J Sci Comput
DOI 10.1007/s10915-014-9856-7

A Third Order Fast Sweeping Method with Linear Computational Complexity for Eikonal Equations

Liang Wu · Yong-Tao Zhang

Received: 17 September 2013 / Revised: 23 February 2014 / Accepted: 5 April 2014

Abstract Fast sweeping methods are a class of efficient iterative methods for solving steady state hyperbolic PDEs. They utilize the Gauss-Seidel iterations and alternating sweeping strategy to cover a family of characteristics of the hyperbolic PDEs in a certain direction simultaneously in each sweeping order. The first order fast sweeping method for solving Eikonal equations (Zhao in Math Comput 74:603–627, 2005) has linear computational complexity, namely, the computational cost is $O(N)$ where N is the number of grid points of the computational mesh. Recently, a second order fast sweeping method with linear computational complexity was developed in Zhang et al. (SIAM J Sci Comput 33:1873–1896, 2011). The method is based on a discontinuous Galerkin (DG) finite element solver and causality indicators which guide the information flow directions of the nonlinear Eikonal equations. How to extend the method to higher order accuracy is still an open problem, due to the difficulties of solving much more complicated local nonlinear systems and calculations of local causality information. In this paper, we extend previous work and develop a third order fast sweeping method with linear computational complexity for solving Eikonal equations. A novel approach is designed for capturing the causality information in the third order DG local solver. Numerical experiments show that the method has third order accuracy and a linear computational complexity.

Keywords Fast sweeping methods · Discontinuous Galerkin methods · High order accuracy · Linear computational complexity · Static Hamilton–Jacobi equations · Eikonal equations

Mathematics Subject Classification 65M99

L. Wu, Y.-T. Zhang (✉)
Department of Applied and Computational Mathematics and Statistics,
University of Notre Dame, Notre Dame, IN 46556, USA
e-mail: yzhang10@nd.edu

L. Wu
e-mail: lwu2@nd.edu

Published online: 29 April 2014

1 Introduction

In this paper, we continue to develop high order accurate and fast numerical methods to solve the Eikonal equations

$$\begin{cases} \sqrt{\phi_x^2 + \phi_y^2} = f(x, y), & (x, y) \in \Omega \setminus \Gamma, \\ \phi(x, y) = g(x, y), & (x, y) \in \Gamma \subset \Omega, \end{cases} \tag{1.1}$$

where $f(x, y)$ is a positive function and $f(x, y)$ and $g(x, y)$ are Lipschitz continuous, Ω is a computational domain in R^2 and Γ is a subset of Ω. The Eikonal equations form a very important class of static Hamilton–Jacobi equations [3]. The numerical calculations of Eikonal equations appear in many applications, such as optimal control, image processing and computer vision, geometric optics, seismic waves, level set methods, etc.

Challenges in designing efficient and accurate numerical methods for solving (1.1) come from nonlinearity and non-smoothness of its solution. A class of numerical methods is to first discretize (1.1) into a system of nonlinear equations and then design an efficient numerical algorithm to solve the nonlinear system. Among such methods are the fast marching method and the fast sweeping method. The fast marching method [6,17–19,22] is based on the Dijkstra's algorithm [4]. The solution is updated by following the causality in a sequential way; i.e., the solution is updated pointwise in the order that the solution is strictly increasing (decreasing). In the fast sweeping method [1,5,10,11,13,14,21,26–29], Gauss-Seidel iterations with alternating orderings are combined with upwind schemes. In contrast to the fast marching method, the fast sweeping method is an iterative method and follows the causality along characteristics in a parallel way; i.e., all characteristics are divided into a finite number of groups according to their directions and each Gauss-Seidel iteration with a specific sweeping ordering covers a group of characteristics simultaneously. The first order fast sweeping method has a remarkable property that the number of iterations for the convergence is independent of the total number of grid points N [28], so that the computational complexity of the algorithm is $O(N)$.

The first high order fast sweeping method was developed in [27]. It is based on high order finite difference WENO approximations. The method provides a quite general framework, and is easy to incorporate any order of accuracy and any type of numerical Hamiltonian into the framework. For example, the fifth order versions were developed in [16,23]. Much faster convergence speed than that of the time-marching approach can be achieved. However unlike the first order fast sweeping method, the computational complexity of high order WENO fast sweeping methods is more than linear. A possible reason is due to the wide stencil of the high order finite difference approximation to the derivatives, hence some downwind information is used. In order to design high order fast sweeping methods with linear computational complexity, we used discontinuous Galerkin (DG) methods to discretize the Eikonal equations and developed second order fast sweeping methods in [12,24]. In [24], we designed causality indicators which guide the information flow directions for the DG local solver. The values of these indicators are initially provided by the first order finite difference fast sweeping method, and they are updated during iterations along with the solution. The resulting algorithm generates uniform second order accuracy in the L^∞ norm (in smooth regions of the solutions) and the linear computational complexity.

In this paper, we extend previous work and develop a third order fast sweeping method with linear computational complexity for solving Eikonal equations. A novel approach is designed for capturing the causality information in the third order DG local solver. Numerical experiments show that the method has third order accuracy and a linear computational complexity.

The rest of the paper is organized as follows. The detailed algorithm is described in Sect. 2. In Sect. 3 we provide numerical examples to show the uniform third order accuracy and linear computational complexity of the proposed algorithm. Concluding remarks and discussions are given in Sect. 4.

2 A Third Order DG Fast Sweeping Method

In this section, we design a third order DG fast sweeping method with linear computational complexity for solving the Eikonal equations (1.1). In order to simplify the nonlinear algebraic system in the DG local solver, we solve a transformed form of (1.1):

$$\begin{cases} \phi_x^2 + \phi_y^2 = f^2(x, y), & (x, y) \in \Omega \setminus \Gamma, \\ \phi(x, y) = g(x, y), & (x, y) \in \Gamma \subset \Omega. \end{cases} \tag{2.1}$$

Solving this transformed equation also simplifies calculations of local causality information and the inverse Lax-Wendroff procedure for dealing with boundary conditions, as shown in following sections.

We first construct a Cartesian mesh $\Omega_h = \cup_{1\le i\le N, 1\le j\le M} I_{ij}$ covering the computational domain Ω, where $I_{ij} = I_i \times J_j$ and $I_i = [x_{i-1/2}, x_{i+1/2}]$, $J_j = [y_{j-1/2}, y_{j+1/2}]$. The centers of I_i, J_j are denoted by $x_i = \frac{1}{2}(x_{i-1/2} + x_{i+1/2})$ and $y_j = \frac{1}{2}(y_{j-1/2} + y_{j+1/2})$, and the sizes are denoted by $h_i = x_{i+1/2} - x_{i-1/2}$, $l_j = y_{j+1/2} - y_{j-1/2}$. The centers of the cells I_{ij} form a grid $\Theta_h = \{(x_i, y_j),\ 1 \le i \le N,\ 1 \le j \le M\}$. The grid Θ_h is called a dual mesh of Ω_h. The important components of the proposed algorithm are described separately below.

2.1 Initial Causality Determination

In order to achieve fast convergence in the fast sweeping methods, a key step is to reliably determine the causality for the nonlinear Eikonal equation (2.1). For the second order DG fast sweeping method in [24], we determined the causality initially by the first order finite difference fast sweeping method [28]. Following the similar strategy, the second order DG fast sweeping method in [24] can provide initial causality information for our third order DG fast sweeping method. Since both the first order fast sweeping method [28] and the second order DG fast sweeping method [24] have linear computational complexity, their iterations converge very fast and the computational cost to determine the initial causality information is very small.

We identify a cell I_{ij} of Ω_h by its center (x_i, y_j), which is a grid point of Θ_h. Two integer flags are assigned to each cell I_{ij} of Ω_h to indicate the information flow directions. They are called the causality indicators of the cell I_{ij}. These integer values are stored in the causality arrays flagx(i, j) and flagy(i, j), for $1 \le i \le N$, $1 \le j \le M$. For a cell I_{ij}, flagx(i, j)=0 indicates that in the x-direction, the information is propagating from the left neighboring cell $I_{i-1,j}$ to the cell I_{ij}, while flagx(i, j)=1 indicates that the information is propagating from the right neighboring cell $I_{i+1,j}$ to the cell I_{ij}. Similarly, flagy(i, j)=0 indicates that in the y-direction, the information is propagating from the bottom neighboring cell $I_{i,j-1}$ to the cell I_{ij}, while flagy(i, j)=1 indicates that the information is propagating from the top neighboring cell $I_{i,j+1}$ to the cell I_{ij}. If there is no information flowing into I_{ij} from the x or y-direction, then we set the flag of that direction to be 10, i.e., flagx(i, j)=10 or flagy(i, j)=10.

In the second order DG fast sweeping method [24], we perform the first order fast sweeping method [28] on the dual mesh Θ_h till it converges, and record the obtained integer flags in the causality arrays flagx and flagy. These integer flags serve as initial causality indicator values. Here for the third order method, naturally we set the initial causality indicator values to be the final causality indicator values obtained after iterations of the second order DG fast sweeping method [24] converge, namely,

$$\mathsf{flagx\_p2}_{init}(i,j)=\mathsf{flagx\_p1}_{final}(i,j), \qquad \mathsf{flagy\_p2}_{init}(i,j)=\mathsf{flagy\_p1}_{final}(i,j),$$

where flagx_p1, flagy_p1 and flagx_p2, flagy_p2 denote causality arrays of the second order DG (using the piecewise linear P^1 finite element space) and the third order DG (using the piecewise quadratic P^2 finite element space) fast sweeping method respectively. For the simplicity of notations, we use flagx, flagy to denote the causality arrays of the third order DG fast sweeping method for the rest of this paper.

Remark The initial causality arrays of the third order DG fast sweeping method can be provided by the second order DG fast sweeping method based on either the "transformed form" (2.1) or the "original form" (1.1). Correct results for the third order scheme can be obtained for both cases, as shown in Sect. 3.

2.2 The Local Solver

In this subsection, we describe a third order accurate local solver for discretizing the Eikonal equations (2.1) on a general cartesian mesh. This local solver is based on a DG scheme developed for directly solving the time-dependent Hamilton–Jacobi equations [2]. The challenge here is how to incorporate the causality information of the Eikonal equations in the DG scheme and derive relatively simple nonlinear algebraic equations which are suitable to be solved by nonlinear Gauss-Seidel iterations. We follow a similar approach as that in the second order DG fast sweeping method [24], with their differences emphasized.

On the cartesian mesh Ω_h, we define the piecewise quadratic finite element space as

$$V_h^2 = \{v : v|_{I_{ij}} \in P^2(I_{ij}), i = 1, \ldots, N, j = 1, \ldots, M\} \tag{2.2}$$

where $P^2(I_{ij})$ denotes all quadratic polynomials on I_{ij}. The DG scheme for the Eikonal equations (2.1) is defined as: find $\phi_h(x, y) \in V_h^2$, such that

$$\begin{aligned}\int_{I_{ij}} ((\phi_h)_x^2 + (\phi_h)_y^2) v_h(x, y)dxdy &+ \alpha_{l,ij} \int_{y_{j-1/2}}^{y_{j+1/2}} [\phi_h](x_{i-\frac{1}{2}}, y) v_h(x^+_{i-\frac{1}{2}}, y)dy \\ &+\alpha_{b,ij} \int_{x_{i-1/2}}^{x_{i+1/2}} [\phi_h](x, y_{j-\frac{1}{2}}) v_h(x, y^+_{j-\frac{1}{2}})dx \\ &+\alpha_{r,ij} \int_{y_{j-1/2}}^{y_{j+1/2}} [\phi_h](x_{i+\frac{1}{2}}, y) v_h(x^-_{i+\frac{1}{2}}, y)dy \\ &+\alpha_{t,ij} \int_{x_{i-1/2}}^{x_{i+1/2}} [\phi_h](x, y_{j+\frac{1}{2}}) v_h(x, y^-_{j+\frac{1}{2}})dx\end{aligned}$$

$$= \int_{I_{ij}} f^2(x,y) v_h(x,y) dxdy, \ i = 1, \cdots, N, \ j = 1, \cdots, M \tag{2.3}$$

holds for any $v_h \in V_h^2$. Here the quadratic polynomial $\phi_h(x,y)$ on I_{ij} can be represented by

$$\phi_h|_{I_{ij}} = \phi_{ij} + u_{ij}\xi_i + v_{ij}\eta_j + a_{ij}\xi_i^2 + b_{ij}\eta_j^2 + c_{ij}\xi_i\eta_j, \tag{2.4}$$

where $\xi_i = \frac{2}{h_i}(x - x_i)$, $\eta_j = \frac{2}{l_j}(y - y_j)$, and $\xi_i, \eta_j \in [-1, 1]$ on each cell I_{ij}. There are six unknowns on each cell I_{ij}: $\phi_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij}$ and c_{ij}. $[\phi_h]$ denotes the jump of ϕ_h across the cell interface, for example,

$$\begin{aligned} [\phi_h](x_{i-\frac{1}{2}}, y) \; &= \; (\phi_{ij} - \phi_{i-1,j} - u_{ij} - u_{i-1,j} + a_{ij} - a_{i-1,j}) \\ &\quad + (v_{ij} - c_{ij} - v_{i-1,j} - c_{i-1,j})\eta_j + (b_{ij} - b_{i-1,j})\eta_j^2, \end{aligned} \tag{2.5}$$

for $y \in [y_{j-1/2}, y_{j+1/2}]$. And

$$v_h(x^+_{i-\frac{1}{2}}, y) \triangleq \lim_{x \to x_{i-1/2}, x > x_{i-1/2}} v_h(x,y), \tag{2.6}$$

$$v_h(x^-_{i+\frac{1}{2}}, y) \triangleq \lim_{x \to x_{i+1/2}, x < x_{i+1/2}} v_h(x,y) \tag{2.7}$$

for $y \in [y_{j-1/2}, y_{j+1/2}]$,

$$v_h(x, y^+_{j-\frac{1}{2}}) \triangleq \lim_{y \to y_{j-1/2}, y > y_{j-1/2}} v_h(x,y), \tag{2.8}$$

$$v_h(x, y^-_{j+\frac{1}{2}}) \triangleq \lim_{y \to y_{j+1/2}, y < y_{j+1/2}} v_h(x,y) \tag{2.9}$$

for $x \in [x_{i-1/2}, x_{i+1/2}]$. $\alpha_{l,ij}, \alpha_{b,ij}, \alpha_{r,ij}, \alpha_{t,ij}$ are local constants which depend on the numerical solutions in the neighboring cells of I_{ij} and the causality of the Eikonal equation. They are called local causality constants and will be discussed in detail in Sect. 2.2.1.

Remark Different ways to calculate local causality constants lead to different accuracy and convergence behaviour of the schemes in [12,24], and the scheme in this paper. "Local causality constants" play a key role in incorporating the causality information of the Eikonal equations. The construction of the causality indicators and the calculation of the local causality constants are motivated by the idea of upwind schemes for solving hyperbolic conservation laws and the iterative framework of fast sweeping methods. The local causality constants reflect the "local" upwind information of the Eikonal equations around a target cell. All different ways to calculate local causality constants follow this principle to acheive the convergence of the iterations. However, the speed of convergence is affected by how accurately the "local" upwind information is captured in different ways.

2.2.1 Calculations of Local Causality Constants

Denote $\hat{H} \triangleq (\phi_h)_x^2 + (\phi_h)_y^2$, $\hat{H}_1 \triangleq \frac{\partial \hat{H}}{\partial(\phi_h)_x} = 2(\phi_h)_x$ and $\hat{H}_2 \triangleq \frac{\partial \hat{H}}{\partial(\phi_h)_y} = 2(\phi_h)_y$. The local causality constants $\alpha_{l,ij}, \alpha_{b,ij}, \alpha_{r,ij}, \alpha_{t,ij}$ are approximations of $\hat{H}_1(\nabla\phi_h)$ and $\hat{H}_2(\nabla\phi_h)$ in the four neighboring cells of I_{ij}. The calculation of the local causality constants needs the causality information of the current iteration step. However due to the nonlinearity of the Eikonal equations, the causality information (i.e., causality indicator values) is unknown beforehand. On the other hand, the iterative framework of fast sweeping methods allows us to initially estimate the causality indicator values, and then iterate them along with the iteration of the solution itself of the DG scheme. Here for the third order DG scheme (2.3), the values of causality indicators are determined initially by the second order DG fast sweeping iterations (Sect. 2.1), and they are updated in the iterations. How to update causality indicators will be described in detail in Sect. 2.2.3.

The local causality constant values are calculated according to the values of causality indicators in the current iteration step. First, let us recall some details of the local causality constants in [24]. For example, $\alpha_{l,ij}$ carries causality information from the left neighboring cell $I_{i-1,j}$ in the x-direction if there is causality information coming in from that direction. If the x-direction causality indicator $\mathsf{flagx(i,j)} = 1$, it indicates that the information comes in from the right neighboring cell $I_{i+1,j}$ in the x-direction. If $\mathsf{flagx(i,j)} = 10$, then it indicates that there is no information coming into the cell I_{ij} in the x-direction. So for these two cases, $\alpha_{l,ij}$ should not carry any causality information and it will be set to 0. If $\mathsf{flagx(i,j)} = 0$, it indicates that information may flow in from the cell $I_{i-1,j}$. In this case, $\hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}}$ is calculated. However, different from the second order scheme in [24], here ϕ_h is a quadratic polynomial on $I_{i-1,j}$. Hence $\hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}}$ is not a constant as that in [24], but a linear function

$$\hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}} = \frac{\partial \hat{H}}{\partial(\phi_h)_x}\Big|_{I_{i-1,j}} = 2(\phi_h)_x|_{I_{i-1,j}} = \frac{4u_{i-1,j}}{h_{i-1}} + \frac{8a_{i-1,j}}{h_{i-1}}\cdot\xi_{i-1} + \frac{4c_{i-1,j}}{h_{i-1}}\cdot\eta_j. \tag{2.10}$$

Since $\alpha_{l,ij}$ carries the causality information from the left into the cell I_{ij}, we set $\alpha_{l,ij}$ to be the value of $\hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}}$ at $\xi_{i-1} = 1,\ \eta_j = 0$, which is the middle point $(x = x_{i-\frac{1}{2}},\ y = y_j)$ of the cell boundary that I_{ij} and $I_{i-1,j}$ share. The motivation of evaluating $\hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}}$ at the middle point of the shared cell boundary is based on the observation that the information flows into the cell I_{ij} from the shared cell boundary and the middle point is the center of this inflow boundary. In summary, the formula to calculate $\alpha_{l,ij}$ is

$$\alpha_{l,ij} = \begin{cases} \max(0, \hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}}) = \max\left(0, \dfrac{4u_{i-1,j}}{h_{i-1}} + \dfrac{8a_{i-1,j}}{h_{i-1}}\right), & \text{If } \mathsf{flagx(i,j)} = 0; \\ 0, \qquad \text{If } \mathsf{flagx(i,j)} = 1 \text{ or } \mathsf{flagx(i,j)} = 10. \end{cases} \tag{2.11}$$

Here $\alpha_{l,ij}$ has a much simpler expression than the one in [24]. This is one of benefits of solving the squared form (2.1) of Eikonal equations in stead of the original one (1.1). For the first case, if we have $\max(0, \hat{H}_1(\nabla\phi_h)|_{I_{i-1,j}}) = 0$, then we need to correct the current causality indicator $\mathsf{flagx(i,j)}$ to be $\mathsf{flagx(i,j)=10}$. By doing this, we shut down this horizontal information flow direction of the cell I_{ij} in the current iteration, and wait for correct causality information on the cell $I_{i-1,j}$ to be ready in next iterations. We observe that this situation usually happens at the early stage of the iteration process, and near shock locations where the characteristics from the left intersect with the characteristics from the right in numerical experiments. Likewise,

$$\alpha_{r,ij} = \begin{cases} \min(0, \hat{H}_1(\nabla\phi_h)|_{I_{i+1,j}}) = \min\left(0, \dfrac{4u_{i+1,j}}{h_{i+1}} - \dfrac{8a_{i+1,j}}{h_{i+1}}\right), & \text{If } \mathsf{flagx(i,j)} = 1; \\ 0, \qquad \text{If } \mathsf{flagx(i,j)} = 0 \text{ or } \mathsf{flagx(i,j)} = 10. \end{cases} \tag{2.12}$$

For the first case, if we have $\min(0, \hat{H}_1(\nabla\phi_h)|_{I_{i+1,j}}) = 0$, then we need to correct the current causality indicator flagx(i,j) to be flagx(i,j)=10. Similarly,

$$\alpha_{b,ij} = \begin{cases} \max(0, \hat{H}_2(\nabla\phi_h)|_{I_{i,j-1}}) = \max\left(0, \dfrac{4v_{i,j-1}}{l_{j-1}} + \dfrac{8b_{i,j-1}}{l_{j-1}}\right), & \text{If flagy(i, j)} = 0; \\ 0, \qquad \text{If flagy(i, j)} = 1 \text{ or flagy(i, j)} = 10. \end{cases} \tag{2.13}$$

For the first case, if we have $\max(0, \hat{H}_2(\nabla\phi_h)|_{I_{i,j-1}}) = 0$, then we need to correct the current causality indicator flagy(i,j) to be flagy(i,j)=10. Finally,

$$\alpha_{t,ij} = \begin{cases} \min(0, \hat{H}_2(\nabla\phi_h)|_{I_{i,j+1}}) = \min\left(0, \dfrac{4v_{i,j+1}}{l_{j+1}} - \dfrac{8b_{i,j+1}}{l_{j+1}}\right), & \text{If flagy(i, j)} = 1; \\ 0, \qquad \text{If flagy(i, j)} = 0 \text{ or flagy(i, j)} = 10. \end{cases} \tag{2.14}$$

For the first case, if we have $\min(0, \hat{H}_2(\nabla\phi_h)|_{I_{i,j+1}}) = 0$, then we need to correct the current causality indicator flagy(i,j) to be flagy(i,j)=10. If both flagx(i,j)=10 and flagy(i,j)=10, we will skip the current cell in the current iteration.

To calculate local causality constants for the second order scheme based on the transformed form (2.1), we can just simply take the coefficients of quadratic terms (i.e., all a's and b's) in (2.11)–(2.14) to be zeros. The resulting formulas are much simpler than those based on the original form (1.1) in [24].

Remark We emphasize the difference of calculating local causality constants between the third order scheme in this paper and the second order scheme in [24]. The local causality constants are approximations of $\hat{H}_1(\nabla\phi_h)$ and $\hat{H}_2(\nabla\phi_h)$, and they carry the causality / upwind information of the Eikonal equations. In the second order scheme [24], direct evaluations of $\hat{H}_1(\nabla\phi_h)$ and $\hat{H}_2(\nabla\phi_h)$ give constants, so the calculations are trival. However for the third order scheme, $\hat{H}_1(\nabla\phi_h)$ and $\hat{H}_2(\nabla\phi_h)$ are linear functions. We evaluate the linear functions at the middle point of shared cell boundaries, and use them as local causality constants. Numerical experiments in Sect. 3 verify that this approach can corretly capture the causality information and guarantee fast convergence of the iterations.

2.2.2 The Local Nonlinear System

By taking $v_h = 1, \xi_i, \eta_j, \xi_i^2, \eta_j^2, \xi_i\eta_j$ on any given element I_{ij}, the DG formulation (2.3) is converted from the integral form to a 6×6 nonlinear system for six unknowns: ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij}. In order to solve this nonlinear system, we adopt the Gauss-Seidel philosophy, i.e., we use the current numerical values of neighboring cells of I_{ij}. Denoting $k \triangleq l_j/h_i$ ($k = 1$ for a uniform mesh), and leaving all unknowns for the cell I_{ij} to the left hand side and moving the rest terms, including numerical values of neighboring cells of I_{ij}, to the right hand side of the nonlinear system, we have the following six local equations

1.

$$\begin{aligned} &4k\, u_{ij}^2 + \frac{4}{k}\, v_{ij}^2 + \frac{16k}{3}\, a_{ij}^2 + \frac{16}{3k}\, b_{ij}^2 + \left(\frac{4k}{3} + \frac{4}{3k}\right) c_{ij}^2 \\ &\quad + \big(l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\big)\phi_{ij} + l_j(-\alpha_l - \alpha_r)u_{ij} + h_i(-\alpha_b - \alpha_t)v_{ij} \\ &\quad + \big(l_j(\alpha_l - \alpha_r) + \frac{1}{3}h_i(\alpha_b - \alpha_t)\big)a_{ij} + \left(\frac{1}{3}l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\right) b_{ij} \end{aligned}$$

$$
\begin{aligned}
&= \int\limits_{I_{ij}} f^2(x,y)\, dxdy - \Big(- l_j\alpha_l\phi_{i-1,j} - h_i\alpha_b\phi_{i,j-1} + l_j\alpha_r\phi_{i+1,j} + h_i\alpha_t\phi_{i,j+1} \\
&\qquad - l_j\alpha_l u_{i-1,j} - l_j\alpha_r u_{i+1,j} - h_i\alpha_b v_{i,j-1} - h_i\alpha_t v_{i,j+1} \\
&\qquad - l_j\alpha_l a_{i-1,j} - \frac{1}{3}h_i\alpha_b a_{i,j-1} + l_j\alpha_r a_{i+1,j} + \frac{1}{3}h_i\alpha_t a_{i,j+1} \\
&\qquad - \frac{1}{3}l_j\alpha_l b_{i-1,j} - h_i\alpha_b b_{i,j-1} + \frac{1}{3}l_j\alpha_r b_{i+1,j} + h_i\alpha_t b_{i,j+1}\Big);
\end{aligned}
\tag{2.15}
$$

2.

$$
\begin{aligned}
&\frac{16k}{3}\, a_{ij}u_{ij} + \frac{8}{3k}\, c_{ij}v_{ij} \\
&\quad + l_j(-\alpha_l - \alpha_r)\phi_{ij} + \Big(l_j(\alpha_l - \alpha_r) + \frac{1}{3}h_i(\alpha_b - \alpha_t)\Big) u_{ij} \\
&\quad + l_j(-\alpha_l - \alpha_r)a_{ij} + \frac{1}{3}l_j(-\alpha_l - \alpha_r)b_{ij} + \frac{1}{3}h_i(-\alpha_b - \alpha_t)c_{ij} \\
&= \int\limits_{I_{ij}} f^2(x,y)\xi_i\, dxdy - \Big(l_j\alpha_l\phi_{i-1,j} + l_j\alpha_r\phi_{i+1,j} \\
&\quad + l_j\alpha_l u_{i-1,j} - \frac{1}{3}h_i\alpha_b u_{i,j-1} - l_j\alpha_r u_{i+1,j} + \frac{1}{3}h_i\alpha_t u_{i,j+1} \\
&\quad + l_j\alpha_l a_{i-1,j} + l_j\alpha_r a_{i+1,j} + \frac{1}{3}l_j\alpha_l b_{i-1,j} + \frac{1}{3}l_j\alpha_r b_{i+1,j} \\
&\quad - \frac{1}{3}h_i\alpha_b c_{i,j-1} - \frac{1}{3}h_i\alpha_t c_{i,j+1}\Big);
\end{aligned}
\tag{2.16}
$$

3.

$$
\begin{aligned}
&\frac{16}{3k}\, b_{ij}v_{ij} + \frac{8k}{3}\, c_{ij}u_{ij} \\
&\quad + h_i(-\alpha_b - \alpha_t)\phi_{ij} + \Big(\frac{1}{3}l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\Big) v_{ij} \\
&\quad + \frac{1}{3}h_i(-\alpha_b - \alpha_t)a_{ij} + h_i(-\alpha_b - \alpha_t)b_{ij} + \frac{1}{3}l_j(-\alpha_l - \alpha_r)c_{ij} \\
&= \int\limits_{I_{ij}} f^2(x,y)\eta_j\, dxdy - \Big(h_i\alpha_b\phi_{i,j-1} + h_i\alpha_t\phi_{i,j+1} \\
&\quad - \frac{1}{3}l_j\alpha_l v_{i-1,j} + h_i\alpha_b v_{i,j-1} + \frac{1}{3}l_j\alpha_r v_{i+1,j} - h_i\alpha_t v_{i,j+1} \\
&\quad + \frac{1}{3}h_i\alpha_b a_{i,j-1} + \frac{1}{3}h_i\alpha_t a_{i,j+1} + h_i\alpha_b b_{i,j-1} + h_i\alpha_t b_{i,j+1} \\
&\quad - \frac{1}{3}l_j\alpha_l c_{i-1,j} - \frac{1}{3}l_j\alpha_r c_{i+1,j}\Big);
\end{aligned}
\tag{2.17}
$$

4.

$$
\begin{aligned}
&\frac{4k}{3}\, u_{ij}^2 + \frac{4}{3k}\, v_{ij}^2 + \frac{16k}{5}\, a_{ij}^2 + \frac{16}{9k}\, b_{ij}^2 + (\frac{4k}{9} + \frac{4}{5k})\, c_{ij}^2 \\
&\quad + \Big(l_j(\alpha_l - \alpha_r) + \frac{1}{3}h_i(\alpha_b - \alpha_t)\Big)\phi_{ij} + l_j(-\alpha_l - \alpha_r)u_{ij} + \frac{1}{3}h_i(-\alpha_b - \alpha_t)v_{ij}
\end{aligned}
$$

$$
\begin{aligned}
&+ \big(l_j(\alpha_l-\alpha_r)+\frac{1}{5}h_i(\alpha_b-\alpha_t)\big)a_{ij}+\frac{1}{3}\big(l_j(\alpha_l-\alpha_r)+h_i(\alpha_b-\alpha_t)\big)b_{ij}\\
&= \int_{I_{ij}} f^2(x,y)\xi_i^2\,dxdy-\Big(-l_j\alpha_l\phi_{i-1,j}-\frac{1}{3}h_i\alpha_b\phi_{i,j-1}+l_j\alpha_r\phi_{i+1,j}+\frac{1}{3}h_i\alpha_t\phi_{i,j+1}\\
&-l_j\alpha_l u_{i-1,j}-l_j\alpha_r u_{i+1,j}-\frac{1}{3}h_i\alpha_b v_{i,j-1}-\frac{1}{3}h_i\alpha_t v_{i,j+1}\\
&-l_j\alpha_l a_{i-1,j}-\frac{1}{5}h_i\alpha_b a_{i,j-1}+l_j\alpha_r a_{i+1,j}+\frac{1}{5}h_i\alpha_t a_{i,j+1}\\
&-\frac{1}{3}l_j\alpha_l b_{i-1,j}-\frac{1}{3}h_i\alpha_b b_{i,j-1}+\frac{1}{3}l_j\alpha_r b_{i+1,j}+\frac{1}{3}h_i\alpha_t b_{i,j+1}\Big); \qquad (2.18)
\end{aligned}
$$

5.

$$
\begin{aligned}
&\frac{4k}{3}\,u_{ij}^2+\frac{4}{3k}\,v_{ij}^2+\frac{16k}{9}\,a_{ij}^2+\frac{16}{5k}\,b_{ij}^2+\left(\frac{4k}{5}+\frac{4}{9k}\right)\,c_{ij}^2\\
&+\left(\frac{1}{3}l_j(\alpha_l-\alpha_r)+h_i(\alpha_b-\alpha_t)\right)\phi_{ij}+\frac{1}{3}l_j(-\alpha_l-\alpha_r)u_{ij}+h_i(-\alpha_b-\alpha_t)v_{ij}\\
&+\frac{1}{3}\,\big(l_j(\alpha_l-\alpha_r)+h_i(\alpha_b-\alpha_t)\big)\,a_{ij}+\left(\frac{1}{5}l_j(\alpha_l-\alpha_r)+h_i(\alpha_b-\alpha_t)\right)b_{ij}\\
&=\int_{I_{ij}} f^2(x,y)\eta_j^2\,dxdy-\Big(-\frac{1}{3}l_j\alpha_l\phi_{i-1,j}-h_i\alpha_b\phi_{i,j-1}+\frac{1}{3}l_j\alpha_r\phi_{i+1,j}+h_i\alpha_t\phi_{i,j+1}\\
&-\frac{1}{3}l_j\alpha_l u_{i-1,j}-\frac{1}{3}l_j\alpha_r u_{i+1,j}-h_i\alpha_b v_{i,j-1}-h_i\alpha_t v_{i,j+1}\\
&-\frac{1}{3}l_j\alpha_l a_{i-1,j}-\frac{1}{3}h_i\alpha_b a_{i,j-1}+\frac{1}{3}l_j\alpha_r a_{i+1,j}+\frac{1}{3}h_i\alpha_t a_{i,j+1}\\
&-\frac{1}{5}l_j\alpha_l b_{i-1,j}-h_i\alpha_b b_{i,j-1}+\frac{1}{5}l_j\alpha_r b_{i+1,j}+h_i\alpha_t b_{i,j+1}\Big); \qquad (2.19)
\end{aligned}
$$

6.

$$
\begin{aligned}
&\frac{16k}{9}\,a_{ij}c_{ij}+\frac{16}{9k}\,b_{ij}c_{ij}\\
&+\frac{1}{3}h_i(-\alpha_b-\alpha_t)u_{ij}+\frac{1}{3}l_j(-\alpha_l-\alpha_r)v_{ij}\\
&+\frac{1}{3}\big(l_j(\alpha_l-\alpha_r)+h_i(\alpha_b-\alpha_t)\big)c_{ij}\\
&=\int_{I_{ij}} f^2(x,y)\xi_i\eta_j\,dxdy-\Big(\frac{1}{3}h_i\alpha_b u_{i,j-1}+\frac{1}{3}h_i\alpha_t u_{i,j+1}+\frac{1}{3}l_j\alpha_l v_{i-1,j}+\frac{1}{3}l_j\alpha_r v_{i+1,j}\\
&+\frac{1}{3}l_j\alpha_l c_{i-1,j}-\frac{1}{3}l_j\alpha_r c_{i+1,j}+\frac{1}{3}h_i\alpha_b c_{i,j-1}-\frac{1}{3}h_i\alpha_t c_{i,j+1}\Big). \qquad (2.20)
\end{aligned}
$$

Note that for simplicity, we omit the subscript ij of local causality constants in these equations with the understanding that they are values on the cell I_{ij}. In a vector form, we have the nonlinear system

$$\vec{f}(\vec{x})=\vec{0}, \qquad (2.21)$$

where $\vec{x}=(\phi_{ij},\,u_{ij},\,v_{ij},\,a_{ij},\,b_{ij},\,c_{ij})^T$, and $\vec{f}=(f_1,\,f_2,\,f_3,\,f_4,\,f_5,\,f_6)^T$ corresponding to the six Eqs. (2.15)–(2.20). The Newton's method is used to solve this local nonlinear system.

Notice that here we have two kind of iterations. One is the outer fast sweeping iteration, and the other is the Newton iteration for the local nonlinear system of every cell I_{ij} inside a fast sweeping iteration. So each fast sweeping iteration consists of many inner Newton iteration procedures. In the $k-$th fast sweeping iteration, the Newton iteration procedure to solve the local nonlinear system (2.21) for the cell I_{ij} is as following.

1. The initial Newton iteration values $\vec{x}^{(0)} = (\phi_{ij}^{\text{old}},\ u_{ij}^{\text{old}},\ v_{ij}^{\text{old}},\ a_{ij}^{\text{old}},\ b_{ij}^{\text{old}},\ c_{ij}^{\text{old}})^T$ are the values of the last (i.e., the $(k-1)$-th) fast sweeping iteration step.
2. At the Newton iteration step $n+1$, we calculate the Jacobian matrix $J(\vec{x}^{(n)}) = \vec{f}'(\vec{x}^{(n)})$ (see Appendix for detailed formulae of the Jacobian matrix) and solve the 6×6 linear system $J(\vec{x}^{(n)}) \cdot \Delta x^{(n)} = -\vec{f}(\vec{x}^{(n)})$, then update $\vec{x}^{(n+1)} = \vec{x}^{(n)} + \Delta x^{(n)}$.
3. Repeat the step 2 until $\|\vec{x}^{(n+1)} - \vec{x}^{(n)}\|_\infty < 10^{-11}$ or stop the Newton iteration if this convergence criterion can not be satisfied in 100 iterations. If the Newton iteration converges (i.e., the convergence criterion is satisfied), then the local nonlinear system for the cell I_{ij} has been solved and we update the values of $(\phi_{ij},\ u_{ij},\ v_{ij},\ a_{ij},\ b_{ij},\ c_{ij})$. Otherwise, we keep the old values on the cell I_{ij} from the last fast sweeping iteration step, and may update them in the next fast sweeping iteration step.

Via numerical experiments, we observe that the non-convergence of the Newton iteration for the local nonlinear system mainly happens around shock locations where the characteristics from the left intersect with the characteristics from the right. However, along with the fast sweeping iterations, Newton iterations for local nonlinear systems of all cells converge and DG solutions are obtained.

Remark To obtain the local nonlinear system for the second order scheme based on the transformed form (2.1), we can just simply take the coefficients of quadratic terms (i.e., all a's, b's and c's) in (2.15)–(2.17) to be zeros. The resulting quadratic system is simple enough to be solved directly as that in [24], without using the Newton iteration.

2.2.3 Update of Causality Arrays

If the values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} have been updated by the third order DG local solver, then we need to make the current values of causality indicators in the neighboring cells of I_{ij} consistent with the current information flow directions determined by the third order DG local solver. As that in [24], we need to consider the causality information on neighboring cells of the cells whose causality arrays may be updated.

Again, as that in the calculation of local causality constants, there is a difference here from the second order method in [24]. The information flow directions of a DG solution on the cell I_{ij} are indicated by $\hat{H}_1(\nabla\phi_h)|_{I_{i,j}}$ and $\hat{H}_2(\nabla\phi_h)|_{I_{i,j}}$. In the second order method [24], they are constants, and their signs directly indicate the causality information flow directions. However, here we have a quadratic polynomial ϕ_h on I_{ij} for the third order method, $\hat{H}_1(\nabla\phi_h)|_{I_{i,j}}$ and $\hat{H}_2(\nabla\phi_h)|_{I_{i,j}}$ are linear functions. We evaluate $\hat{H}_1(\nabla\phi_h)|_{I_{i,j}}$ and $\hat{H}_2(\nabla\phi_h)|_{I_{i,j}}$ at the middle points of the shared cell boundaries of I_{ij} and its four neighboring cells, and use the signs of the obtained values to indicate the causality information flow directions. The detailed algorithm is described in the following.

In the $x-$direction, $\mathsf{if}(u_{ij} + 2a_{ij} > 0\ \mathsf{.and.} i < N)$: this indicates that the information in the cell (i, j) is propagating to the right cell (i+1, j), and it is possible that we need to update $\mathsf{flagx}(\mathsf{i}+1, \mathsf{j})$. If the cell (i+1, j) is a boundary cell (i.e. a cell around Γ which has preassigned values and these values are fixed during iterations), then we do *not* need to update $\mathsf{flagx}(\mathsf{i}+1, \mathsf{j})$. Otherwise we need to look at the causality information at the right hand side of the cell (i+1, j). If the cell (i+1, j) happens to be at the boundary of the computational domain, then there is no causality information at the right hand side of the cell (i+1, j), and we just update $\mathsf{flagx}(i+1, j) = 0$. If the cell (i+1, j) is an interior cell, then there is causality information at its right neighboring cell (i+2, j) which we need to consider. Following [24], we should update $\mathsf{flagx}(i+1, j)$ if and only if the current numerical values on cell (i+2, j) have been provided by the third order DG local solver (i.e., not the initial iteration values from the second order scheme), and the "global" causality between the cell (i, j) and the cell (i+2, j) is consistent with the current "local" causality for the cell (i+1, j). Here the current "local" causality is just the information propagation direction indicated by the third order DG solution in the current iteration step and current cell. In this case, it is indicated by $u_{ij} + 2a_{ij} > 0$. The "global" causality between the cell (i, j) and the cell (i+2, j) is motivated by the "first arrival time" used in the first order fast sweeping method [28]. In the Eikonal equations (1.1), $\phi(x, y)$ has the physical meaning that it represents the "first arrival time" which the waves take from the boundary Γ to a point (x, y) in the domain, and $f(x, y)$ is the reciprocal of the wave velocity at the point (x, y). Hence the "first arrival time" from the middle point of the shared cell boundary of I_{ij} and $I_{i+1,j}$ to the center of the cell $I_{i+1,j}$ can be approximated by $\phi_{ij} + u_{ij} + a_{ij} + \frac{1}{2}h_{i+1} f_{i+1,j}$, where $f_{i+1,j} = f(x_{i+1}, y_j)$. And the "first arrival time" from the middle point of the shared cell boundary of $I_{i+1,j}$ and $I_{i+2,j}$ to the center of the cell $I_{i+1,j}$ can be approximated by $\phi_{i+2,j} - u_{i+2,j} + a_{i+2,j} + \frac{1}{2}h_{i+1} f_{i+1,j}$. If $\phi_{ij} + u_{ij} + a_{ij} + \frac{1}{2}h_{i+1} f_{i+1,j} < \phi_{i+2,j} - u_{i+2,j} + a_{i+2,j} + \frac{1}{2}h_{i+1} f_{i+1,j}$, then the causality information is flowing from the cell (i,j) into the cell (i+2, j) "globally". In summary, with the cancellation of the common term, we have

```
if ((the values on cell $(i+2, j)$ are from the third order DG solver) .and.
    $\phi_{ij} + u_{ij} + a_{ij} < \phi_{i+2,j} - u_{i+2,j} + a_{i+2,j}$), then
    $\mathsf{flagx}(i+1, j) = 0$;
```

otherwise, we do *not* update $\mathsf{flagx}(i+1, j)$. We would like to point out the reason that we need the current numerical values on cell (i+2, j) to be provided by the third order DG local solver. This is because the numerical values on the cell (i,j) have been provided by the third order DG local solver in the current iteration. So we request the similar type of numerical information on the cell (i+2,j) in order to be consistent in computing the "global" causality.

Similarly if$(u_{ij} - 2a_{ij} < 0$ .and.$i > 1)$: this indicates that the information in the cell (i, j) is propagating to the left cell (i−1, j) and it is possible that we need to update $\mathsf{flagx}(i-1, j)$. If the cell (i−1, j) is a boundary cell, then we do *not* need to update $\mathsf{flagx}(i-1, j)$. Otherwise, if the cell (i−1, j) happens to be at the boundary of the computational domain, we update $\mathsf{flagx}(i-1, j) = 1$. If the cell (i−1, j) is an interior cell, then there is causality information at its left neighboring cell (i−2, j) which we need to consider.

```
if ((the values on cell $(i-2, j)$ are from the third order DG solver) .and.
    $\phi_{ij} - u_{ij} + a_{ij} < \phi_{i-2,j} + u_{i-2,j} + a_{i-2,j}$), then
    $\mathsf{flagx}(i-1, j) = 1$;
```

otherwise we do *not* update $\mathsf{flagx}(i-1, j)$.

Cases in the y-direction are similar. In the y-direction, if$(v_{ij} + 2b_{ij} > 0$ .and.$j < M)$: this indicates that the information in the cell (i, j) is propagating to the top cell (i, j+1), and it is possible that we need to update $\mathsf{flagy}(i, j+1)$. If the cell (i, j+1) is a boundary cell, then we do *not* need to update $\mathsf{flagy}(i, j+1)$. Otherwise, if the cell (i, j+1) happens to be at the boundary of the computational domain, we update $\mathsf{flagy}(i, j+1) = 0$. If the cell (i, j+1) is an interior cell, then there is causality information at its top neighboring cell (i, j+2) which we need to consider.

```
if ((the values on cell (i, j + 2) are from the third order DG solver) .and.
  $\phi_{ij} + v_{ij} + b_{ij} < \phi_{i,j+2} - v_{i,j+2} + b_{i,j+2}$), then
  flagy(i, j + 1) = 0;
```

otherwise we do *not* update flagy(i, j + 1).

If($v_{ij} - 2b_{ij} < 0$ .and.$j > 1$): this indicates that the information in the cell (i, j) is propagating to the bottom cell (i, j−1), and it is possible that we need to update flagy(i, j − 1). If the cell (i, j−1) is a boundary cell, then we do *not* need to update flagy(i, j − 1). Otherwise, if the cell (i, j−1) happens to be at the boundary of the computational domain, we update flagy(i, j − 1) = 1. If the cell (i, j−1) is an interior cell, then there is causality information at its bottom neighboring cell (i, j−2) which we need to consider.

```
if ((the values on cell (i, j − 2) are from the third order DG solver) .and.
  $\phi_{ij} - v_{ij} + b_{ij} < \phi_{i,j-2} + v_{i,j-2} + b_{i,j-2}$), then
  flagy(i, j − 1) = 1;
```

otherwise we do *not* update flagy(i, j − 1).

2.2.4 Initialization of the DG Local Solver and Boundary Conditions

Cells in the computational domain are classified as "boundary cells" or "non-boundary cells". Boundary cells are around the boundary Γ and the values on them are fixed during iterations. For this third order DG fast sweeping method, on non-boundary cells the initial iteration values of ϕ_{ij}, u_{ij}, v_{ij} are the values from the second order DG fast sweeping iteration, and the initial iteration values of a_{ij}, b_{ij} and c_{ij} are zeros. And we use L^2 projection of the exact or approximating boundary values to pre-assign the values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} on boundary cells. The formulae resulting from the L^2 projection of a function ϕ on the boundary cell I_{ij} are

$$\phi_{ij} = A - \frac{15}{4}\left(B + C - \frac{2}{3}A\right), \tag{2.22}$$

$$u_{ij} = \frac{3}{h_i l_j}\int_{I_{ij}} \phi \cdot \xi_i \, dxdy, \tag{2.23}$$

$$v_{ij} = \frac{3}{h_i l_j}\int_{I_{ij}} \phi \cdot \eta_j \, dxdy, \tag{2.24}$$

$$a_{ij} = \frac{45}{4}\left(B - \frac{1}{3}A\right), \tag{2.25}$$

$$b_{ij} = \frac{45}{4}\left(C - \frac{1}{3}A\right), \tag{2.26}$$

$$c_{ij} = \frac{9}{h_i l_j}\int_{I_{ij}} \phi \cdot \xi_i \eta_j \, dxdy, \tag{2.27}$$

where

$$A = \frac{1}{h_i l_j} \int\limits_{I_{ij}} \phi \, dxdy, \quad B = \frac{1}{h_i l_j} \int\limits_{I_{ij}} \phi \cdot \xi_i^2 \, dxdy, \quad C = \frac{1}{h_i l_j} \int\limits_{I_{ij}} \phi \cdot \eta_j^2 \, dxdy.$$

The integrals here are numerically approximated by an 8-point Gaussian quadrature rule. Gaussian quadrature points are four corners and four middle points of cell boundaries on a boundary cell. Hence the numerical values of ϕ are needed at these Gaussian quadrature points of boundary cells around the inflow boundary Γ in order to perform the L^2 projection (2.22)–(2.27). In applications, if these points of boundary cells are not on the boundary Γ and numerical values on them can not be directly provided by the boundary condition in the Eq. (1.1), the Richardson extrapolation or inverse Lax-Wendroff procedure developed in [8] can provide accurate approximations to these values. In the Richardson extrapolation procedure, first order accurate solutions on several locally successively refined meshes are used to obtain high order approximations to numerical values at the points of boundary cells. The Richardson extrapolation procedure is suitable for different types of inflow boundaries, especially for the point source boundary. In the inverse Lax-Wendroff procedure, the PDE itself and the given boundary condition are repeatedly used to obtain high order approximations to the numerical values for the points near the boundary. More details can be found in [8] and its application to fifth order WENO fast sweeping method in [23]. It can also be applied to complex domains and other hyperbolic PDEs, as shown in [20]. In this paper, our major focus is on the development of the third order fast sweeping method itself. Hence in most of numerical experiments of the next section, we directly use exact solutions to specify the values at Gaussian quadrature points of boundary cells. The inverse Lax-Wendroff procedure is tested in one of the examples in order to demonstrate its performance.

2.3 Algorithm Summary

Now we summarize the third order DG fast sweeping method in the following.

1. Use the final causality arrays provided by the second order DG fast sweeping method in [24] as the initial causality arrays for this third order DG fast sweeping method. See the description in Subsect. 2.1.
2. Initialize the third order DG local solver as described in Subsect. 2.2.4.
3. Perform iterations on non-boundary cells with four alternating direction sweepings. In each sweeping, use the procedure described in Subsects. 2.2.1, 2.2.2 and 2.2.3 to update values ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} on specific cells whose causality values are consistent with current sweeping direction (shown in Table 1), and update the causality arrays of their neighboring cells when it is needed.
4. Convergence: we check iteration convergence every four sweepings, i.e., if

$$||\phi_h^{(k)} - \phi_h^{(k-4)}||_{L^\infty} \leq \delta,$$

where δ is a given convergence threshold value, the iteration converges and stops. □

Remark The procedure of step 3 indicates that in each sweeping, candidate cells for which the DG local solver may be applied are only the cells whose causality indicator values are consistent with the current sweeping direction. This approach can save a lot of computational costs since the cells where the correct characteristic information has not reached are excluded from the current sweeping. In step 4 of the algorithm, $\delta = 10^{-11}$ is taken as the threshold value to stop the iterations in all numerical experiments of this paper.

Table 1 Specific cells to be updated in different direction sweepings

Sweeping direction	Causality arrays of cells to be updated
$i = 1 : N,\ j = 1 : M$	flagx(i,j) $\neq 1$ and flagy(i,j) $\neq 1$
$i = N : 1,\ j = 1 : M$	flagx(i,j) $= 1$ and flagy(i,j) $\neq 1$
$i = N : 1,\ j = M : 1$	flagx(i,j) $\neq 0$ and flagy(i,j) $= 1$
$i = 1 : N,\ j = M : 1$	flagx(i,j) $= 0$ and flagy(i,j) $= 1$

Table 2 Correspondence of colors and causality indicator values and information flow directions

Color	Flag x-direction	Flag y-direction
Red	0 $\rightarrow$	0 $\uparrow$
Blue	0 $\rightarrow$	1 $\downarrow$
Green	0 $\rightarrow$	10 no info
Pale Violet Red	1 $\leftarrow$	0 $\uparrow$
Dark Blue	1 $\leftarrow$	1 $\downarrow$
Aqua	1 $\leftarrow$	10 no info
Yellow	10 no info	0 $\uparrow$
Deep Pink	10 no info	1 $\downarrow$
Black	10 no info	10 no info

3 Numerical Examples

In this section, numerical examples are presented for solving the Eikonal equations (1.1). Examples include solutions which have discontinuities in their derivatives, and benchmark test cases such as the examples of shape from shading arising in the application area of computer vision. Numerical results demonstrate a uniform third order accuracy of the proposed method in smooth regions of the solutions, and the linear computational complexity. We calculate numerical errors for non-boundary cells in all examples.

Since in each sweeping, only if causality indicator values of the cells are consistent with the current sweeping direction, the third order DG local solver may be applied for them. Hence to measure the computational complexity accurately, we define the effective sweeping number as that in [24]:

$$\text{effective sweeping number} \triangleq \frac{\text{the total \# of times the third order DG local solver is executed}}{\text{the total \# of cells excluding the boundary cells}},$$

where "the third order DG local solver is executed" includes all cases that the subroutine for solving the quadratic system in the Subsect. 2.2.2 has been executed no matter whether the Newton iteration for local system converges or not. The resulting effective sweeping number can be a non-integer value since many cells are not updated in specific sweeps and hence are not counted towards the computation of this number. In all examples, if we have sweeping / effective sweeping number n, then it means n iterations, not $n \times 4$ iterations.

In all examples, we plot the patterns of causality indicator values when the iterations converge. Different colors represent different values of causality indicators. Their correspondence relationship is given in Table 2. Patterns of the second order scheme in [24] and the third order one developed in this paper are compared.

Example 1 $\Omega = [-1, 1]^2$, $\Gamma = \{(0, 0)\}$, and

$$f(x, y) = \frac{\pi}{2}\sqrt{\sin^2\left(\frac{\pi}{2}x\right) + \sin^2\left(\frac{\pi}{2}y\right)}, \qquad g(0, 0) = -2.$$

Table 3 Example 1: The values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} are preassigned on the cells in the fixed region $[-0.1, 0.1]^2$

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp.number	Swp. number
20 × 20	1.52E−5	–	1.29E−4	–	2.00	8
40 × 40	1.90E−6	3.00	1.57E−5	3.03	2.00	8
80 × 80	2.39E−7	2.99	1.95E−6	3.01	2.00	8
160 × 160	2.99E−8	3.00	2.43E−7	3.00	2.00	8
320 × 320	3.75E−9	3.00	3.03E−8	3.00	2.00	8
640 × 640	4.69E−10	3.00	3.79E−9	3.00	2.00	8
1280 × 1280	5.86E−11	3.00	4.74E−10	3.00	2.00	8

Table 4 Example 1: The values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} are preassigned on the cells in the region $[-h, h]^2$

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20 × 20	1.52E−5	–	1.29E−4	–	2.00	8
40 × 40	1.84E−6	3.04	1.61E−5	3.00	2.00	8
80 × 80	2.27E−7	3.02	2.02E−6	3.00	2.00	8
160 × 160	2.83E−8	3.01	2.52E−7	3.00	2.00	8
320 × 320	3.53E−9	3.00	3.15E−8	3.00	2.00	8
640 × 640	4.41E−10	3.00	3.94E−9	3.00	2.00	8
1,280 × 1,280	5.50E−11	3.00	4.93E−10	3.00	2.00	8

The exact solution is

$$\phi(x, y) = -\cos\left(\frac{\pi}{2}x\right) - \cos\left(\frac{\pi}{2}y\right).$$

To initialize the third order DG solver, we preassign the values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} on the boundary cells in the fixed region $[-0.1, 0.1]^2$ around Γ. The results are listed in Table 3. We can see that only 8 sweepings and 2 effective sweepings are needed for convergence regardless of the mesh size, and the error is uniformly third order both in L^1 and in L^∞ norms. The fact that iteration numbers are independent of mesh sizes verifies the linear computational complexity of the algorithm. If we preassign the values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} on the cells in the region $[-h, h]^2$ (h is grid size, so the number of boundary cells is 4 for all meshes) around Γ, we still observe uniformly third order accuracy as shown in Table 4. This shows that the third order DG fast sweeping method is robust with respect to different setups of the boundary cells in this example. In Fig. 1, the patterns of causality indicator values when the iterations converge are shown for both the second order and the third order method. We can see that they are totally same.

Example 2 (Point source distance problem). $\Omega = [-1, 1]^2$, $\Gamma = \{(0, 0)\}$ and $f(x, y) = 1$, $g = 0$. The boundary cells are the cells in the domain $[-0.1, 0.1]^2$. We preassign values for them based on the exact solution (the distance function from the point (0, 0)). The numerical results are listed in Table 5. We can again observe that only 8 sweepings and 2 effective sweepings are needed for convergence regardless of the mesh size and third order accuracy is obtained in both L^1 and L^∞ norms. As that in the example 1, the linear computational

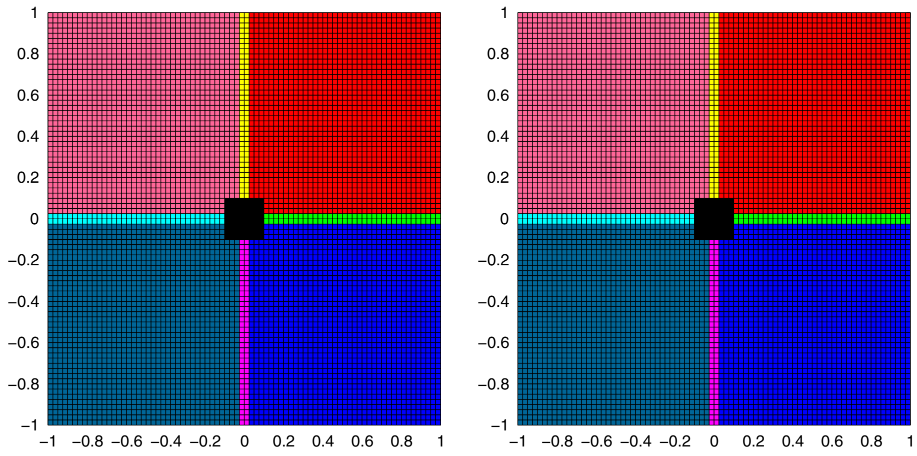

Fig. 1 Example 1. Visualization of final causality arrays on the 80 × 80 mesh. *Left*: the third order method; *right*: the second order method

Table 5 Example 2: The point source distance problem

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20 × 20	1.59E−3	–	3.52E−3	–	2.00	8
40 × 40	8.47E−5	4.23	7.96E−4	2.14	2.01	8
80 × 80	9.59E−6	3.14	9.16E−5	3.12	2.01	8
160 × 160	1.01E−6	3.25	1.21E−5	2.92	2.00	8
320 × 320	1.13E−7	3.16	1.56E−6	2.96	2.00	8
640 × 640	1.32E−8	3.10	1.98E−7	2.98	2.00	8
1,280 × 1,280	1.58E−9	3.05	2.49E−8	2.99	2.00	8

complexity of the algorithm is verified. In Fig. 2, the patterns of causality indicator values when the iterations converge are shown for both the second order and the third order method. We can see that they are same for most of cells, except for the cells at the middle lines of the domain where the information flows in just one direction.

Example 3 $\Omega = [-1, 1]^2$, Γ is a circle with center $(0, 0)$ and radius 0.5, and $f(x, y) = 1$, $g = 0$. To initialize the third order DG solver, we preassign the values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} on the boundary cells whose centers are within the $2h$ distance from Γ (h is grid size). The numerical results are listed in Tables 6 and 7. We observe as before that only 8 sweepings and 2 effective sweepings are needed for convergence regardless of the mesh size. Hence the algorithm has a linear computational complexity for this example. The error is uniformly third order in both L^1 and L^∞ norms if we measure it in the smooth region outside the circle center (see Table 6); or we have third order in L^1 and first order in L^∞ if the error is measured in the whole computational domain, as shown in Table 7. The patterns of causality indicator values when the iterations converge are shown in Fig. 3. As that in the example 2, we see similar patterns for the second order and the third order method, except at the middle lines of the domain where the information flows in just one direction.

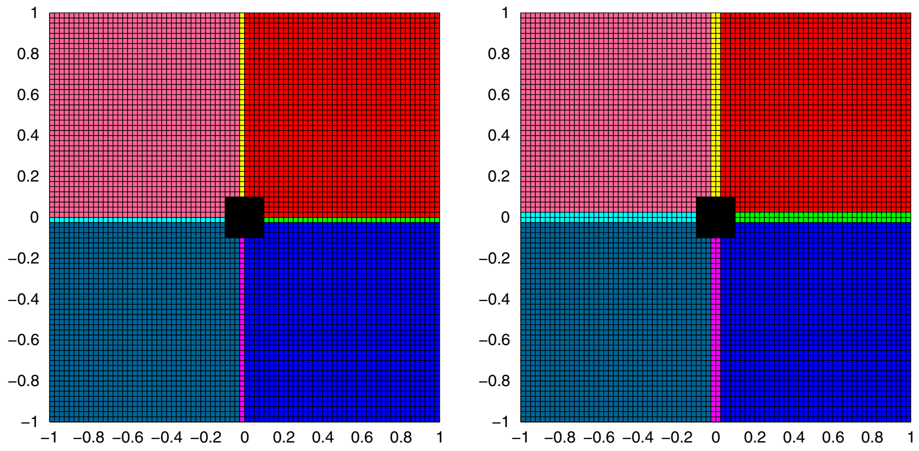

Fig. 2 Example 2. Visualization of final causality arrays on the 80 × 80 mesh. *Left*: the third order method; *right*: the second order method

Table 6 Example 3: The distance from one circle problem

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20 × 20	3.79E−5	–	2.99E−4	–	2.01	8
40 × 40	5.08E−6	2.90	5.28E−4	2.50	2.01	8
80 × 80	6.48E−7	2.97	7.49E−5	2.82	2.00	8
160 × 160	8.26E−8	2.97	1.03E−5	2.87	2.00	8
320 × 320	1.04E−8	2.99	1.35E−6	2.92	2.00	8
640 × 640	1.31E−9	3.00	1.75E−7	2.95	2.00	8
1,280 × 1,280	1.63E−10	3.00	2.23E−8	2.97	2.00	8

Errors are measured in the smooth region, which is outside of $[-0.1, 0.1]^2$

Table 7 Example 3: The distance from one circle problem

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20 × 20	5.58E−5	–	5.58E−3	–	2.01	8
40 × 40	8.18E−6	2.77	2.73E−3	1.03	2.01	8
80 × 80	1.16E−6	2.82	1.36E−3	1.00	2.00	8
160 × 160	1.61E−7	2.84	6.81E−4	1.00	2.00	8
320 × 320	2.22E−8	2.86	3.41E−4	1.00	2.00	8
640 × 640	3.03E−9	2.87	1.70E−4	1.00	2.00	8
1,280 × 1,280	4.10E−10	2.88	8.52E−5	1.00	2.00	8

Errors are measured in the whole region

Example 4 Consider Eikonal equation (1.1) with $f(x, y) = 1$, $g = 0$. The computational domain is $\Omega = [-1, 1] \times [-1, 1]$, and Γ consists of two circles of equal radius 0.3 with centers located at $(-0.5, -0.5)$ and $(0.5, 0.5)$, respectively. The exact solution is the distance function to Γ, i.e.

$$\phi(x, y) = \min(|\sqrt{(x-0.5)^2 + (y-0.5)^2} - 0.3|, \ |\sqrt{(x+0.5)^2 + (y+0.5)^2} - 0.3|).$$

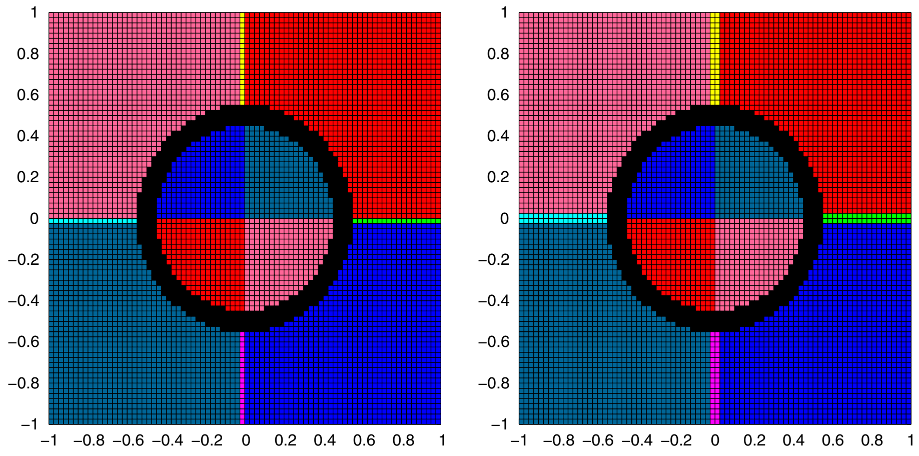

Fig. 3 Example 3. Visualization of final causality arrays on the 80 × 80 mesh. *Left*: the third order method; *right*: the second order method

Table 8 Example 4: Γ consists of two circles

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20 × 20	3.58E−5	–	2.49E−4	–	5.03	20
40 × 40	5.44E−6	2.72	3.79E−5	2.72	6.02	24
80 × 80	8.06E−7	2.75	5.79E−6	2.71	9.01	36
160 × 160	1.06E−7	2.92	8.06E−7	2.84	9.00	36
320 × 320	1.37E−8	2.95	1.07E−7	2.92	8.00	32
640 × 640	1.75E−9	2.97	1.37E−8	2.96	9.00	36
1,280 × 1,280	2.20E−10	2.99	1.74E−9	2.98	7.00	28

Errors are measured in the smooth region, which is outside of $[-0.6, -0.4]^2$, $[0.4, 0.6]^2$ and $|x + y| \leq 0.1$

The boundary cells are chosen to be cells whose centers are within the $2h$ distance from Γ (h is grid size). Characteristics intersect at the center of each circle and the line that is of equal distance to the two circles, and the solution develops singularities there. This test case is a challenge problem for high order fast sweeping methods, see for example [12,27], in terms of obtaining both high order accuracy in smooth regions and linear computational complexity. Our third order algorithm works well, as that shown in Tables 8 and 9. More iterations are needed in this example than previous ones due to the shock waves in the solution. Since high order schemes do not have monotonicity property as that in the first order scheme [28], the iteration numbers exhibit a little nonuniformity for different mesh sizes. As shown in Tables 8 and 9, about 5 to 9 effective sweepings are needed for different mesh sizes. In Table 10, the CPU times for computing causality indicators and arrays, the global CPU times, and their ratios are shown. We can see that the computations of causality information only take about 4–5 % of global CPU times. The global CPU times approximately exhibit a linear relationship with respect to different meshes. The error is uniformly third order in both L^1 and L^∞ norms if it is measured in smooth regions excluding the derivative singularities (Table 8). If the error is measured in the whole computational domain, we obtain second order in L^1 norm and first order in L^∞ norm (Table 9). We observe a little oscillation of the L^∞ accuracy order, again, due to the non-monotonicity of high order linear schemes around the shock wave regions.

Table 9 Example 4: Γ consists of two circles: whole region

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	1.06E−3	–	1.00E−1	–	5.03	20
40×40	2.03E−4	2.38	4.88E−2	1.04	6.02	24
80×80	4.39E−5	2.21	2.40E−2	1.02	9.01	36
160×160	1.04E−5	2.08	1.83E−2	0.39	9.00	36
320×320	2.48E−6	2.06	1.12E−2	0.71	8.00	32
640×640	6.06E−7	2.03	6.10E−3	0.88	9.00	36
$1{,}280 \times 1{,}280$	1.50E−7	2.02	2.89E−3	1.08	7.00	28

Table 10 Example 4: CPU time (Unit: seconds) and iteration numbers

Mesh	CPU for causal. info.	Global CPU	Percentage (%)	Swp. #	Eff. swp. #
80×80	0.07	1.56	4.49	36	9.01
160×160	0.29	6.11	4.75	36	9.00
320×320	1.12	22.29	5.02	32	8.00
640×640	5.13	97.30	5.27	36	9.00
$1{,}280 \times 1{,}280$	18.10	325.68	5.56	28	7.00

The pictures of the numerical solution on the 160×160 mesh are presented in Fig. 4. The derivative singularities are captured sharply. The patterns of causality indicator values when the iterations converge are shown in Fig. 5. For this example, similar patterns are observed on most of the cells for the second order and the third order method. The differences appear around the shock line that is of equal distance to the two circles, and the places where the information flows in just one direction.

To study the differences of weak solutions from the transformed equation (2.1) and the original Eq. (1.1), we implement the second order scheme based on the transformed equation, and compare the results with that in [24] based on the original equation. The implementation of the second order scheme based on the transformed equation follows same procedure as that in [24], except that it has different local causality constants and different local nonlinear system as that described in Sect. 2. Numerical errors, accuracy orders and iteration numbers are reported in Tables 11 and 12. Correct accuracy orders are obtained in both smooth regions and the whole region. CPU times for computing causality indicators and arrays, the global CPU times, and their ratios are shown in Table 13. We can see that the computations of causality information only take about 11–12 % of global CPU times. Both the global CPU times and CPU times for computing causality information approximately exhibit a linear relationship with respect to different meshes. The pictures of the numerical solution on the 160×160 mesh are shown in Fig. 6. We can see that the viscosity solution is captured very well. Comparing with the numerical errors reported in [24] based on the original equation (1.1), we can see very slight differences. We also compare the patterns of causality indicator values when the iterations converge. They are shown in Fig. 7 for 80×80 mesh, and Fig. 8 for 40×40 mesh. The patterns are exactly the same on the 40×40 mesh. On the 80×80 mesh, same patterns are also obtained except only four cells around the shock line. The comparisons here show that correct numerical viscosity solution is obtained for both formulations. Slight differences in numerical errors are observed. It could be due to that the transformed equation

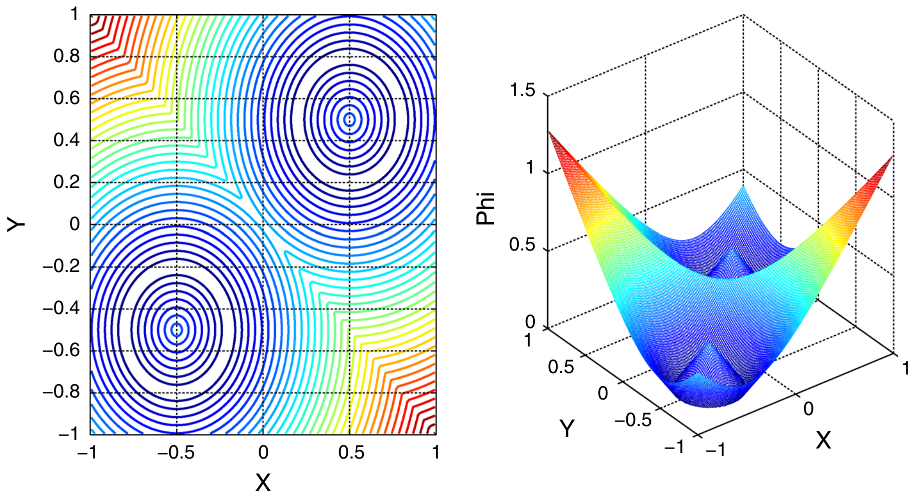

Fig. 4 Example 4. The pictures of the numerical solution of ϕ on the 160×160 mesh. *Left*: the contour plot for ϕ; *right*: the 3D plot for ϕ

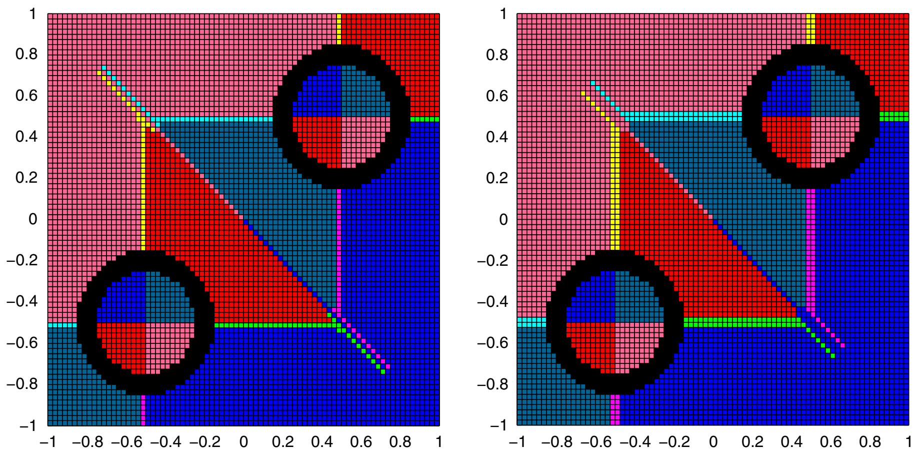

Fig. 5 Example 4. Visualization of final causality arrays on the 80×80 mesh. *Left*: the third order method; *right*: the second order method

has simpler formulas for local causality constants than the original equation. Hence the DG schemes have slight different fluxes.

Initial numerical values and causality information of the third order DG fast sweeping method can be provided by the second order DG fast sweeping method based on either the "transformed form" (2.1) or the "original form" (1.1). Here we test the third order method based on the second order one using the "transformed form". Correct numerical results are obtained as shown in Table 14. We also observe that the patterns of causality indicator values when the iterations converge are the same for different initial values.

We compare the computation efficiency of the third order fast sweeping DG method in this paper with the third order WENO fast sweeping method in [27], for this example. The numerical results of WENO fast sweeping method are reported in Tables 15 and 16. The

Table 11 Example 4: Γ consists of two circles

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	1.30E−3	–	1.73E−2	–	2.96	12
40×40	3.49E−4	1.89	2.54E−3	2.77	3.98	16
80×80	9.27E−5	1.91	7.64E−4	1.73	3.99	16
160×160	2.40E−5	1.95	2.14E−4	1.84	3.99	16
320×320	6.10E−6	1.98	5.74E−5	1.90	4.00	16
640×640	1.54E−6	1.99	1.48E−5	1.96	4.00	16
$1{,}280 \times 1{,}280$	3.87E−7	1.99	3.76E−6	1.98	4.00	16

Transformed equation is solved using second order method. Errors are measured in the smooth region, which is outside of $[-0.6, -0.4]^2$, $[0.4, 0.6]^2$ and $|x + y| \leq 0.1$

Table 12 Example 4: Γ consists of two circles

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	2.70E−3	–	7.65E−2	–	2.96	12
40×40	6.45E−4	2.07	3.88E−2	0.98	3.98	16
80×80	1.59E−4	2.02	1.95E−2	0.99	3.99	16
160×160	4.02E−5	1.99	9.79E−3	1.00	3.99	16
320×320	1.01E−5	1.99	4.90E−3	1.00	4.00	16
640×640	2.54E−6	1.99	2.45E−3	1.00	4.00	16
$1{,}280 \times 1{,}280$	6.38E−7	1.99	1.23E−3	1.00	4.00	16

Transformed equation is solved using second order method. Errors in whole region

Table 13 Example 4: CPU time (Unit: seconds) and iteration numbers when solving transformed equation by second order method

Mesh	CPU for causal. info.	Global CPU	Percentage (%)	Swp. #	Eff. swp. #
80×80	0.02	0.20	11.64	16	3.99
160×160	0.10	0.84	11.59	16	3.99
320×320	0.40	3.43	11.60	16	4.00
640×640	1.59	13.83	11.50	16	4.00
$1{,}280 \times 1{,}280$	6.57	55.54	11.83	16	4.00

computational complexity of the WENO fast sweeping method is not linear. However, each method has its advantages. We found that for this example, the third order fast sweeping DG method needs fewer CPU times to reach certain accuracy in the smooth region than the WENO one by comparing Tables 8, 10 and Table 15. Hence the DG one is more efficient than the third order WENO fast sweeping method to obtain accurate results for the smooth region of the solution. On the other hand, the third order WENO fast sweeping method is more efficient for the regions with derivative singularities to get certain accuracy, as shown in Tables 9, 10 and Table 16.

Example 5 (Shape-from-shading I) Consider Eikonal equation (1.1) with

$$f(x, y) = 2\sqrt{y^2(1 - x^2)^2 + x^2(1 - y^2)^2}. \tag{3.1}$$

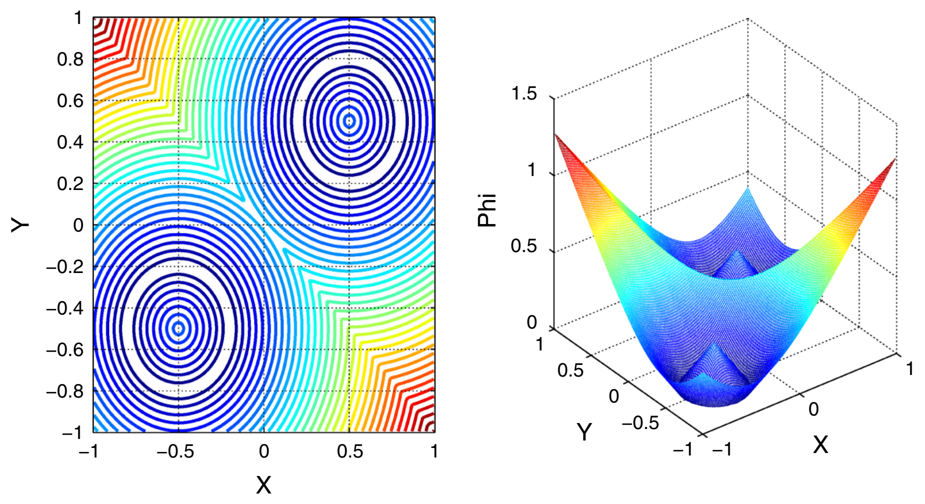

Fig. 6 Example 4. The pictures of the numerical solution of ϕ on the 160×160 mesh when solving transformed equation using second order method. *Left*: the contour plot for ϕ; *right*: the 3D plot for ϕ

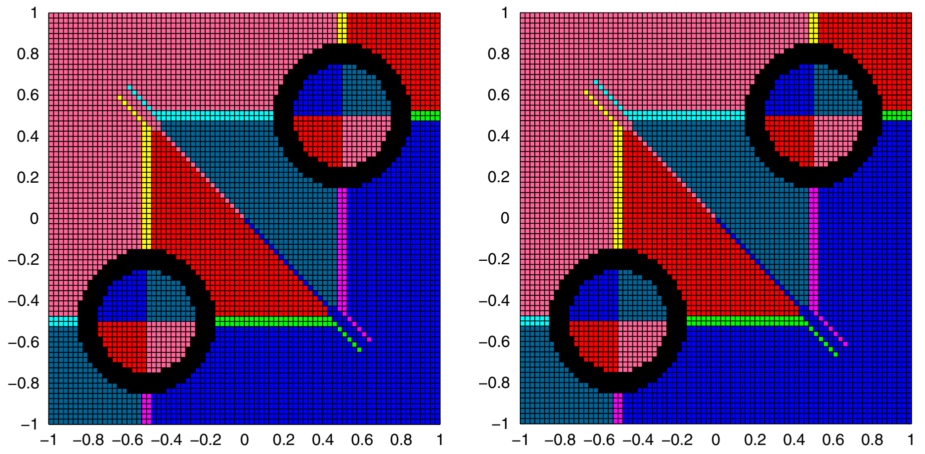

Fig. 7 Example 4. Visualization of final causality arrays on the 80 $\times$ 80 mesh using second order method. *Left*: solving transformed equation; *right*: solving original equation

The computational domain $\Omega = [-1, 1] \times [-1, 1]$. $\phi(x, y) = 0$ is prescribed at the boundary of the square with the additional boundary condition $\phi(0, 0) = 1$. This one and the next two examples are called shape-from-shading problems from the applications in computer vision [15]. They are typically used to test the high order numerical methods for Hamilton–Jacobi equations (e.g. [7,9,12,23,25,27]). The exact solution for this example is $\phi(x, y) = (1 - x^2)(1 - y^2)$. The boundary cells consist of the cells whose centers are within 0.1 distance from the boundary of the square, and the cells whose centers are in the domain $[-0.1, 0.1]^2$. We test the algorithm on both the uniform meshes and nonuniform meshes. The nonuniform meshes are obtained by randomly perturbing grid points of the uniform meshes in the range $[-0.1h, 0.1h] \times [-0.1h, 0.1h]$, where h is the mesh size of a uniform mesh. The numerical results are reported in Tables 17 and 18. We can observe that only

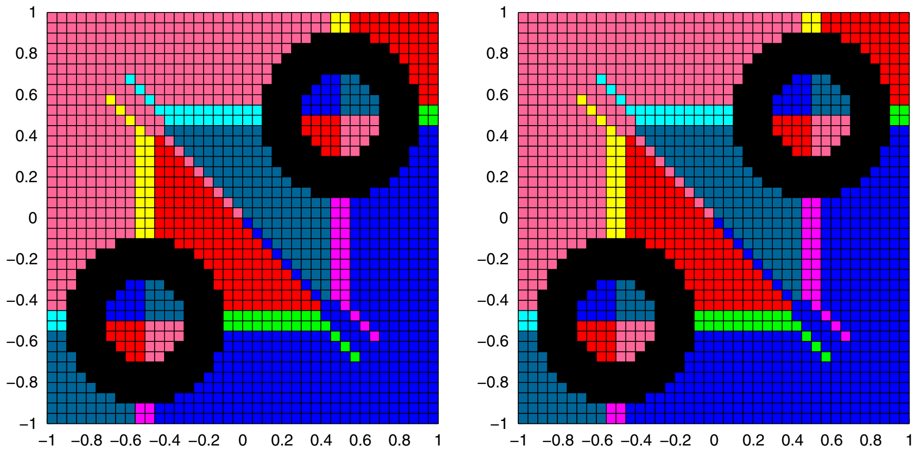

Fig. 8 Example 4. Visualization of final causality arrays on the 40×40 mesh using second order method. *Left*: solving transformed equation; *right*: solving original equation

Table 14 Example 4: Γ consists of two circles

mesh	L^1 error	order	L^∞ error	order	eff. swp.number	swp. number
20×20	3.58E−5	–	2.49E−4	–	5.03	20
40×40	5.44E−6	2.72	3.79E−5	2.72	6.02	24
80×80	8.06E−7	2.75	5.79E−6	2.71	9.00	36
160×160	1.06E−7	2.92	8.06E−7	2.84	10.00	40
320×320	1.37E−8	2.95	1.07E−7	2.92	8.00	32
640×640	1.75E−9	2.97	1.37E−8	2.96	8.00	32
$1{,}280 \times 1{,}280$	2.20E−10	2.99	1.74E−9	2.98	7.00	28

Transformed equation is solved by the second order method to provide initial information for the third order method. Errors are measured in the smooth region, which is outside of $[-0.6, -0.4]^2$, $[0.4, 0.6]^2$ and $|x+y| \leq 0.1$

Table 15 Example 4: Γ consists of two circles

Mesh	L^1 error	Order	L^∞ error	Order	Swp. number	Global CPU time (s)
80×80	9.30E−6	–	4.44E−4	–	116	0.2
160×160	5.26E−7	4.14	1.39E−5	4.99	176	1.22
320×320	1.09E−7	2.27	3.94E−6	1.82	176	4.93
640×640	1.46E−8	2.90	5.84E−7	2.76	280	31.15
$1{,}280 \times 1{,}280$	1.86E−9	2.97	7.65E−8	2.93	860	377.63

Third order WENO fast sweeping method. Errors are measured in the smooth region, which is outside of $[-0.6, -0.4]^2$, $[0.4, 0.6]^2$ and $|x + y| \leq 0.1$

2 and 3 effective sweepings are needed for convergence of the algorithm on uniform and nonuniform meshes respectively, regardless of mesh sizes. Uniform third order accuracy are obtained in both L^1 and L^∞ norms for uniform meshes. For nonuniform meshes, a clean

Table 16 Example 4: Γ consists of two circles

Mesh	L^1 error	Order	L^∞ error	Order	Swp. number	Global CPU time (s)
80×80	7.90E−5	–	5.87E−3	–	116	0.2
160×160	1.59E−5	2.31	2.80E−3	1.07	176	1.22
320×320	3.56E−6	2.16	1.27E−3	1.14	176	4.93
640×640	9.07E−7	1.97	5.34E−4	1.25	280	31.15
$1{,}280 \times 1{,}280$	1.90E−7	2.26	2.04E−4	1.39	860	377.63

Third order WENO fast sweeping method. Errors are measured in the whole region

Table 17 Example 5: Shape-from-shading problem I, uniform mesh

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	3.92E−5	–	5.25E−4	–	2.00	8
40×40	4.62E−6	3.08	6.83E−5	2.94	2.00	8
80×80	5.55E−7	3.06	8.69E−6	2.97	2.00	8
160×160	6.77E−8	3.03	1.10E−6	2.99	2.00	8
320×320	8.35E−9	3.02	1.37E−7	2.99	2.00	8
640×640	1.04E−9	3.01	1.72E−8	3.00	2.00	8
$1{,}280 \times 1{,}280$	1.29E−10	3.01	2.15E−9	3.00	2.00	8

Table 18 Example 5: Shape-from-shading problem I, non-uniform mesh

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	4.00E−5	–	5.63E−4	–	3.01	12
40×40	4.71E−6	3.08	7.35E−5	2.94	3.01	12
80×80	5.68E−7	3.05	9.36E−6	2.97	3.01	12
160×160	6.90E−8	3.04	2.72E−6	1.79	3.00	12
320×320	8.54E−9	3.02	1.58E−7	4.10	3.00	12
640×640	1.06E−9	3.01	6.37E−8	1.31	3.00	12
$1{,}280 \times 1{,}280$	1.31E−10	3.01	4.59E−9	3.79	3.00	12

The mesh is obtained by randomly perturbing grid points of the uniform mesh in the range $[-0.1h, 0.1h] \times [-0.1h, 0.1h]$

third order accuracy is obtained in L^1 norm, with some oscillations in the accuracy orders in L^∞ norm. The pictures of the numerical solution on the 160×160 mesh are presented in the Fig. 9. In Fig. 10, the patterns of causality indicator values when the iterations converge are shown for both the second order and the third order method. We can see that they are totally same.

Example 6 (Shape-from-shading II). Consider Eikonal equation (1.1) with

$$f(x, y) = 2\pi\sqrt{[\cos(2\pi x)\sin(2\pi y)]^2 + [\sin(2\pi x)\cos(2\pi y)]^2}. \tag{3.2}$$

The computational domain $\Omega = [0, 1]\times[0, 1]$. $\Gamma = \{(\frac{1}{4}, \frac{1}{4}), (\frac{3}{4}, \frac{3}{4}), (\frac{1}{4}, \frac{3}{4}), (\frac{3}{4}, \frac{1}{4}), (\frac{1}{2}, \frac{1}{2})\}\cup \partial\Omega$, consisting of five isolated points and the domain boundary. $g(\frac{1}{4}, \frac{1}{4}) = g(\frac{3}{4}, \frac{3}{4}) = 1$,

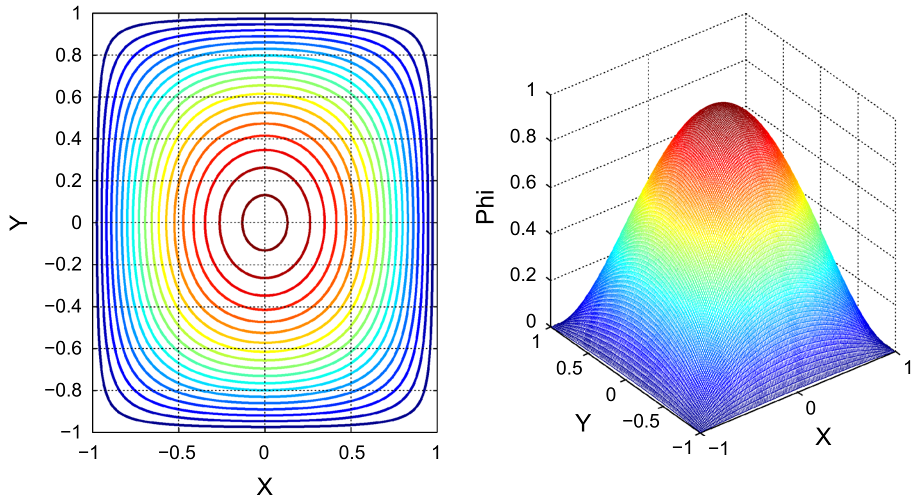

Fig. 9 Example 5. The pictures of the numerical solution of ϕ on the 160×160 mesh. *Left*: the contour plot for ϕ; *right*: the 3D plot for ϕ

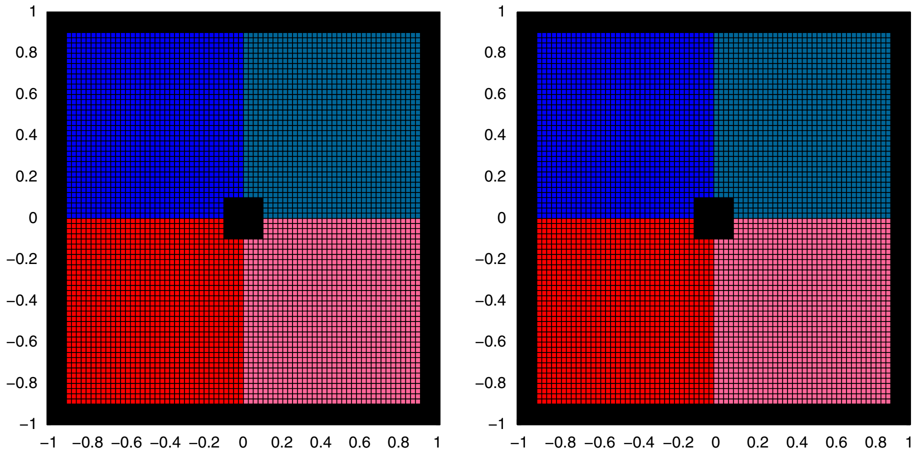

Fig. 10 Example 5. Visualization of final causality arrays on the 80×80 mesh. *Left*: the third order method; *right*: the second order method

$g(\frac{1}{4}, \frac{3}{4}) = g(\frac{3}{4}, \frac{1}{4}) = -1$, and $g(\frac{1}{2}, \frac{1}{2}) = 0$. In addition, $\phi(x, y) = 0$ is prescribed on $\partial\Omega$. The solution for this problem is the shape function, which has the brightness $I(x, y) = 1/\sqrt{1 + f(x, y)^2}$ under vertical lighting [15]. The exact solution is $\phi(x, y) = \sin(2\pi x)\sin(2\pi y)$. This is a challenge problem for high order fast sweeping methods to achieve both high order accuracy and fast convergence speed, as shown in [12,27].

The boundary cells are chosen to consist of cells whose centers are within 0.05 distance from the boundary of the unit square, and the cells which are in the five square boxes with length 0.1 and the centers $\{(\frac{1}{4}, \frac{1}{4}), (\frac{3}{4}, \frac{3}{4}), (\frac{1}{4}, \frac{3}{4}), (\frac{3}{4}, \frac{1}{4}), (\frac{1}{2}, \frac{1}{2})\}$. The third order DG fast sweeping method is tested on both the uniform meshes and non-uniform meshes. Again, the nonuniform meshes are obtained by randomly perturbing grid points of the uniform meshes

Table 19 Example 6: Shape-from-shading problem II, uniform mesh

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	4.10E−4	–	5.03E−3	–	3.01	12
40×40	3.92E−5	3.39	5.18E−4	3.28	3.00	12
80×80	4.32E−6	3.18	5.95E−5	3.12	3.00	12
160×160	5.21E−7	3.05	7.14E−6	3.06	4.00	16
320×320	6.43E−8	3.02	8.75E−7	3.03	4.00	16
640×640	7.98E−9	3.01	1.08E−7	3.01	4.00	16
$1{,}280 \times 1{,}280$	9.93E−10	3.01	1.35E−8	3.01	4.00	16

Table 20 Shape-from-shading problem II, Example 6, non-uniform mesh

Mesh	L^1 error	Order	L^∞ error	Order	Eff. swp. number	Swp. number
20×20	6.35E−4	–	1.26E−1	–	4.00	16
40×40	4.02E−5	3.98	6.29E−4	7.65	4.00	16
80×80	4.44E−6	3.18	7.27E−5	3.11	4.00	16
160×160	5.44E−7	3.03	1.25E−5	2.54	4.00	16
320×320	6.66E−8	3.03	1.12E−6	3.49	4.00	16
640×640	8.27E−9	3.01	1.56E−7	2.84	4.00	16
$1{,}280 \times 1{,}280$	1.03E−9	3.00	2.36E−8	2.73	4.00	16

The mesh is obtained by randomly perturbing grid points of the uniform mesh in the range $[-0.1h, 0.1h] \times [-0.1h, 0.1h]$

in the range $[-0.1h, 0.1h] \times [-0.1h, 0.1h]$ where h is the mesh size of a uniform mesh. The numerical results are reported in Tables 19 and 20. We can observe that only 3 or 4 effective sweepings are needed for the convergence for uniform meshes. For nonuniform meshes, the effective sweeping number is 4 regardless of the mesh size. We observe a uniform third order accuracy for both the L^1 and the L^∞ norms, with a slight oscillation in L^∞ accuracy orders for nonuniform meshes as that in Example 5.

In Table 21, the CPU times for computing causality indicators and arrays, the global CPU times, and their ratios are reported. We can see that the computations of causality information only take about 1–1.5 % of global CPU times. CPU times show linear computational complexity of the algorithm. The pictures of the numerical solution on the 160×160 mesh are shown in Fig. 11. The patterns of causality indicator values when the iterations converge are shown in Fig. 12. For this example, similar patterns are observed on most of the cells for the second order and the third order method. The differences appear around the places where the information flows in just one direction.

Example 7 (Shape-from-shading III). Consider Eikonal equation (1.1) with

$$f(x, y) = \sqrt{(1 - |x|)^2 + (1 - |y|)^2}. \tag{3.3}$$

The computational domain $\Omega = [-1, 1] \times [-1, 1]$. $\phi(x, y) = 0$ is prescribed at the boundary of the square. This example is used to test the inverse Lax-Wendroff procedure for the boundary cells. The exact solution of this problem is $\phi = (1 - |x|)(1 - |y|)$. Here we do not use the exact solution to provide boundary values. The values of ϕ_{ij}, u_{ij}, v_{ij}, a_{ij}, b_{ij} and c_{ij} on the boundary cells whose centers are within the distance h from Γ are generated by the inverse Lax-Wendroff procedure [8]. The numerical results

Table 21 Example 6, uniform mesh

Mesh	CPU for causal. info.	Global CPU	Percentage (%)	Swp. #	Eff. swp. #
40 × 40	0.017	1.29	1.32	12	3.00
80 × 80	0.06	4.88	1.23	12	3.00
160 × 160	0.28	21.91	1.28	16	4.00
320 × 320	1.11	80.26	1.38	16	4.00
640 × 640	4.25	303.36	1.40	16	4.00
1,280 × 1,280	12.53	1202.26	1.04	16	4.00

CPU time (Unit: seconds) and iteration numbers

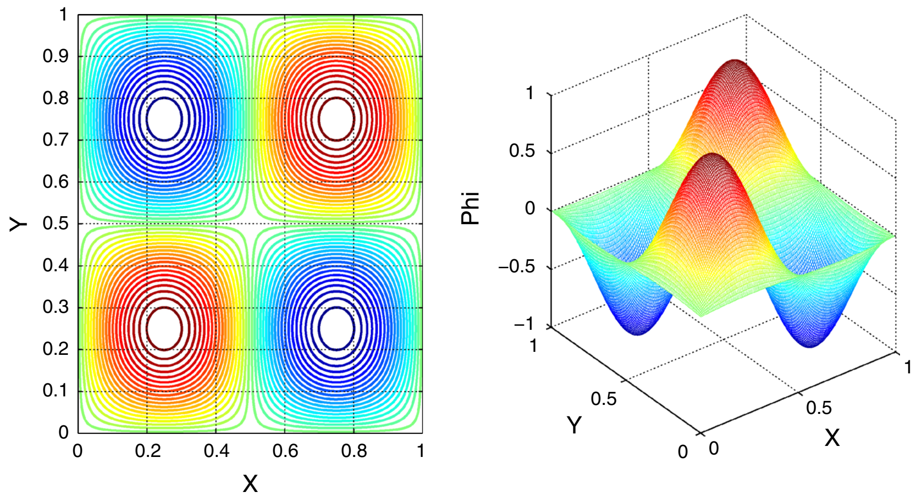

Fig. 11 Example 6. The pictures of the numerical solution of ϕ on the 160 × 160 mesh. *Left*: the contour plot for ϕ; *right*: the 3D plot for ϕ

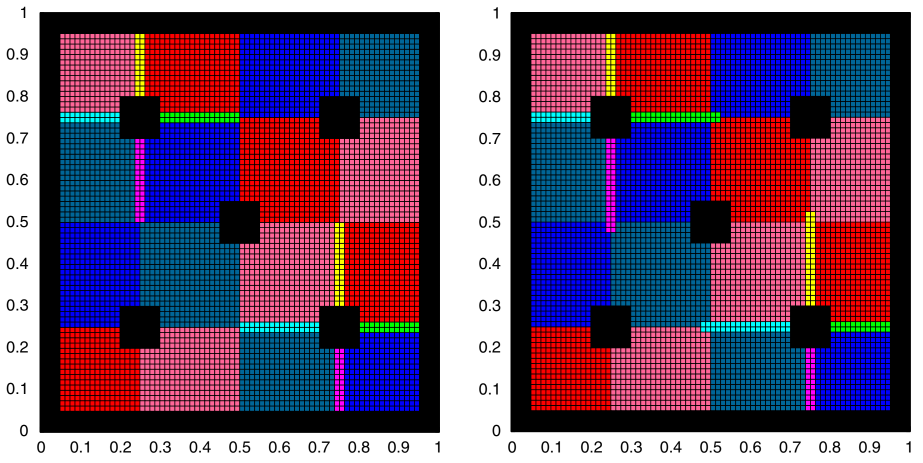

Fig. 12 Example 6. Visualization of final causality arrays on the 80 × 80 mesh. *Left*: the third order method; *right*: the second order method

Table 22 Example 7: Shape-from-shading problem III

Mesh	L^1 error	Order	L^∞ error	Order	Swp. #	Eff. swp. #
20×20	2.25E−16	–	1.79E−15	–	8	2.00
40×40	2.35E−16	–	2.23E−15	–	8	2.00
80×80	2.43E−16	–	2.17E−15	–	8	2.00
160×160	2.45E−16	–	3.52E−15	–	8	2.00
320×320	2.75E−16	–	4.81E−15	–	8	2.00
640×640	3.41E−16	–	6.91E−15	–	8	2.00
$1{,}280 \times 1{,}280$	3.90E−16	–	8.52E−15	–	8	2.00

A test for the inverse Lax-Wendroff boundary treatment

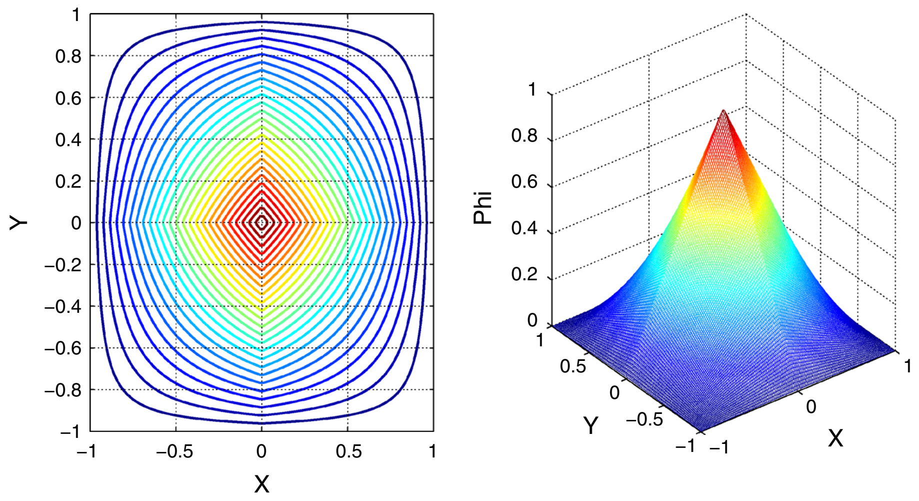

Fig. 13 Example 7. The pictures of the numerical solution of ϕ on the 160×160 mesh. *Left*: the contour plot for ϕ; *right*: the 3D plot for ϕ

are reported in Table 22. We can see that only 8 sweepings and 2 effective sweepings are needed for convergence regardless of the mesh size. Since the exact solution of this problem is a piecewise quadratic polynomial in the finite element space V_h^2 (2.2), the numerical errors are at the round-off error level as shown in Table 22. The pictures of the numerical solution on the 160×160 mesh are presented in Fig. 13. In Fig. 14, the patterns of causality indicator values when the iterations converge are shown for both the second order and the third order method. We can see that they are totally same.

Remark In all numerical examples of this section, we present patterns of causality indicator values when the iterations converge and compare them for the second order and the third order DG fast sweeping method. It is interesting that similar patterns are observed on most of the cells for different order methods. The slight differences often appear around the shock lines and the places where the information propagates in just one direction.

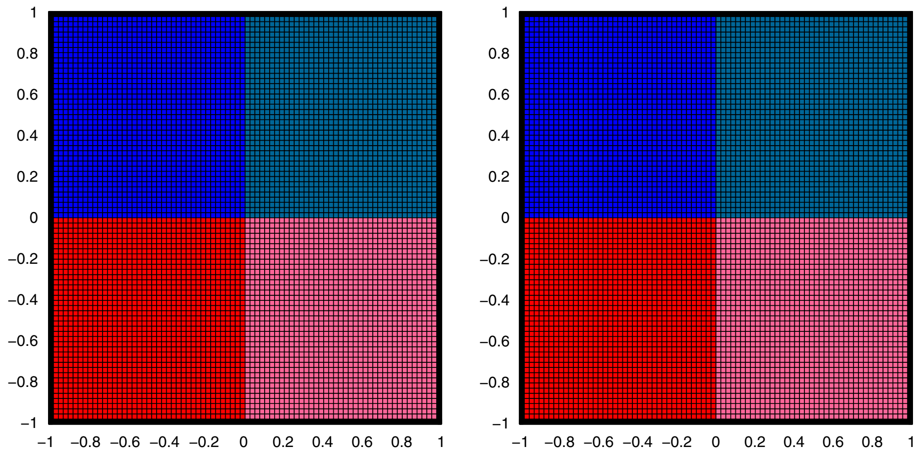

Fig. 14 Example 7. Visualization of final causality arrays on the 80 × 80 mesh. *Left*: the third order method; *right*: the second order method

4 Conclusions

In this paper, we extend previous work in [24] and design a third order fast sweeping algorithm with linear computational complexity for solving two dimensional Eikonal equations. The difficulties of solving a more complicated local nonlinear system and calculations of causality information in the third order DG scheme are resolved. The algorithm is tested by various numerical examples. We observe both a uniform third order accuracy in smooth regions of the solution and the fast convergence speed (i.e., linear computational complexity) in the numerical examples. In this paper, we focus on the algorithm development and tests. The theoretical analysis of the algorithm has not been carried out and is still an open problem. This important aspect will be one of our next studies and provide guidance in applications of the algorithm. Extension of the algorithm to solve higher dimensional problems follows similar procedures, and its implementation and numerical experiments will also be carried out in our next research.

Acknowledgments We thank helpful discussions with Chi-Wang Shu and Hongkai Zhao on this project

Appendix: Detailed Formulae of the 6 × 6 Jacobian Matrix in Solving the Local Nonlinear System

$$
\begin{aligned}
J(1,1) &= \frac{\partial f_1}{\partial \phi} = l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\\
J(1,2) &= \frac{\partial f_1}{\partial u} = 8k\, u_{ij} + l_j(-\alpha_l - \alpha_r)\\
J(1,3) &= \frac{\partial f_1}{\partial v} = \frac{8}{k}\, v_{ij} + h_i(-\alpha_b - \alpha_t)\\
J(1,4) &= \frac{\partial f_1}{\partial a} = \frac{32k}{3}\, a_{ij} + l_j(\alpha_l - \alpha_r) + \frac{1}{3}h_i(\alpha_b - \alpha_t)
\end{aligned}
$$

$$J(1,5) = \frac{\partial f_1}{\partial b} = \frac{32}{3k}\, b_{ij} + \frac{1}{3} l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)$$
$$J(1,6) = \frac{\partial f_1}{\partial c} = (\frac{8k}{3} + \frac{8}{3k})\, c_{ij}$$

$$J(2,1) = \frac{\partial f_2}{\partial \phi} = l_j(-\alpha_l - \alpha_r)$$
$$J(2,2) = \frac{\partial f_2}{\partial u} = \frac{16k}{3}\, a_{ij} + l_j(\alpha_l - \alpha_r) + \frac{1}{3} h_i(\alpha_b - \alpha_t)$$
$$J(2,3) = \frac{\partial f_2}{\partial v} = \frac{8}{3k}\, c_{ij}$$
$$J(2,4) = \frac{\partial f_2}{\partial a} = \frac{16k}{3}\, u_{ij} + l_j(-\alpha_l - \alpha_r)$$
$$J(2,5) = \frac{\partial f_2}{\partial b} = \frac{1}{3} l_j(-\alpha_l - \alpha_r)$$
$$J(2,6) = \frac{\partial f_2}{\partial c} = \frac{8}{3k}\, v_{ij} + \frac{1}{3} h_i(-\alpha_b - \alpha_t)$$

$$J(3,1) = \frac{\partial f_3}{\partial \phi} = h_i(-\alpha_b - \alpha_t)$$
$$J(3,2) = \frac{\partial f_3}{\partial u} = \frac{8k}{3}\, c_{ij}$$
$$J(3,3) = \frac{\partial f_3}{\partial v} = \frac{16}{3k}\, b_{ij} + \frac{1}{3} l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)$$
$$J(3,4) = \frac{\partial f_3}{\partial a} = \frac{1}{3} h_i(-\alpha_b - \alpha_t)$$
$$J(3,5) = \frac{\partial f_3}{\partial b} = \frac{16}{3k} v_{ij} + h_i(-\alpha_b - \alpha_t)$$
$$J(3,6) = \frac{\partial f_3}{\partial c} = \frac{8k}{3}\, u_{ij} + \frac{1}{3} l_j(-\alpha_l - \alpha_r)$$

$$J(4,1) = \frac{\partial f_4}{\partial \phi} = l_j(\alpha_l - \alpha_r) + \frac{1}{3} h_i(\alpha_b - \alpha_t)$$
$$J(4,2) = \frac{\partial f_4}{\partial u} = \frac{8k}{3}\, u_{ij} + l_j(-\alpha_l - \alpha_r)$$
$$J(4,3) = \frac{\partial f_4}{\partial v} = \frac{8}{3k}\, v_{ij} + \frac{1}{3} h_i(-\alpha_b - \alpha_t)$$
$$J(4,4) = \frac{\partial f_4}{\partial a} = \frac{32k}{5}\, a_{ij} + l_j(\alpha_l - \alpha_r) + \frac{1}{5} h_i(\alpha_b - \alpha_t)$$
$$J(4,5) = \frac{\partial f_4}{\partial b} = \frac{32}{9k}\, b_{ij} + \frac{1}{3}\big(l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\big)$$
$$J(4,6) = \frac{\partial f_4}{\partial c} = (\frac{8k}{9} + \frac{8}{5k})\, c_{ij}$$

$$\begin{aligned}
J(5,1) &= \frac{\partial f_5}{\partial \phi} = \frac{1}{3} l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t) \\
J(5,2) &= \frac{\partial f_5}{\partial u} = \frac{8k}{3}\, u_{ij} + \frac{1}{3} l_j(-\alpha_l - \alpha_r) \\
J(5,3) &= \frac{\partial f_5}{\partial v} = \frac{8}{3k}\, v_{ij} + h_i(-\alpha_b - \alpha_t) \\
J(5,4) &= \frac{\partial f_5}{\partial a} = \frac{32k}{9}\, a_{ij} + \frac{1}{3}\big(l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\big) \\
J(5,5) &= \frac{\partial f_5}{\partial b} = \frac{32}{5k}\, b_{ij} + \frac{1}{5} l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t) \\
J(5,6) &= \frac{\partial f_5}{\partial c} = (\frac{8k}{5} + \frac{8}{9k})\, c_{ij}
\end{aligned}$$

$$\begin{aligned}
J(6,1) &= \frac{\partial f_6}{\partial \phi} = 0 \\
J(6,2) &= \frac{\partial f_6}{\partial u} = \frac{1}{3} h_i(-\alpha_b - \alpha_t) \\
J(6,3) &= \frac{\partial f_6}{\partial v} = \frac{1}{3} l_j(-\alpha_l - \alpha_r) \\
J(6,4) &= \frac{\partial f_6}{\partial a} = \frac{16k}{9}\, c_{ij} \\
J(6,5) &= \frac{\partial f_6}{\partial b} = \frac{16}{9k}\, c_{ij} \\
J(6,6) &= \frac{\partial f_6}{\partial c} = \frac{16k}{9}\, a_{ij} + \frac{16}{9k}\, b_{ij} + \frac{1}{3}\big(l_j(\alpha_l - \alpha_r) + h_i(\alpha_b - \alpha_t)\big)
\end{aligned}$$

References

1. Boué, M., Dupuis, P.: Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control. SIAM J. Numer. Anal. **36**, 667–695 (1999)
2. Cheng, Y., Shu, C.-W.: A discontinuous Galerkin finite element method for directly solving the Hamilton–Jacobi equations. J. Comput. Phys. **223**, 398–415 (2007)
3. Crandall, M.G., Lions, P.L.: Viscosity solutions of Hamilton–Jacobi equations. Trans. Am. Math. Soc. **277**, 1–42 (1983)
4. Dijkstra, E.W.: A note on two problems in connection with graphs. Numer. Math. **1**, 269–271 (1959)
5. Fomel, S., Luo, S., Zhao, H.: Fast sweeping method for the factored eikonal equation. J. Comput. Phys. **228**, 6440–6455 (2009)
6. Helmsen, J., Puckett, E., Colella, P., Dorr, M.: Two new methods for simulating photolithography development in 3D. Proc. SPIE **2726**, 253–261 (1996)
7. Hu, C., Shu, C.-W.: A discontinuous Galerkin finite element method for Hamilton–Jacobi equations. SIAM J. Sci. Comput. **20**, 666–690 (1999)
8. Huang, L., Shu, C.-W., Zhang, M.: Numerical boundary conditions for the fast sweeping high order WENO methods for solving the Eikonal equation. J. Comput. Math. **26**, 336–346 (2008)
9. Jiang, G.-S., Peng, D.: Weighted ENO schemes for Hamilton–Jacobi equations. SIAM J. Sci. Comput. **21**, 2126–2143 (2000)
10. Kao, C.Y., Osher, S., Qian, J.: Lax–Friedrichs sweeping schemes for static Hamilton–Jacobi equations. J. Comput. Phys. **196**, 367–391 (2004)
11. Kao, C.Y., Osher, S., Qian, J.: Legendre-transform-based fast sweeping methods for static Hamilton–Jacobi equations on triangulated meshes. J. Comput. Phys. **227**, 10209–10225 (2008)
12. Li, F., Shu, C.-W., Zhang, Y.-T., Zhao, H.-K.: A second order discontinuous Galerkin fast sweeping method for Eikonal equations. J. Comput. Phys. **227**, 8191–8208 (2008)

13. Qian, J., Zhang, Y.-T., Zhao, H.-K.: Fast sweeping methods for Eikonal equations on triangular meshes. SIAM J. Numer. Anal. **45**, 83–107 (2007)
14. Qian, J., Zhang, Y.-T., Zhao, H.-K.: A fast sweeping method for static convex Hamilton–Jacobi equations. J. Sci. Comput. **31**, 237–271 (2007)
15. Rouy, E., Tourin, A.: A viscosity solutions approach to shape-from-shading. SIAM J. Numer. Anal. **29**, 867–884 (1992)
16. Serna, S., Qian, J.: A stopping criterion for higher-order sweeping schemes for static Hamilton–Jacobi equations. J. Comput. Math. **28**, 552–568 (2010)
17. Sethian, J.A.: A fast marching level set method for monotonically advancing fronts. Proc. Natl. Acad. Sci. USA **93**, 1591–1595 (1996)
18. Sethian, J.A., Vladimirsky, A.: Ordered upwind methods for static Hamilton–Jacobi equations. Proc. Natl. Acad. Sci. USA **98**, 11069–11074 (2001)
19. Sethian, J.A., Vladimirsky, A.: Ordered upwind methods for static Hamilton–Jacobi equations: theory and algorithms. SIAM J. Numer. Anal. **41**, 325–363 (2003)
20. Tan, S., Shu, C.-W.: Inverse Lax-Wendroff procedure for numerical boundary conditions of conservation laws. J. Comput. Phys. **229**, 8144–8166 (2010)
21. Tsai, Y.-H., Cheng, L.-T., Osher, S., Zhao, H.-K.: Fast sweeping algorithms for a class of Hamilton–Jacobi equations. SIAM J. Numer. Anal. **41**, 673–694 (2003)
22. Tsitsiklis, J.N.: Efficient algorithms for globally optimal trajectories. IEEE Trans. Autom. Control **40**, 1528–1538 (1995)
23. Xiong, T., Zhang, M., Zhang, Y.-T., Shu, C.-W.: Fifth order fast sweeping WENO scheme for static Hamilton–Jacobi equations with accurate boundary treatment. J. Sci. Comput. **45**, 514–536 (2010)
24. Zhang, Y.-T., Chen, S., Li, F., Zhao, H., Shu, C.-W.: Uniformly accurate discontinuous Galerkin fast sweeping methods for Eikonal equations. SIAM J. Sci. Comput. **33**, 1873–1896 (2011)
25. Zhang, Y.-T., Shu, C.-W.: High order WENO schemes for Hamilton–Jacobi equations on triangular meshes. SIAM J. Sci. Comput. **24**, 1005–1030 (2003)
26. Zhang, Y.-T., Zhao, H.-K., Chen, S.: Fixed-point iterative sweeping methods for static Hamilton–Jacobi equations. Methods Appl. Anal. **13**, 299–320 (2006)
27. Zhang, Y.-T., Zhao, H.-K., Qian, J.: High order fast sweeping methods for static Hamilton–Jacobi equations. J. Sci. Comput. **29**, 25–56 (2006)
28. Zhao, H.-K.: A fast sweeping method for Eikonal equations. Math. Comput. **74**, 603–627 (2005)
29. Zhao, H., Osher, S., Merriman, B., Kang, M.: Implicit and non-parametric shape reconstruction from unorganized points using variational level set method. Comput. Vis. Image Underst. **80**, 295–319 (2000)