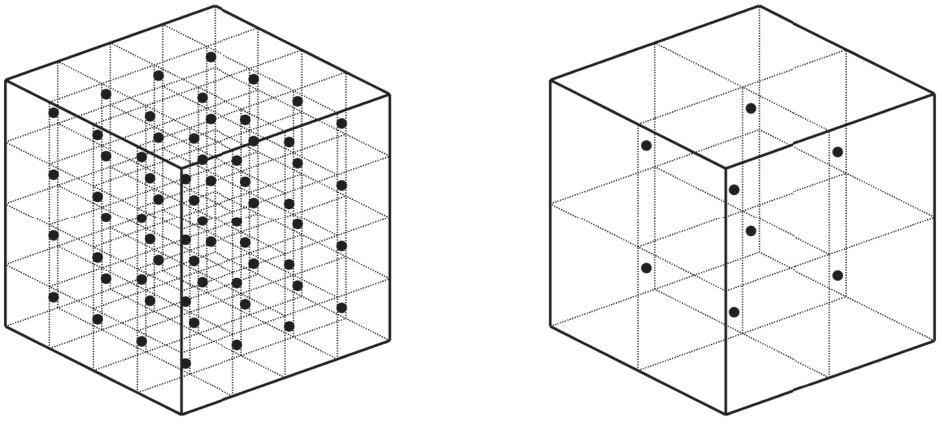
To apply the multigrid algorithm, we define the hierarchy of discrete domains $\tilde{\Omega}_m$ for the multigrid levels $m = 1, 2, \ldots, M$ as follows:

$$\tilde{\Omega}_m = \left\{ \big(x_i = L_1 + (i-0.5)h_m,\ y_j = L_2 + (j-0.5)h_m,\ z_k = L_3 + (k-0.5)h_m\big) \mid i = 1, 2, \ldots, \frac{N_x}{2^{m-1}},\ j = 1, 2, \ldots, \frac{N_y}{2^{m-1}},\ k = 1, 2, \ldots, \frac{N_z}{2^{m-1}},\ h_m = 2^{m-1}h \right\}.$$

That is, the finest grid in the multigrid hierarchy, $\tilde{\Omega}_1$, corresponds to the global discrete domain Ω^d. Figure 1(a,b) show schematic diagrams of $\tilde{\Omega}_1$ and $\tilde{\Omega}_2$, respectively, for $N_x = N_y = N_z = 2^2$, corresponding to $M = 2$.

We define $\tilde{u}^n_{m,ijk}$ as the numerical solution of $\tilde{u}(x_i, y_j, z_k, n\Delta t)$ on $\tilde{\Omega}_m$. In particular, on the finest grid, we set $\tilde{u}^n_{1,ijk} = u^n_{ijk}$. The following MATLAB code demonstrates how to create and store data using a cell structure:

```
xL = 0; xR = 1; Nx = 8; h = (xR-xL)/Nx; yL = 0; yR = 1;
Ny = (yR-yL)/h; zL = 0; zR = 1; Nz = (zR-zL)/h;
x=linspace(xL+0.5*h,xR-0.5*h,Nx);y=linspace(yL+0.5*h,yR-0.5*h,Ny);
z = linspace(zL+0.5*h,zR-0.5*h,Nz);
total_levels = min([log2(Nx) log2(Ny) log2(Nz)]);
for m = 1:total_levels
```



(a) $\tilde{\Omega}_1 = \Omega^d$ (b) $\tilde{\Omega}_2$

Figure 1. Schematic diagram of the discrete domain $\tilde{\Omega}_m$ for the multigrid levels $m = 1, 2$. (a) $\tilde{\Omega}_1 = \Omega^d$ (b) $\tilde{\Omega}_2$.

```
    tilde_u(m) = {zeros(Nx/2^(m-1),Ny/2^(m-1),Nz/2^(m-1))};
    H(m) = 2^(m-1)*h;
    NX(m) = Nx/2^(m-1); NY(m) = Ny/2^(m-1); NZ(m) = Nz/2^(m-1);
end
```

This code creates a cell array 'tilde_u', where each entry is initialized as a zero-valued array of size $N_x/2^{m-1} \times N_y/2^{m-1} \times N_z/2^{m-1}$, to represent the discrete concentration $\tilde{u}^n_{ijk}$ on the discrete domain at each multigrid level $m = 1, 2, 3$. In addition, the spatial step size $h_m = 2^{m-1}h$ and the total number of grid points $N_x/2^{m-1}$, $N_y/2^{m-1}$, and $N_z/2^{m-1}$ at each multigrid level are stored in the variables 'H', 'NX', 'NY', and 'NZ', respectively. Then, for $m = 1, \ldots, M$, the Neumann boundary conditions are applied on the discrete domain $\tilde{\Omega}_m$ as

$$\begin{aligned}
&\tilde{u}^n_{m,0,jk} = \tilde{u}^n_{m,1,jk}, \quad \tilde{u}^n_{m,\frac{N_x}{2^{m-1}}+1,jk} = \tilde{u}^n_{m,\frac{N_x}{2^{m-1}},jk} \quad \text{for } j = 1, \ldots, \frac{N_y}{2^{m-1}},\ k = 1, \ldots, \frac{N_z}{2^{m-1}},\\
&\tilde{u}^n_{m,i,0,k} = \tilde{u}^n_{m,i,1,k}, \quad \tilde{u}^n_{m,i,\frac{N_y}{2^{m-1}}+1,k} = \tilde{u}^n_{m,i,\frac{N_y}{2^{m-1}},k} \quad \text{for } i = 1, \ldots, \frac{N_x}{2^{m-1}},\ k = 1, \ldots, \frac{N_z}{2^{m-1}}, \qquad (5)\\
&\tilde{u}^n_{m,ij,0} = \tilde{u}^n_{m,ij,1}, \quad \tilde{u}^n_{m,ij,\frac{N_z}{2^{m-1}}+1} = \tilde{u}^n_{m,ij,\frac{N_z}{2^{m-1}}} \quad \text{for } i = 1, \ldots, \frac{N_x}{2^{m-1}},\ j = 1, \ldots, \frac{N_y}{2^{m-1}}.
\end{aligned}$$

The linear operator $\mathcal{L}_m$ on $\tilde{\Omega}_m$ is defined by

$$\begin{aligned}
\mathcal{L}_m(\tilde{u}^{n+1}_{m,ijk}) = \left(\frac{1}{\Delta t} + \frac{6}{h_m^2}\right)\tilde{u}^{n+1}_{m,ijk} - \frac{1}{h_m^2}\Big(&\tilde{u}^{n+1}_{m,i+1,jk} + \tilde{u}^{n+1}_{m,i-1,jk}\\
&+\tilde{u}^{n+1}_{m,i,j+1,k} + \tilde{u}^{n+1}_{m,i,j-1,k} + \tilde{u}^{n+1}_{m,ij,k+1} + \tilde{u}^{n+1}_{m,ij,k-1}\Big) \quad \text{on } \tilde{\Omega}_m
\end{aligned}$$

and the source term $\tilde{f}_{1,ijk}$ on $\tilde{\Omega}_1$ is defined as

$$\tilde{f}_{1,ijk} = \frac{\tilde{u}^n_{1,ijk}}{\Delta t} \quad \text{on } \tilde{\Omega}_1.$$

Let $\tilde{\mathbf{u}}_m^n$ be the set of $\tilde{u}_{m,ijk}^n$ on $\tilde{\Omega}_m$, where $\tilde{u}_{m,ijk}^n$ represents an arbitrary discrete function defined on $\tilde{\Omega}_m$. We describe the multigrid *V-cycle* function, defined as *MGV*, as follows:

$$\mathbf{u}^{n+1} = MGV(M, \mathbf{u}^n, s_1, s_2, tol), \tag{6}$$

where M is the maximum multigrid level and $\mathbf{u}^n = \tilde{\mathbf{u}}_1^n$ on $\Omega^d = \tilde{\Omega}_1$. The parameters s_1 and s_2 are the number of pre-smoothing and post-smoothing iterations, respectively, and *tol* is the given tolerance. We define the *V-cycle* as

$$\tilde{\mathbf{u}}_1^{n+1,v+1} = V\text{-}cycle\left(M, \tilde{\mathbf{u}}_1^{n+1,v}, \mathcal{L}_m, \tilde{\mathbf{f}}_m, s_1, s_2\right), \tag{7}$$

where v is the number of *V-cycle* function iterations, $\tilde{\mathbf{u}}_m^{n+1,0} = \tilde{\mathbf{u}}_m^n$, and $\tilde{\mathbf{u}}_m^{n+1,v+1}$ is the numerical solution on $\tilde{\Omega}_m$ after applying the *V-cycle* function $v+1$ iterations. We introduce the following smoothing function:

$$\hat{\mathbf{u}}_m^{n+1,v} = SMOOTH^s(\tilde{\mathbf{u}}_m^{n+1,v}, \mathcal{L}_m, \tilde{\mathbf{f}}_m), \tag{8}$$

where s is the number of the smoothing iterations, such as s_1 and s_2. The smoothed numerical solution $\hat{\mathbf{u}}_m^{n+1,v}$ is obtained by repeating the following Gauss–Seidel method s times.

$$\begin{aligned}
&\text{For } i = 1, 2, \ldots, \frac{N_x}{2^{m-1}},\\
&\text{For } j = 1, 2, \ldots, \frac{N_y}{2^{m-1}},\\
&\text{For } k = 1, 2, \ldots, \frac{N_z}{2^{m-1}},\\
&\tilde{u}_{m,ijk}^{n+1,v,g+1} = \Big(\tilde{f}_{m,ijk} + \tilde{u}_{m,i-1,jk}^{n+1,v,g+1} + \tilde{u}_{m,i+1,jk}^{n+1,v,g} + \tilde{u}_{m,i,j-1,k}^{n+1,v,g+1} + \tilde{u}_{m,i,j+1,k}^{n+1,v,g}\\
&\qquad + \tilde{u}_{m,ij,k-1}^{n+1,v,g+1} + \tilde{u}_{m,ij,k+1}^{n+1,v,g}\Big) \Big/ \left(\frac{1}{\Delta t} + \frac{6}{h^2}\right),
\end{aligned}$$

where g is the Gauss–Seidel iteration step and the initial values are given by $\tilde{\mathbf{u}}_m^{n+1,v,0} = \tilde{\mathbf{u}}_m^{n+1,v}$. The update follows forward lexicographic ordering. The updates start from $i = 1, j = 1, k = 1$. For fixed i and j, the index k increases from 1 to $N_z/2^{m-1}$. After the updates for k are completed, the index j increases by one and the same updates for k are repeated. After the updates for $j = 1, \ldots, N_y/2^{m-1}$ are completed, the index i increases by one and the same procedure is repeated. Thus, the final smoothed solution is obtained as $\hat{\mathbf{u}}_m^{n+1,v} = \tilde{\mathbf{u}}_m^{n+1,v,s}$. The *V-cycle* (7) consists of the following two steps.

- Pre-smoothing and computation of the correction values

We perform pre-smoothing for $m = 1$ as follows using the smoothing function (8).

$$\hat{\mathbf{u}}_1^{n+1,v} = SMOOTH^{s_1}(\tilde{\mathbf{u}}_1^{n+1,v}, \mathcal{L}_1, \tilde{\mathbf{f}}_1).$$

Then, for $m = 1, 2, \ldots, M-1$, we compute the defect $\hat{\mathbf{d}}_{m+1}$ and correction values $\hat{\mathbf{w}}_{m+1}$ on $\tilde{\Omega}_{m+1}$ through the following procedures. First, for $m = 1$, the defect $\hat{\mathbf{d}}_1$ on $\tilde{\Omega}_1$ is computed as follows using

the source term $\tilde{\mathbf{f}}_1$ and the numerical solution $\hat{\mathbf{u}}_1^{n+1,\nu}$ of Equation (9).

$$\hat{\mathbf{d}}_1 = \tilde{\mathbf{f}}_1 - \mathcal{L}_1(\hat{\mathbf{u}}_1^{n+1,\nu}).$$

Then, we define the restriction operator R_{m+1}, which transfers values from the fine grid Ω_m to the coarse grid Ω_{m+1}, as follows:

$$\tilde{\mathbf{d}}_{m+1} = R_{m+1}\left(\hat{\mathbf{d}}_m\right),$$

where the coarse grid values $\tilde{\mathbf{d}}_{m+1}$ are given by $\tilde{d}_{m+1,ijk}$ for $i = 1, \ldots, N_x/2^m$, $j = 1, \ldots, N_y/2^m$, and $k = 1, \ldots, N_z/2^m$ are computed as

$$\begin{aligned}\tilde{d}_{m+1,ijk} = \frac{1}{8}\Big(&\hat{d}_{m,2i,2j,2k} + \hat{d}_{m,2i-1,2j,2k} + \hat{d}_{m,2i,2j-1,2k} + \hat{d}_{m,2i,2j,2k-1} \\ &+\hat{d}_{m,2i-1,2j-1,2k} + \hat{d}_{m,2i-1,2j,2k-1} + \hat{d}_{m,2i,2j-1,2k-1} + \hat{d}_{m,2i-1,2j-1,2k-1}\Big).\end{aligned}$$

We define the correction equation on the coarse grid as

$$\mathcal{L}_{m+1}\left(\tilde{\mathbf{u}}_{m+1}^{n+1,\nu}\right) = R_{m+1}\left(\hat{\mathbf{d}}_m\right).$$

To obtain the correction values, we use the smoothing function (8), which uses the Gauss–Seidel method.

$$\hat{\mathbf{w}}_2 = SMOOTH^{s_1}\left(\tilde{\mathbf{w}}_2, \mathcal{L}_2, R_2\left(\hat{\mathbf{d}}_1\right)\right),$$

where the initial values are given by $\tilde{\mathbf{w}}_2 = \mathbf{0}$ on $\tilde{\Omega}_2$. Next, for $m = 2, \ldots, M-1$, the defect is computed by

$$\hat{\mathbf{d}}_m = R_m\left(\hat{\mathbf{d}}_{m-1}\right) - \mathcal{L}_m(\hat{\mathbf{w}}_m).$$

The correction values $\hat{\mathbf{w}}_{m+1}$ on $\tilde{\Omega}_{m+1}$ are obtained by

$$\hat{\mathbf{w}}_{m+1} = SMOOTH^{s_1}\left(\tilde{\mathbf{w}}_{m+1}, \mathcal{L}_{m+1}, R_{m+1}\left(\hat{\mathbf{d}}_m\right)\right),$$

where the initial values are $\tilde{\mathbf{w}}_{m+1} = \mathbf{0}$ on $\tilde{\Omega}_{m+1}$. Consequently, we obtain the correction values $\hat{\mathbf{w}}_{m+1}$ on $\tilde{\Omega}_{m+1}$ for $m = 1, \ldots, M-1$. The correction value on the coarsest grid level $\tilde{\Omega}_M$ is computed in the same manner as the smoothing step on the other levels. We set the initial value to $\tilde{\mathbf{w}}_M = 0$ and obtain the correction value $\hat{\mathbf{w}}_M$ by applying s_1 iterations of the Gauss–Seidel method. The following MATLAB code presents the pre-smoothing process and the computation of the correction values, together with a comment on the computation of the correction value on the coarsest grid level.

```
f = tilde_u; f(1) = {tilde_u{1}/dt};
for m = 1:total_levels-1
    if m > 1
         tilde_u{m} = zeros(NX(m),NY(m),NZ(m));
    end
    tilde_u{m} = SMOOTH(tilde_u{m},f{m},s1,m);
    d = defect(tilde_u{m},f{m},m);
    for i = 1:NX(m+1)
    for j = 1:NY(m+1)
    for k = 1:NZ(m+1)
         f{m+1}(i,j,k) = 0.125*(d(2*i,2*j,2*k)+d(2*i-1,2*j,2*k) ...
             +d(2*i,2*j-1,2*k)+d(2*i,2*j,2*k-1) ...
             +d(2*i-1,2*j-1,2*k)+d(2*i-1,2*j,2*k-1) ...
             +d(2*i,2*j-1,2*k-1)+d(2*i-1,2*j-1,2*k-1));
    end
```

```
    end
    end
end
% On the coarsest grid level, the correction is initialized to zero and
% approximated using SMOOTH with Gauss--Seidel iterations.
m=total_levels; tilde_u{m}=zeros(NX(m),NY(m),NZ(m));
tilde_u{m}=SMOOTH(tilde_u{m},f{m},s1,m);
function hat_u = SMOOTH(tilde_u,f,relax,m)
global NX NY NZ dt H
for iter = 1:relax
    for i = 1:NX(m)
    for j = 1:NY(m)
    for k = 1:NZ(m)
         sor = f(i,j,k); coef = 1/dt;
         if i>1


             sor=sor+tilde_u(i-1,j,k)/H(m)^2; coef=coef+1/H(m)^2;
         end
         if i<NX(m)
             sor=sor+tilde_u(i+1,j,k)/H(m)^2; coef=coef+1/H(m)^2;
         end
         if j>1
             sor=sor+tilde_u(i,j-1,k)/H(m)^2; coef=coef+1/H(m)^2;
         end
         if j<NY(m)
             sor=sor+tilde_u(i,j+1,k)/H(m)^2; coef=coef+1/H(m)^2;
         end
         if k>1
             sor=sor+tilde_u(i,j,k-1)/H(m)^2; coef=coef+1/H(m)^2;
         end
         if k<NZ(m)
             sor=sor+tilde_u(i,j,k+1)/H(m)^2; coef=coef+1/H(m)^2;
         end
         tilde_u(i,j,k) = sor/coef;
    end
    end
    end
end
hat_u=tilde_u;
end
function d = defect(tilde_u,f,m)
global NX NY NZ dt H
for i = 1:NX(m)
for j = 1:NY(m)
for k = 1:NZ(m)
    Lap=0;
    if i>1
         Lap=Lap+(tilde_u(i-1,j,k)-tilde_u(i,j,k))/H(m)^2;
    end
    if i<NX(m)
         Lap=Lap+(tilde_u(i+1,j,k)-tilde_u(i,j,k))/H(m)^2;
    end
    if j>1
         Lap=Lap+(tilde_u(i,j-1,k)-tilde_u(i,j,k))/H(m)^2;
    end
    if j<NY(m)
         Lap=Lap+(tilde_u(i,j+1,k)-tilde_u(i,j,k))/H(m)^2;
    end
    if k>1
         Lap=Lap+(tilde_u(i,j,k-1)-tilde_u(i,j,k))/H(m)^2;
    end
    if k<NZ(m)
         Lap=Lap+(tilde_u(i,j,k+1)-tilde_u(i,j,k))/H(m)^2;
    end
    d(i,j,k) = f(i,j,k)-(tilde_u(i,j,k)/dt-Lap);
end
```

```
end
end
end
```

The source term 'f' is created as a cell array using the previously defined 'tilde_u', and the multigrid computations are performed for each level. In addition, the functions 'SMOOTH' and 'defect' are defined and used to perform computations on each multigrid level. In the function 'defect', the term 'tilde_u(i,j,k)/dt - Lap' represents the evaluation of the linear operator $\mathcal{L}(\tilde{u}_{ijk})$. The homogeneous Neumann boundary condition is incorporated in the code through the conditional statements such as 'if i¿1' and 'if i¡NX(m)'. At boundary indices, the terms that would require values outside the domain are not added. This treatment follows the discrete boundary condition (5), which implies that the outside-neighbour contributions cancel at $\partial\Omega$. The same convention is applied in the y- and z-directions.

- Correction of the numerical solution and post-smoothing

We define the interpolation operator I_m for $m = 1, \ldots, M-1$ as

$$\tilde{\mathbf{w}}_m = I_m \left(\hat{\mathbf{w}}_{m+1}\right),$$

where the fine grid values $\tilde{\mathbf{w}}_m$ on $\tilde{\Omega}_m$ are computed as

$$\begin{aligned}
&\text{For } i = 1, \ldots, N_x/2^m,\ j = 1, \ldots, N_y/2^m,\ k = 1, \ldots, N_z/2^m,\\
&\tilde{w}_{m,2i-1,2j-1,2k-1} = \hat{w}_{m+1,ijk}, \quad \tilde{w}_{m,2i,2j-1,2k-1} = \hat{w}_{m+1,ijk}, \quad \tilde{w}_{m,2i-1,2j,2k-1} = \hat{w}_{m+1,ijk},\\
&\tilde{w}_{m,2i-1,2j-1,2k} = \hat{w}_{m+1,ijk}, \quad \tilde{w}_{m,2i,2j,2k-1} = \hat{w}_{m+1,ijk}, \quad \tilde{w}_{m,2i,2j-1,2k} = \hat{w}_{m+1,ijk},\\
&\tilde{w}_{m,2i-1,2j,2k} = \hat{w}_{m+1,ijk}, \quad \tilde{w}_{m,2i,2j,2k} = \hat{w}_{m+1,ijk}.
\end{aligned}$$

Then, for $m = M-1, M-2, \ldots, 2$, we calculate the correction of the numerical solution $\hat{\mathbf{w}}_m$ as follows. The correction is computed by incorporating the interpolated values from the coarser grid.

$$\mathbf{w}_m = \hat{\mathbf{w}}_m + I_m \left(\hat{\mathbf{w}}_{m+1}\right)$$

We apply post-smoothing to refine the corrected solution.

$$\hat{\mathbf{w}}_m = SMOOTH^{s_2}\left(\mathbf{w}_m, \mathcal{L}_m, \hat{\mathbf{d}}_m\right).$$

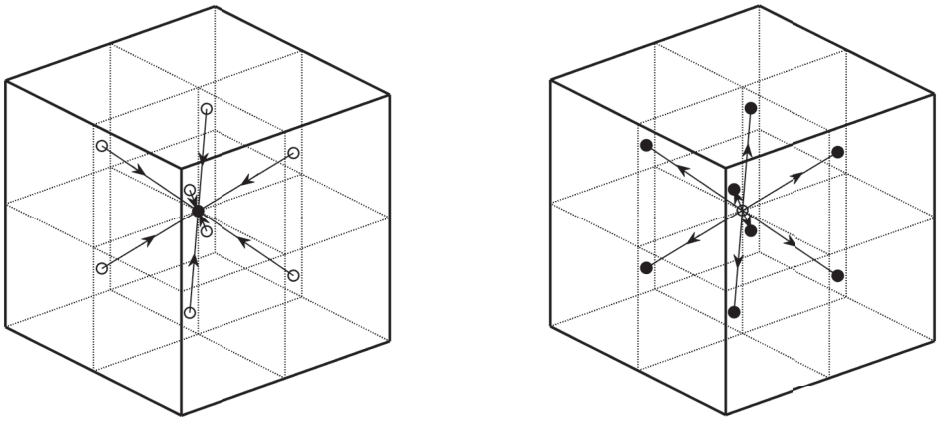
Finally, we compute the numerical solution $\tilde{\mathbf{u}}_1^{n+1,\nu+1}$ of the *V-cycle* (7) as follows.

$$\tilde{\mathbf{u}}_1^{n+1,\nu+1} = SMOOTH^{s_2}\left(\mathbf{w}_1, \mathcal{L}_1, \tilde{\mathbf{f}}_1\right),$$

where $\mathbf{w}_1 = \hat{\mathbf{w}}_1 + I_1(\hat{\mathbf{w}}_2)$. The following MATLAB code applies correction and post-smoothing to the numerical solution.

```
for m=total_levels-1:-1:1
    for i = 1:NX(m+1)
    for j = 1:NY(m+1)
    for k = 1:NZ(m+1)
         tilde_u{m}(2*i-1:2*i,2*j-1:2*j,2*k-1:2*k) ...


           = tilde_u{m}(2*i-1:2*i,2*j-1:2*j,2*k-1:2*k) ...
           +tilde_u{m+1}(i,j,k);
    end
    end
    end
    tilde_u{m}=SMOOTH(tilde_u{m},f{m},s2,m);
end
```



(a) Restriction (b) Interpolation

Figure 2. Schematic diagram of the restriction and interpolation operations in the *V-cycle* (7). (a) Restriction (b) Interpolation.

Figure 2(a,b) present schematic diagrams of the restriction and interpolation operations, respectively.

Thus, one iteration of the *V-cycle* (7) has been described. We perform the *V-cycle* until the resultant error $\|\tilde{\mathbf{u}}_1^{n+1,\nu+1} - \tilde{\mathbf{u}}_1^{n+1,\nu}\|_\infty \leq tol$ is satisfied in order to obtain the numerical solution of the *MGV*. Therefore, the numerical solution of the *MGV* is given as $\mathbf{u}^{n+1} = \tilde{\mathbf{u}}_1^{n+1,V+1}$, where V represents the total number of iterations performed until convergence.

3. Numerical experiments

3.1. Convergence experiment

We perform a convergence test to verify whether the proposed non-recursive multigrid algorithm accurately solves the 3D diffusion equation. All computations are performed using a MATLAB cell array implementation. We consider the following initial condition on $\Omega = (0,1) \times (0,1) \times (0,1)$.

$$u(x,y,z,0) = \cos(\pi x)\cos(\pi y)\cos(\pi z), \quad \mathbf{x} \in \Omega.$$

Thus, the analytical solution of the diffusion Equation (1) is given by

$$u(x,y,z,t) = e^{-3\pi^2 t}\cos(\pi x)\cos(\pi y)\cos(\pi z), \quad \mathbf{x} \in \Omega,\ t > 0.$$

The numerical error is defined as $e_{ijk}^{N_t} = u(x_i, y_j, z_k, N_t \Delta t) - u_{ijk}^{N_t}$, and we define the numerical L_2-error as

$$\left\| \mathbf{e}_{N_x,N_y,N_z}^{N_t} \right\|_2 = \sqrt{\frac{1}{N_x N_y N_z} \sum_{k=1}^{N_z} \sum_{j=1}^{N_y} \sum_{i=1}^{N_x} \left(e_{ijk}^{N_t} \right)^2}.$$

The convergence rates for the temporal and spatial step sizes are defined by

$$R_t = \log_2 \left(\frac{\|\mathbf{e}_{N_x,N_y,N_z}^{N_t}\|_2}{\|\mathbf{e}_{N_x,N_y,N_z}^{2N_t}\|_2} \right) \quad \text{and} \quad R_s = \log_2 \left(\frac{\|\mathbf{e}_{N_x,N_y,N_z}^{N_t}\|_2}{\|\mathbf{e}_{2N_x,2N_y,2N_z}^{N_t}\|_2} \right),$$

Table 1. Errors and convergence rates of the fully implicit scheme with respect to the time step Δt.

Δt	2.00e−04		1.00e−04		5.00e−05		2.50e−05		1.25e−05
l_2-error	5.8263e−05		2.9219e−05		1.4654e−05		7.3616e−06		3.7127e−06
R_t		1.00		1.00		0.99		0.99	

Table 2. Errors and convergence rates of the fully implicit scheme with respect to the grid size h.

$N_x = N_y = N_z$	16		32		64		128		256		512
l_2-error	3.3589e−09		8.4055e−10		2.1020e−10		5.2567e−11		1.3155e−11		3.3019e−12
R_s		2.00		2.00		2.00		2.00		1.99	

respectively. First, we conduct a convergence test with respect to the time step size Δt. The parameters used are $N_x = N_y = N_z = 512$, $s_1 = s_2 = 2$, $tol = 1.0\text{e}{-12}$, and the final time $T = 0.002$. The test is performed for different values of $N_t = 10, 20, 40, 80$, and 160, corresponding to different time step sizes $\Delta t = 2.00\text{e}{-4}$, $1.00\text{e} - 4$, $5.00\text{e}{-5}$, $2.50\text{e}{-5}$, and $1.25\text{e}{-5}$, respectively. Table 1 lists the errors and convergence rates with respect to the time step size Δt for the numerical solution obtained by the multigrid algorithm based on the MATLAB cell structure.

Next, we conduct a convergence test with respect to the spatial step size h. The parameters used are $\Delta t = 1.00\text{e}{-9}$, $s_1 = s_2 = 2$, $tol = 1.0\text{e}{-12}$, and the final time $T = 1.00\text{e}{-7}$. The test is performed for different values of $N_x = N_y = N_z = 16, 32, 64, 128, 256$, and 512, corresponding to different spatial step sizes $h = 1/16, 1/32, 1/64, 1/128, 1/256$, and 1/512, respectively. Table 2 lists the errors and convergence rates with respect to the spatial step size h for the computational solution obtained by the multigrid algorithm based on the MATLAB cell structure. We observe from Tables 1 and 2 that the numerical solution of the diffusion equation in 3D, obtained using the multigrid algorithm based on the MATLAB cell structure, converges with first-order accuracy in time and second-order accuracy in space.

3.2. Dynamic behaviour of the diffusion equation

In this section, we conduct numerical tests to observe the time-dependent dynamics of the diffusion equation, which is solved using a multigrid algorithm based on the MATLAB cell structure. To visualize the three-dimensional numerical solution, we partition the value range $[-1, 1]$ into 11 equal subintervals. Let $r_p = -1 + 2p/11$ for $p = 0, 1, \ldots, 11$. For $p = 1, 2, \ldots, 10$, we define

$$\psi_{p,ijk} = \begin{cases} 1, & \text{if } r_{p-1} \le u_{ijk}^{N_t} < r_p, \\ -1, & \text{otherwise.} \end{cases}$$

For $p = 11$, we set

$$\psi_{11,ijk} = \begin{cases} 1, & \text{if } r_{10} \le u_{ijk}^{N_t} \le r_{11}, \\ -1, & \text{otherwise.} \end{cases}$$

The region corresponding to the p-th subinterval of $u_{ijk}^{N_t}$ is visualized through the zero-level isosurface of $\psi_{p,ijk}$. This visualization may yield jagged isosurfaces, and the degree of roughness depends on the spatial grid resolution. Many smoothing techniques can be used to reduce such jaggedness. In this study, we apply the MATLAB built-in function 'smooth3' for this purpose. As the spatial step size decreases, the computed isosurfaces become smoother even without smoothing. The initial condition on $\Omega = (0, 1) \times (0, 1) \times (0, 1)$ is given as

$$u(x, y, z, 0) = \cos(\pi x)\cos(\pi y)\cos(\pi z), \quad \mathbf{x} \in \Omega. \tag{9}$$

The parameters used are $N_x = N_y = N_z = 128$, $s_1 = s_2 = 2$, and $tol = 1.0\text{e}{-9}$. Figure 3 displays the temporal evolution of the computational solution of the diffusion equation computed using the

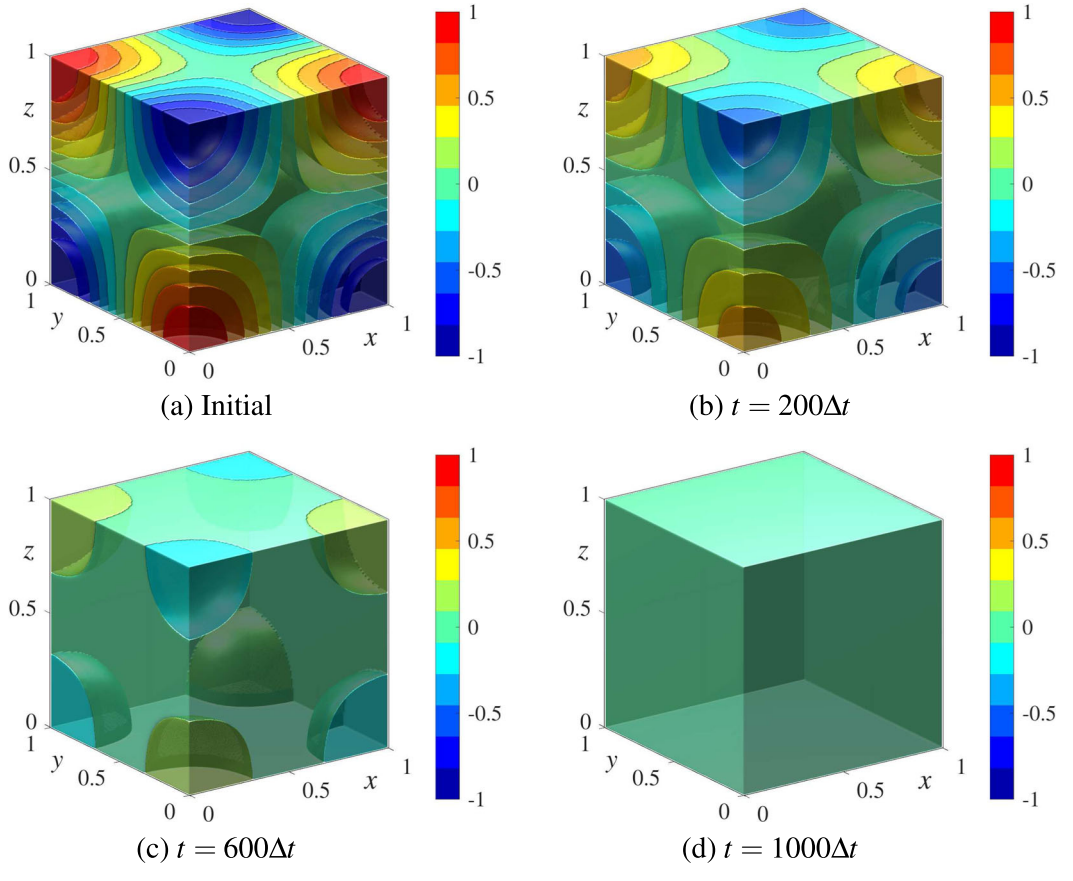


Figure 3. Snapshots of the numerical solution of the 3D diffusion equation with initial condition (9) obtained using a multigrid algorithm based on the MATLAB cell structure at $t = 0$, $200\Delta t$, $600\Delta t$, and $1000\Delta t$. (a) Initial (b) $t = 200\Delta t$ (c) $t = 600\Delta t$ (d) $t = 1000\Delta t$.

multigrid algorithm based on the MATLAB cell structure for the initial condition (9). Figure 3(a) illustrates the initial concentration distribution, which exhibits a periodic spatial pattern with values ranging from -1 to 1. At $t = 200\Delta t$, as shown in Figure 3(b), the diffusion process reduces the steep gradients observed in the initial condition and produces a smoother concentration profile. Figure 3(c) corresponds to $t = 600\Delta t$, where the concentration field becomes more uniform, and the differences between high and low concentration regions diminish further. Finally, at $t = 1000\Delta t$, depicted in Figure 3(d), the concentration distribution approaches a nearly homogeneous state, which implies that the diffusion process has nearly reached equilibrium.

Next, we consider an initial condition with a more abrupt concentration variation. The initial condition on $\Omega = (0,1) \times (0,1) \times (0,1)$ is given as follows.

$$u(x,y,z,0) = \begin{cases} 1, & \text{if } \sqrt{x^2+y^2+z^2} < 0.8, \\ -1, & \text{otherwise.} \end{cases} \tag{10}$$

Figure 4 shows the snapshots of the computational solution of the diffusion equation at $t = 0, 400\Delta t$, $1000\Delta t$, and $2000\Delta t$. The initial concentration, shown in Figure 4(a), consists only of the values -1 and 1, which results in a steep concentration variation in space. Figure 4(b–d) show the diffusion process, where the concentration distribution gradually becomes smoother over time. As time progresses, the concentration field approaches a more uniform state.

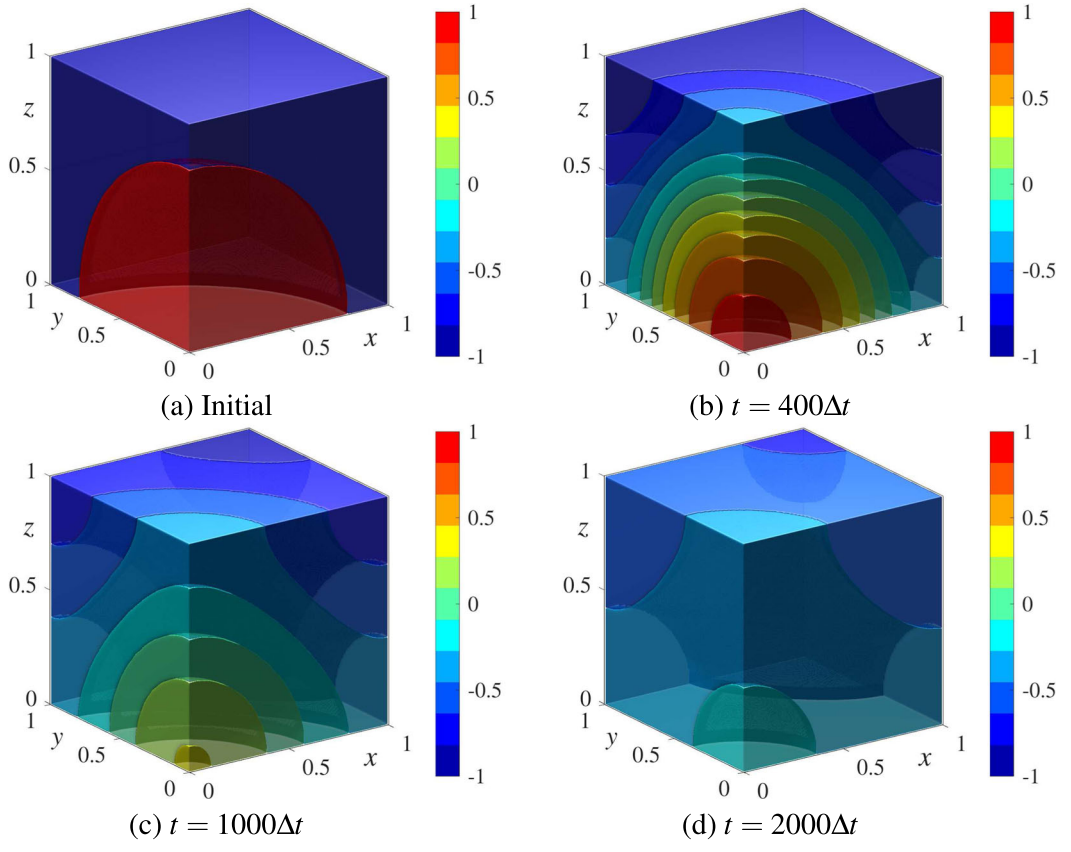


Figure 4. Snapshots of the numerical solution of the 3D diffusion equation with initial condition (10) obtained using a multigrid algorithm based on the MATLAB cell structure at $t = 0$, $400\Delta t$, $1000\Delta t$, and $2000\Delta t$. (a) Initial (b) $t = 400\Delta t$ (c) $t = 1000\Delta t$ (d) $t = 2000\Delta t$.

4. Conclusion

In this study, we presented a non-recursive implementation of the multigrid method for solving the 3D diffusion equation with level-wise data management based on cell structure. We included a MATLAB example based on cell arrays. The proposed implementation removes recursion, which is a standard algorithmic form in multigrid methods. Such recursive structures can make it difficult to debug an incorrect implementation at a specific grid level. The proposed level-separated structure reduces this difficulty and makes the procedure easier to follow and apply to various problems. Furthermore, the cell-based organization provides a structured and intuitive method. The computational efficiency of the multigrid algorithm is also preserved. Numerical experiments confirmed that the implementation achieves first-order accuracy in time and second-order accuracy in space. The diffusion process under different initial conditions was also analysed, and this analysis verified the reliability of the algorithm. The results demonstrate that the cell-based multigrid approach is an accessible and efficient method for solving the 3D diffusion equation. This implementation provides a practical alternative for researchers who want to apply the multigrid method without using recursion. In future work, this approach will be extended to nonlinear equations.

Acknowledgments

We thank the reviewers for their helpful comments, which improved the manuscript.

Author contributions

CRediT: **Youngjin Hwang:** Conceptualization, Data curation, Formal analysis, Investigation, Methodology, Validation, Visualization, Writing – original draft, Writing – review & editing; **Soobin Kwak:** Formal analysis, Investigation, Methodology, Writing – original draft; **Jyoti:** Formal analysis, Investigation, Visualization, Writing – original draft; **Hyunho Shin:** Conceptualization, Validation, Writing – original draft, Writing – review & editing; **Xinpei Wu:** Formal analysis, Investigation, Methodology, Writing – original draft, Writing – review & editing; **Junseok Kim:** Conceptualization, Formal analysis, Funding acquisition, Investigation, Methodology, Project administration, Supervision, Validation, Visualization, Writing – original draft, Writing – review & editing

Disclosure statement

No potential conflict of interest was reported by the author(s).

Funding

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. RS-2025-00562584); Brain Korea 21 FOUR from the Ministry of Education of Korea.

Code availability

The complete multigrid algorithm for the 3D diffusion equation using the cell structure is available at: https://github.com/Youngjin0830/cell-structure-multigrid.git.

References

[1] M. Anselmann and M. Bause, *A geometric multigrid method for space-time finite element discretizations of the Navier–Stokes equations and its application to 3d flow simulation*, ACM Trans. Math. Software 49(1) (2023), pp. 1–25.
[2] A. Brandt, *Multi-level adaptive solutions to boundary-value problems*, Math. Comput. 31(138) (1977), pp. 333–390.
[3] W.L. Briggs, V.E. Henson, and S.F. McCormick, *A Multigrid Tutorial*, SIAM, Philadelphia, 2000.
[4] M. Caldana, P.F. Antonietti, and L. Dede, *A deep learning algorithm to accelerate algebraic multigrid methods in finite element solvers of 3d elliptic pdes*, Comput. Math. Appl. 167 (2024), pp. 217–231.
[5] Y. Chen, B. Dong, and J. Xu, *Meta-mgnet: meta multigrid networks for solving parameterized partial differential equations*, J. Comput. Phys. 455 (2022), p.110996.
[6] G. Ciaramella, F. Nobile, and T. Vanzan, *A multigrid solver for pde-constrained optimization with uncertain inputs*, J. Sci. Comput. 101(1) (2024), p. 13.
[7] J. Crank, *The Mathematics of Diffusion*, Oxford University Press, New York, 1979.
[8] J. Dünnebacke, S. Turek, C. Lohmann, A. Sokolov, and P. Zajac, *Increased space-parallelism via time-simultaneous newton-multigrid methods for nonstationary nonlinear pde problems*, Int. J. High. Perform. Comput. Appl. 35(3) (2021), pp. 211–225.
[9] Dipesh and P. Kumar, *Modelling the role of delay in blood flow dynamics in human body using delay differential equations*, Phys. A: Stat. Mech. Appl. 668 (2025), p.130602.
[10] E. Dufresne, H.A. Harrington, J.D. Hauenstein, P.G. Kevrekidis, and P. Tripoli, *On some configurations of oppositely charged trapped vortices in the plane*, Adv. Appl. Math. 124 (2021), p.102099.
[11] M. Echchehira, M. Hannabou, M. Atraoui, and M. Bouaouid, *On an image denoising model based on Atangana–Baleanu fractional derivative*, Phys. A: Stat. Mech. Appl. 674 (2025), p.130633.
[12] L.C. Evans, *Partial Differential Equations*, Vol. 19, American Mathematical Society, 2022.
[13] A.V. Gorobets, *An approach to the implementation of the multigrid method with full approximation for cfd problems*, Comput. Math. Math. Phys. 63(11) (2023), pp. 2150–2161.
[14] W. Hackbusch, *Convergence of the multi-grid iteration*, in *Multi-Grid Methods and Applications*, Springer, Berlin, Heidelberg, 1985, pp. 160–168.
[15] M. Hashemi, S. Shalbaf, M. Jadidi, and A. Dolatabadi, *Effects of gas viscosity and liquid-to-gas density ratio on liquid jet atomization in crossflow*, AIP. Adv. 13(3) (2023), p.035105.
[16] R. Huang, R. Li, and Y. Xi, *Learning optimal multigrid smoothers via neural networks*, SIAM. J. Sci. Comput. 45(3) (2022), pp. S199–S225.
[17] Y. Hwang, Jyoti, S. Kwak, H. Kim, and J. Kim, *An explicit numerical method for the conservative Allen–Cahn equation on a cubic surface*, AIMS Math 9(12) (2024), pp. 34447–34465.
[18] Y. Jiang and H. Bai, *Multigrid methods for time fractional conservation laws*, Numer. Algorithms. 97(3) (2024), pp. 1301–1322.

[19] Y. Jin, S. Kwak, S. Ham, and J. Kim, *A fast and efficient numerical algorithm for image segmentation and denoising*, AIMS Math. 9(2) (2024), pp. 5015–5027.
[20] C. Jin, T. Wu, and J. Zhou, *Multi-grid representation with field regularization for self-supervised surface reconstruction from point clouds*, Comput. Graph. 114 (2023), pp. 379–386.
[21] J. Kim and Y. Hwang, *Maximum principle-preserving computational algorithm for the 3d high-order Allen–Cahn equation*, Mathematics 13(7) (2025), p. 1085.
[22] D. Lee, *Gradient-descent-like scheme for the allen–cahn equation*, AIP. Adv. 13(8) (2023), p.085010.
[23] C. Lee, S. Ham, Y. Hwang, S. Kwak, and J. Kim, *An explicit fourth-order accurate compact method for the Allen–Cahn equation*, AIMS Math. 9(1) (2024), pp. 735–762.
[24] Y. Li, K. Qin, Q. Xia, and J. Kim, *A second-order unconditionally stable method for the anisotropic dendritic crystal growth model with an orientation-field*, Appl. Numer. Math. 184 (2023), pp. 512–526.
[25] F.-S. Lien, H. Ji, S.D. Ryan, R.C. Ripley, and F. Zhang, *A coupled multigrid solver with wall functions for three-dimensional turbulent flows over urban-like obstacles*, Int. J. Numer. Methods. Fluids. 95(9) (2023), pp. 1349–1371.
[26] T. Liu, J. Yu, Y. Zheng, C. Liu, Y. Yang, and Y. Qi, *A nonlinear multigrid method for the parameter identification problem of partial differential equations with constraints*, Mathematics 10(16) (2022), p. 2938.
[27] N. Margenberg, D. Hartmann, C. Lessig, and T. Richter, *A neural network multigrid solver for the Navier–Stokes equations*, J. Comput. Phys. 460 (2022), p.110983.
[28] J. Rade, A. Jignasu, E. Herron, A. Corpuz, B. Ganapathysubramanian, S. Sarkar, A. Balu, and A. Krishnamurthy, *Deep learning-based 3d multigrid topology optimization of manufacturable designs*, Eng. Appl. Artif. Intell. 126 (2023), p.107033.
[29] D. Sondak, T.M. Smith, R.P. Pawlowski, S. Conde, and J.N. Shadid, *High rayleigh number variational multiscale large eddy simulations of Rayleigh-Bénard convection*, Mech. Res. Commun. 112 (2021), p.103614.
[30] K. Stüben, An introduction to algebraic multigrid, in *Multigrid*, London, 2001. pp. 413–532.
[31] A.V. Trofimov, *Adjoint state method in the block-parametric approach to the inverse problem analysis for elastic layered packages*, Mech. Res. Commun. 147 (2025), p.104426.
[32] U. Trottenberg, C.W. Oosterlee, and A. Schuller, *Multigrid Methods*, Academic Press, 2001.
[33] S. Wang, S. Ma, H. Zhao, and W. Wang, *A multigrid approach for generating harmonic measured foliations*, Comput. Graph. 102 (2022), pp. 380–389.
[34] P. Wesseling, *Introduction to multigrid methods*, Technical report, 1995.
[35] H. Yang, B. Zuo, Z. Wei, H. Luo, and J. Fei, *Geometric multigrid method for isogeometric analysis*, Comput. Model. Eng. Sci. 126(3) (2021), pp. 1033–1052.
[36] B. Zhang, T.G. Vlogman, P. Andric, T.C. Bor, and C.H. Venner, *Local grid refinement in multigrid method for point contact problems of polycrystalline anisotropic material under dry and lubricated conditions*, Friction 10(12) (2022), pp. 2086–2110.