Computers and Mathematics with Applications 156 (2024) 74–86

Contents lists available at ScienceDirect

Computers and Mathematics with Applications

journal homepage: www.elsevier.com/locate/camwa

Laplacian regularized eikonal equation with Soner boundary condition on polyhedral meshes ☆

Jooyoung Hahn *, Karol Mikula, Peter Frolkovič

Department of Mathematics and Descriptive Geometry, Slovak University of Technology, Radlinskeho 11, 810 05 Bratislava, Slovakia

ARTICLE INFO

Keywords:
Vanishing viscosity method
Eikonal equation
Soner boundary condition
Laplacian regularizer
Cell-centered finite volume method
Polyhedral meshes

ABSTRACT

In this paper, we propose a numerical algorithm based on a cell-centered finite volume method to compute a distance from given objects on a three-dimensional computational domain discretized by polyhedral cells. Inspired by the vanishing viscosity method, a Laplacian regularized eikonal equation is solved and the Soner boundary condition is applied to the boundary of the domain to avoid a non-viscosity solution. As the regularization parameter depending on a characteristic length of the discretized domain is reduced, a corresponding numerical solution is calculated. A convergence to the viscosity solution is verified numerically as the characteristic length becomes smaller and the regularization parameter accordingly becomes smaller. From the numerical experiments, the second experimental order of convergence in the L^1 norm error is confirmed for smooth solutions. Compared to solve a time-dependent form of the eikonal equation, the Laplacian regularized eikonal equation has the advantage of reducing computational cost dramatically when a more significant number of cells is used or a region of interest is far away from the given objects. The implementation of parallel computing using domain decomposition with 1-ring face neighborhood structure can be done straightforwardly by a standard cell-centered finite volume code.

1. Introduction

The viscosity solution of an eikonal equation is used in various applications from pure geometrical analysis to complicated problems mentioned in [1,2]. In the premixed turbulent combustion with thin flame fronts [3], a distance from the thin flame modeled by a surface is used to design the flame-wall interaction and quenching [4] or the end-gas autoignition for knock prediction [5]. A distance from a computational boundary, so-called wall distance, is a crucial feature in turbulence modeling methods [6–10]. It is also useful to obtain the medial axis transformation [11,12] of a given domain, which is crucial to automated mesh generation [13,14]. In cardiac electrophysiology [15–17], a properly modeled eikonal equation approximates a propagation of excitation wavefront by the time to excite all points in the myocardium. In geophysics, a propagation of seismic waves is described by an eikonal equation in the high frequency regions [18].

In order to make the more realistic simulation of the mentioned applications, it is necessary to use three-dimensional (3D) discretized domain in a fine scale to capture detailed phenomena. On such a domain, parallel computing using domain decomposition is inevitable because of the significantly high consumption of memory. Moreover, computational domains of the industrial problems described by a complex boundary shape are commonly discretized by polyhedral cells; see more advantages to using polyhedral cells [19]. Therefore, the target we would like to achieve here is to compute a distance function from given objects on polyhedral meshes by parallel computing using domain decomposition with the simplest structure of overlapping domains, that is, 1-ring face neighbor structure [20]. For usability of the developed algorithm, it should be possible to make a straightforward implementation in a standard code of cell-centered finite volume method (FVM).

The most well-known algorithm to efficiently solve an eikonal equation is usually considered to be the fast marching method (FMM) [21–23]. The fast computation is obtained by keeping a heap data structure to handle active nodes on a propagating front as candidates for updating the values. However, for typical parallel computing using domain decomposition, the heap structure is difficult to be maintained

☆ The work was supported by grants, VEGA 1/0436/20, VEGA 1/0314/23, and APVV-19-0460. This project No. 2140/01/01 has received funding from the European Unionś Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 945478.

* Corresponding author.
E-mail addresses: jooyoung.hahn@stuba.sk (J. Hahn), karol.mikula@stuba.sk (K. Mikula), peter.frolkovic@stuba.sk (P. Frolkovič).

https://doi.org/10.1016/j.camwa.2023.12.016

efficiently in parallel computation. An alternative approach is the fast sweeping method (FSM) [24–27] by updating necessary values with a Gauss-Seidel type iterations and it achieves better computational speed in simple computational domain because sorting is not used; see detailed computational study of FMM and FSM in [28,29]. In the fast iterative method (FIM) [30–32], a fine-grained parallel algorithm to solve an eikonal equation is presented on regular square, triangular, and tetrahedron meshes. A particular assumption to use FIM and FMM on triangular or tetrahedral meshes is that the shape of the cell is restricted to an acute triangle or tetrahedron. For obtuse shapes, a smart division is necessary to make all cells as acute shapes but it is not clear how efficiently it can be divided in polyhedral meshes in a situation of moving mesh or remeshing that commonly happens in combustion simulation. The mentioned limitation is resolved in [33] based on the jet marching method (JMM) [34] to solve the eikonal equation, using Hermite interpolation and a compact high-order semi-Lagrangian method.

In this paper, we numerically find a viscosity solution of an eikonal equation:

$$\begin{aligned} |\nabla u(\mathbf{x})| &= 1, \quad \mathbf{x}\in\Omega\setminus\Gamma, \\ u(\mathbf{x}) &= 0, \quad \mathbf{x}\in\Gamma, \end{aligned} \tag{1}$$

where a computational domain $\Omega\subset\mathbb{R}^3$ is either convex or non-convex and Γ indicates fixed locations represented by a collection of curves or surfaces or a part of the boundary of the computational domain. The viscosity solution of (1) defined in [35] is the Euclidean distance function from Γ on the domain Ω. A noticeable necessary condition of being the viscosity solution of (1) is an inequality condition on the boundary of the domain:

$$\boldsymbol{\nu}(\mathbf{x})\cdot\nabla u(\mathbf{x})\geq 0, \quad \mathbf{x}\in\partial\Omega\setminus\Gamma, \tag{2}$$

where $\boldsymbol{\nu}$ is the outward normal to the boundary of the domain. The above inequality is presented in the *Remark* of interpreting Proposition II.1 in [36]. It is so-called the Soner boundary condition [37] or the state constraint condition in optimal control problems [36,38]. The condition is applied on obstacle boundaries [39] and it restricts the discrete set of admissible control on all points in a domain in order to avoid an incorrect search direction. A general shape of obstacle embedded in a discretized domain is considered in [29]. The eikonal equation (1) and the Soner boundary condition (2) are discretized by a monotone finite difference scheme in [37] when a set Γ is a collection of finite discrete points and the error bound of the scheme is derived to the order of the square of cell size on a regular rectangular mesh. The obstacle [39] can be understood as a hole in a domain [40]. The necessity of using the Soner boundary condition and its geometrical interpretation is explained in [40] by numerical examples.

A time-relaxed formulation of (1) with the Soner boundary condition (2) is presented to compute a signed distance function when a shape of Γ is a closed, bounded, orientable, and connected surface Γ in a general computational domain $\Omega\subset\mathbb{R}^3$ [40]:

$$\begin{aligned} \frac{\partial}{\partial t}\phi(\mathbf{x},t)\pm|\nabla\phi(\mathbf{x},t)| &= \pm 1 \quad (\mathbf{x},t)\in\Omega^{\pm}\times(0,T], \\ \phi(\mathbf{x},t) &= 0 \quad (\mathbf{x},t)\in\Gamma\times[0,T], \\ \boldsymbol{\nu}(\mathbf{x})\cdot\nabla\phi(\mathbf{x},t) &\geq 0 \quad (\mathbf{x},t)\in(\partial\Omega\setminus\Gamma)\times(0,T], \end{aligned} \tag{3}$$

where $\phi(\mathbf{x},0)>0$ on Ω^+ and $\phi(\mathbf{x},0)<0$ on Ω^- are outside and inside the closed surface, respectively. The Soner boundary condition is essential to avoid a non-viscosity solution, especially on a non-convex domain. The distance information from Γ is propagated into the rest of the domain $\Omega\setminus\Gamma$ along the normal direction to Γ over time. The steady state solution eventually becomes a signed distance function from Γ. In the case of computing a wall distance function, that is, $\Gamma=\partial\Omega$, a transport form of eikonal equation (1) is presented in [11] and the algorithm is implemented by a standard FVM code with the first order upwind scheme. Even if the time relaxation in [11,40] with a proper choice of time step brings robustness of the algorithm, a main disadvantage of using (3) is a large amount of computational cost when a region of interest is located far away from Γ.

Inspired by the vanishing viscosity method [36], the equation we would like to solve numerically in this paper is combined with a Laplacian regularizer:

$$\begin{aligned} -\epsilon\,\triangle u_\epsilon(\mathbf{x})+|\nabla u_\epsilon(\mathbf{x})| &= 1 \quad \mathbf{x}\in\Omega\setminus\Gamma, \\ u_\epsilon(\mathbf{x}) &= 0 \quad \mathbf{x}\in\Gamma, \end{aligned} \tag{4}$$

where $\epsilon>0$ is the regularization parameter. When $\Gamma=\partial\Omega$, the vanishing viscosity method proves that the solution u_ϵ converges to the viscosity solution of (1) as $\epsilon\to 0$. In the case of $\Gamma\subsetneq\partial\Omega$, a boundary condition on $\partial\Omega$ has to be imposed in order to solve (4) numerically. A Dirichlet boundary condition on $\partial\Omega$ can only be applied to extremely simple problems, *e.g.*, where Γ is a sphere and Ω is a box, so it is not a realistic boundary condition for general shapes of Γ and Ω. When a Neumann boundary condition is used, it is also unrealistic because the level surfaces of the solution arrive to the boundary with any angles, which cannot be known in advance; see also [39]. A linearly extended boundary condition in [20] is used with the bidirectional flow, however, it is obviously not accurate enough except the case of a linear solution. As shown in [40], the Soner boundary condition plays a crucial role in keeping a numerical solution not to being a non-viscosity solution. Using the Soner boundary condition of u_ϵ, $\epsilon>0$, in (4),

$$\boldsymbol{\nu}(\mathbf{x})\cdot\nabla u_\epsilon(\mathbf{x})\geq 0 \quad \mathbf{x}\in\partial\Omega\setminus\Gamma, \tag{5}$$

we can avoid a non-viscosity solution when $\epsilon\to 0$. The inequality makes intuitive sense because the distance function from a given Γ is an increasing function and must be increasing on the boundary. When (4) is solved by a standard finite volume method with an iterative numerical solver, since there is no guarantee to keep the Soner boundary condition, it is critical to impose (5) in order to eventually find an approximation of a viscosity solution (1).

Compared to solve (3), a clear advantage of solving the equations (4) and (5) is to improve computational cost because of an infinite propagation speed caused by the Laplacian regularization term. In order to numerically solve (4) and (5), two difficulties should be resolved: the first is how to deal with the nonlinear term and the second is how to choose a regularization parameter. In [41,42], the same Laplacian regularizer is used for computing a wall distance function, that is, $\Gamma=\partial\Omega$. The non-linearity in (4) is resolved by using $|\nabla u_\epsilon|^2$ and its linearization. The choice of the regularization parameter depends on an approximated distance from Γ, which makes more inaccurate results on the far field. In [43–46], the non-linearity in (1) is managed by an energy minimization with the constraint $\mathbf{p}=\nabla u$ and then a penalty method or augmented Lagrangian method is used to approximate a viscosity solution of (1) for the cases of $\Gamma=\partial\Omega$. Throughout this paper, we discuss the details of two mentioned difficulties of solving (4) and (5) in order to obtain a meaningful convergence order numerically.

The rest of the paper is presented as follows. In Section 2, we explain the proposed method to compute a solution of the governing equation (4) and (5) on a polyhedron mesh. In Section 3, numerical properties of the proposed algorithm are presented by examples with exact solutions. Finally, we conclude in Section 4

2. Proposed method

We start with explaining concrete notations to bring a clear understanding of polyhedral cells. In the following subsections, a linearized eikonal equation with Laplacian regularizer is introduced and its discretization based on a cell-centered FVM is presented in detail. Finally, we explain how to design a decreasing sequence of regularization parameters and propose an algorithm to approximate a viscosity solution of (1) by solving (4) and (5) in the last subsection.

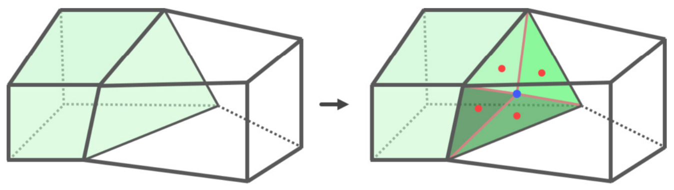


Fig. 1. An illustration of two polyhedral cells with a tessellated face.

2.1. Notations

Let us denote a discretized computational domain as a union of non-overlapped polyhedral cells with a non-zero volume:

$$\bar{\Omega} = \bigcup_{p\in\mathcal{I}} \bar{\Omega}_p, \tag{6}$$

where Ω_p is open and $\mathcal{I}$ is a set of the indices of cells; see an illustration of two polyhedral cells in Fig. 1. If a face is in-between two adjacent cells, we call it an internal face. Otherwise, we call it a boundary face. A set $\mathcal{G}$ is the collection of indices of all internal faces. Since the face of a polyhedron cell is difficult to be a plane in a general shape of the computational domain, we always consider a tessellation of a face into triangles unless the face is already a triangle. From a face e_g, $g \in \mathcal{G}$, whose vertices are $\mathbf{x}_{v_i}$, $i = 1, \dots, r_g$, a triangle $\mathcal{T}_i = \mathcal{T}(\mathbf{x}_{v_i}, \mathbf{x}_{v_{i+1}}, \mathbf{x}_0)$ of three points, $\mathbf{x}_{v_i}$, $\mathbf{x}_{v_{i+1}}$, and the center of the mass $\mathbf{x}_0 = \frac{1}{r_g}\sum_{i=1}^{r_g} \mathbf{x}_{v_i}$ is used to define a center of the face:

$$\mathbf{x}_g = \frac{\sum_{i=1}^{r_g} |\mathcal{T}_i|\, \bar{\mathbf{x}}_i}{\sum_{i=1}^{r_g} |\mathcal{T}_i|}, \tag{7}$$

where $\mathbf{x}_{v_{r_g+1}} = \mathbf{x}_{v_1}$ and $\bar{\mathbf{x}}_i$ and $|\mathcal{T}_i|$ are the center and area of the triangle $\mathcal{T}_i$, respectively. Note that $\mathbf{x}_g$ is not necessarily the same as $\mathbf{x}_0$ the center of the mass in general. In order to indicate the tessellated faces of a general face indexed by $\mathcal{G}$, we define a set of the indices of a tessellated internal and boundary faces as $\mathcal{F}$ and $\mathcal{B}$. For example, e_f, $f \in \mathcal{F}$, is a triangle on a face between left and right cells in Fig. 1 and $\mathbf{x}_f$ (red point) is the center of the triangle, where all triangles share a vertex, the center of the face $\mathbf{x}_g$ (blue point). To sum up, for a face e_g, $g \in \mathcal{G}$, there exists a subset $\mathcal{F}_g \subset \mathcal{F}$ such that

$$e_g = \bigcup_{f\in\mathcal{F}_g} e_f.$$

If a face e_g is not a triangle, it is a collection of tessellated faces (triangles) e_f, $f \in \mathcal{F}_g$, whose common vertex is $\mathbf{x}_g$. If e_g is a triangle, then there is an index $f \in \mathcal{F}$ such that $e_g = e_f$.

For a cell Ω_p, $p \in \mathcal{I}$, we define a set $\mathcal{N}_p$ as the indices of neighbor cells Ω_q such that the intersection $\partial\Omega_p \cap \partial\Omega_q = e_g$, $g \in \mathcal{G}$, is a face of non-zero area between two adjacent cells. We also define $\mathcal{F}_p$ and $\mathcal{B}_p$ as internal and boundary triangles tessellated by faces of Ω_p. When $\mathcal{B}_p$ is empty, we call the cell Ω_p as an internal cell. Otherwise, it is called a boundary cell. For example, if a green cell Ω_p in Fig. 1 is a boundary cell whose only left side is a part of the boundary of the computational domain, $|\mathcal{N}_p| = 5$, $|\mathcal{F}_p| = 20$, and $|\mathcal{B}_p| = 4$. If the cell next to the green cell is Ω_q, then $q \in \mathcal{N}_p$ and there is an index $g \in \mathcal{G}$ such that $e_g = \partial\Omega_p \cap \partial\Omega_q$. In the rest of the paper, we use the subscripts f, b, and g to indicate an internal triangle e_f, a boundary triangle e_b, and an internal face e_g, respectively, unless otherwise noted.

For an internal triangle e_f, $f \in \mathcal{F}_p$, $p \in \mathcal{I}$, the vector $\mathbf{n}_{pf}$ is the outward normal to the triangle and its length is the area of the triangle, $|\mathbf{n}_{pf}| = |e_f|$. Then, $e_f \subset \partial\Omega_q$ for $q \in \mathcal{N}_p$, $\mathbf{n}_{qf} = -\mathbf{n}_{pf}$ holds. For a boundary triangle e_b, $b \in \mathcal{B}_p$, $p \in \mathcal{I}$, the vector $\mathbf{n}_b = \mathbf{n}_{pb}$ is the outward normal to the triangle, that is, the outward normal to the boundary of the computational domain, and its length is the area of the triangle, $|\mathbf{n}_b| = |e_b|$. When a directional vector is specified by two position vectors $\mathbf{x}_a$ and $\mathbf{x}_b$, we use a notation $\mathbf{d}_{ab} = \mathbf{x}_b - \mathbf{x}_a$. For an internal face $e_g = \partial\Omega_p \cap \partial\Omega_q$, $g \in \mathcal{G}$, p, $q \in \mathcal{I}$, whose vertices are written by $\mathbf{x}_{v_i}$, $i = 1, \dots, r_g$, we define a vector:

$$\mathbf{n}_g = \frac{1}{2}\sum_{i=2}^{r_g-1} \mathbf{d}_{v_1 v_i} \times \mathbf{d}_{v_1 v_{i+1}}, \tag{8}$$

where the order of vertices is decided such that the cross product $\mathbf{d}_{v_1 v_i} \times \mathbf{d}_{v_1 v_{i+1}}$ indicates the outward to the cell Ω_p for all $i = 2, \dots, r_g - 1$. If the face e_g is planar, the vector $\mathbf{n}_g$ becomes an outward normal vector to the face of the cell Ω_p and its length $|\mathbf{n}_g| = |e_g|$ is the area of the face.

The characteristic length of a discretized domain $\cup_{p\in\mathcal{I}_\mathrm{L}} \Omega_p$ is defined by the average of one-third power to the volume of the bounding box of a cell:

$$h_\mathrm{L} = \frac{1}{|\mathcal{I}_\mathrm{L}|} \sum_{p\in\mathcal{I}_\mathrm{L}} |\Omega_p|_B^{\frac{1}{3}}, \tag{9}$$

where $|\Omega_p|_B$ is the volume of the box whose diagonal is a vector $\mathbf{x}_M - \mathbf{x}_m$, where $\mathbf{x}_m$ and $\mathbf{x}_M$ are component-wise minimum and maximum of all vertices $\mathbf{x}_{v_i}$ of Ω_p, respectively. The L indicates the level of mesh refinement, that is, when L increases, finer cells are generated. In Section 3, we use four levels of cells, roughly $h_{\mathrm{L}+1} \approx \frac{1}{2} h_\mathrm{L}$, to check the experimental order of convergence (EOC).

2.2. Linearized eikonal equation with Laplacian regularization

In this subsection, we assume that there is a known function $u_{\epsilon'}$ which is possibly close to the solution of (4) and (5) with a regularization parameter $\epsilon' > 0$. We present how to use a cell-centered finite volume method with the Soner boundary condition to numerically find a solution u_ϵ of (4) and (5) with a smaller regularization parameter $\epsilon < \epsilon'$. Firstly, a linearization of the nonlinear term in (4) is used to obtain an equation of unknown function u_ϵ:

$$-\epsilon \bigtriangleup u_\epsilon(\mathbf{x}) + \mathbf{v}(\mathbf{x}) \cdot \nabla u_\epsilon(\mathbf{x}) = 1, \quad \mathbf{v}(\mathbf{x}) = \frac{\nabla u_{\epsilon'}(\mathbf{x})}{|\nabla u_{\epsilon'}(\mathbf{x})|_\sigma}, \quad \mathbf{x} \in \Omega \setminus \Gamma, \tag{10}$$

where $|\mathbf{x}|_\sigma = (|\mathbf{x}|^2 + \sigma^2)^{\frac{1}{2}}$ with a small constant $\sigma = 10^{-12}$. Note that $\mathbf{v}$ is a fixed vector and the details of computing $u_{\epsilon'}$ are explained in the next subsection. Secondly, we show how to apply the Soner boundary condition in a cell-centered finite volume method. Even if a discretization of the normal flow term, $\mathbf{v} \cdot \nabla u_\epsilon$, with Soner boundary condition is already presented in [40], we repeat the key points of the numerical scheme in order to completely explain a discretization of the Laplacian term with Soner boundary condition based on the flux-balanced approximation [47] on a polyhedral cell.

Before we derive a discretization of using the Soner boundary condition, a gradient computation is necessary at the center $\mathbf{x}_p$ of the cell Ω_p. Since Γ can be a part of the boundary of the computational domain, let us denote an index set to indicate triangles on the boundary and Γ:

$$\mathcal{B}_D = \{ b \in \mathcal{B} : e_b \subset \Gamma \cap \partial\Omega \}. \tag{11}$$

Defining $\mathcal{A}_p = \mathcal{N}_p \cup \left(\mathcal{B}_p \cap \mathcal{B}_D\right)$, the weighted least-squares method is used to compute the gradient at the center $\mathbf{x}_p$:

$$\nabla u_p \equiv \nabla u(\mathbf{x}_p) = \underset{\substack{\mathbf{y}\in\mathbb{R}^3\\ |\mathbf{y}|\le 1}}{\arg\min} \left(\sum_{a\in\mathcal{A}_p} \frac{(u_p + \mathbf{y} \cdot \mathbf{d}_{pa} - u_a)^2}{|\mathbf{d}_{pa}|^2} \right). \tag{12}$$

Note that $u_a = u(\mathbf{x}_a) = 0$, $a \in \mathcal{B}_p \cap \mathcal{B}_D$, because of Dirichlet boundary condition in (4). The constraint in (12) is also used in [40] which brings a more stable numerical computation. A component-wise constraint of the gradient is presented in [41,11] to improve stability. In [48], the same constraint in (12) is shown for a minimization approach to solve the eikonal equation.

Now, we use the basic idea of flux-balanced approximation [47] and a deferred correction method on the linearized equation (10). By the relation $\nabla u \cdot \mathbf{v} = \nabla \cdot (u\mathbf{v}) - u\nabla \cdot \mathbf{v}$, the equation is evaluated at the center of the cell Ω_p:

$$-\epsilon \nabla \cdot \nabla u(\mathbf{x}_p) + \nabla \cdot (u\mathbf{v})(\mathbf{x}_p) - u(\mathbf{x}_p)\nabla \cdot \mathbf{v}(\mathbf{x}_p) = 1,$$

where $u = u_\epsilon$ for simplicity of formula derivation. Approximating a divergence of vector-valued function $\mathbf{F}$ evaluated at $\mathbf{x}_p$ by integrating over the cell Ω_p:

$$\nabla \cdot \mathbf{F}(\mathbf{x}_p) \approx \frac{1}{|\Omega_p|} \int_{\Omega_p} \nabla \cdot \mathbf{F} dV = \frac{1}{|\Omega_p|} \int_{\partial\Omega_p} \mathbf{F} \cdot \mathbf{n} dS,$$

where $\mathbf{n}$ is an unit outward normal vector to $\partial\Omega_p$, then we have

$$0 = \underbrace{-\epsilon \int_{\partial\Omega_p} \nabla u \cdot \mathbf{n} dS}_{\texttt{(II)}} + \underbrace{\int_{\partial\Omega_p} u\mathbf{v} \cdot \mathbf{n} dS - u_p \int_{\partial\Omega_p} \mathbf{v} \cdot \mathbf{n} dS}_{\texttt{(I)}} - |\Omega_p| \tag{13}$$

After the complete discretization of two terms (I) and (II) is derived, we present a deferred correction method to compute the solution of (10) at the end of this subsection.

The term (I) in (13) is further calculated:

$$\begin{aligned}\texttt{(I)} &= \sum_{f \in \mathcal{F}_p \cup \mathcal{B}_p} \left(\int_{e_f} u\mathbf{v} \cdot \frac{\mathbf{n}_{pf}}{|\mathbf{n}_{pf}|} dS - u_p \int_{e_f} \mathbf{v} \cdot \frac{\mathbf{n}_{pf}}{|\mathbf{n}_{pf}|} dS \right) \\ &\approx \sum_{f \in \mathcal{F}_p \cup \mathcal{B}_p} \left(u_{pf} - u_p\right) \mu_{pf},\end{aligned} \tag{14}$$

where u_{pf} is a value at the center of face e_f, $f \in \mathcal{F}_p$, $u_p = u(\mathbf{x}_p)$, and the normal flux μ_{pf} is computed by

$$\mu_{pf} = \int_{e_f} \mathbf{v} \cdot \frac{\mathbf{n}_{pf}}{|\mathbf{n}_{pf}|} dS \approx \mathbf{v}_f \cdot \mathbf{n}_{pf}. \tag{15}$$

The last term above is obtained by a formula with a small constant $\sigma = 10^{-12}$:

$$\mu_{pf} \approx \frac{\boldsymbol{\beta}_f}{\left(|\boldsymbol{\beta}_f|^2 + \sigma^2\right)^{\frac{1}{2}}} \cdot \mathbf{n}_{pf}, \tag{16}$$

where $\boldsymbol{\beta}_f$ is a gradient whose length is less than or equal to 1 at the center of the triangle e_f, $f \in \mathcal{F}_p \cup \mathcal{B}_p$, computed by a constraint minimization using the pre-computed known function $u_{\epsilon'}$; see the equation (33) and the *Remark 1* in [40] for the technical details. In order to find the complete discretization of the first term, we define the sets of indices depending on the sign of the normal flux:

$$\begin{aligned}&\mathcal{B}_p^- = \{b \in \mathcal{B}_p : \mu_{pb} < 0\}, \quad \mathcal{B}_p^+ = \mathcal{B}_p \setminus \mathcal{B}_p^-, \\ &\mathcal{F}_p^- = \{f \in \mathcal{F}_p : \mu_{pf} < 0\}, \quad \mathcal{F}_p^+ = \mathcal{F}_p \setminus \mathcal{F}_p^-.\end{aligned} \tag{17}$$

Considering a general case of Γ in (4), for example, a part of $\partial\Omega$, we split the index set of boundary triangles into three cases:

$$\mathcal{B}_p = \left(\mathcal{B}_p^- \cap \mathcal{B}_D\right) \cup \left(\mathcal{B}_p^- \setminus \mathcal{B}_D\right) \cup \mathcal{B}_p^+, \tag{18}$$

where $\mathcal{B}_D$ is defined by (11). On a boundary triangle e_b, $b \in \mathcal{B}_p$, we derive the numerical scheme on $\mathcal{B}_p^+$ because it does not violate Soner boundary condition and on $\mathcal{B}_p^- \cap \mathcal{B}_D$ because Dirichlet boundary condition should be explicitly applied. The terms occurring on $\mathcal{B}_p^- \setminus \mathcal{B}_D$ should be set to zero so as not to violate the Soner boundary condition. Then, the original discretization of the normal flow in [49,50] is changed because of using the Soner boundary condition:

$$\begin{aligned}\texttt{(I)} \approx &\sum_{f \in \mathcal{F}_p^-} \left(u_q + \mathcal{D}_q u \cdot \mathbf{d}_{qf} - u_p\right) \mu_{pf} + \sum_{f \in \mathcal{B}_p^+ \cup \mathcal{F}_p^+} \left(\mathcal{D}_p u \cdot \mathbf{d}_{pf}\right) \mu_{pf} \\ &+ \sum_{b \in \mathcal{B}_p^- \cap \mathcal{B}_D} \left(u_b - u_p\right) \mu_{pb}\end{aligned} \tag{19}$$

where $u_b = u(\mathbf{x}_b) = 0$, $b \in \mathcal{B}_p^- \cap \mathcal{B}_D$ and the modified inflow-based gradient $\mathcal{D}_p u$ is used to include the influence of the Soner boundary condition:

$$\mathcal{D}_p u = \frac{\displaystyle\sum_{f \in \mathcal{F}_p^- \cup (\mathcal{B}_p^- \cap \mathcal{B}_D)} \frac{1}{|\mathbf{d}_{pf}|} \boldsymbol{\beta}_f}{\displaystyle\sum_{f \in \mathcal{F}_p^- \cup (\mathcal{B}_p^- \cap \mathcal{B}_D)} \frac{1}{|\mathbf{d}_{pf}|}}. \tag{20}$$

The term (II) is followed by the discretization of flux-balanced approximation [47]:

$$\texttt{(II)} = \sum_{q \in \mathcal{N}_p} \int_{e_g} \nabla u \cdot \frac{\mathbf{n}_g}{|\mathbf{n}_g|} dS + \sum_{b \in \mathcal{B}_p} \int_{e_b} \nabla u \cdot \frac{\mathbf{n}_{pb}}{|\mathbf{n}_{pb}|} dS, \tag{21}$$

where a polygon face $e_g = \partial\Omega_p \cap \partial\Omega_q$, $g \in \mathcal{G}$, $q \in \mathcal{N}_p$, $p \in \mathcal{I}$. From the centers of two cells, $\mathbf{x}_p$ and $\mathbf{x}_q$, we find two points $\mathbf{x}_{p'}$ and $\mathbf{x}_{q'}$ such that the directional vectors $\mathbf{d}_{pp'}$ and $\mathbf{d}_{qq'}$ are perpendicular to the line passing at $\mathbf{x}_g$ (7) along the direction $\mathbf{n}_g$ (8):

$$\mathbf{d}_{pp'} = \mathbf{d}_{pg} - \left(\frac{\mathbf{n}_g}{|\mathbf{n}_g|} \cdot \mathbf{d}_{pg}\right) \frac{\mathbf{n}_g}{|\mathbf{n}_g|}, \quad \mathbf{d}_{qq'} = \mathbf{d}_{qg} - \left(\frac{\mathbf{n}_g}{|\mathbf{n}_g|} \cdot \mathbf{d}_{qg}\right) \frac{\mathbf{n}_g}{|\mathbf{n}_g|}.$$

Using the explicit expression $\mathbf{d}_{pp'}$ and $\mathbf{d}_{qq'}$, we have an approximation of the first integral in (21):

$$\begin{aligned}\sum_{q \in \mathcal{N}_p} \int_{e_g} \nabla u \cdot \frac{\mathbf{n}_g}{|\mathbf{n}_g|} dS &\approx \sum_{q \in \mathcal{N}_p} \frac{|e_g|}{|\mathbf{d}_{p'q'}|} \left(u_{q'} - u_{p'}\right) \\ &\approx \sum_{q \in \mathcal{N}_p} \frac{|e_g|}{|\mathbf{d}_{p'q'}|} \left(\left(u_q + \nabla u_q \cdot \mathbf{d}_{qq'}\right) - \left(u_p + \nabla u_p \cdot \mathbf{d}_{pp'}\right)\right)\end{aligned} \tag{22}$$

Note that more technical details are described in [47]. The second integral in (21) should be considered more carefully to apply the Soner boundary condition. Similar to (18), we split the index set of $\mathcal{B}_p$ into three cases.

$$\mathcal{B}_p = \left(\mathcal{B}_p \cap \mathcal{B}_D\right) \cup \left(\mathcal{B}_p \setminus \mathcal{B}_D\right) = \left(\mathcal{B}_p \cap \mathcal{B}_D\right) \cup \left(\mathcal{B}_p^+ \setminus \mathcal{B}_D\right) \cup \left(\mathcal{B}_p^- \setminus \mathcal{B}_D\right). \tag{23}$$

In the first case, on a triangle e_b, $b \in \mathcal{B}_p \cap \mathcal{B}_D$, the Dirichlet condition is applied. In the second case, we use numerical values inside the computational domain. In the third case, the terms occurring on $\mathcal{B}_p^- \setminus \mathcal{B}_D$ should be set to zero not to violate the Soner boundary condition. Considering the mentioned three cases, we have an approximation of the second integral (21):

$$\begin{aligned}\sum_{b \in \mathcal{B}_p} \int_{e_b} \nabla u \cdot \frac{\mathbf{n}_{pb}}{|\mathbf{n}_{pb}|} dS \approx &\sum_{b \in \mathcal{B}_p \cap \mathcal{B}_D} \frac{|e_b|}{|\mathbf{d}_{p'b}|} \left(u_b - u_p - \nabla u_p \cdot \mathbf{d}_{pp'}\right) \\ &+ \sum_{b \in \mathcal{B}_p^+ \setminus \mathcal{B}_D} \nabla u_p \cdot \mathbf{n}_b,\end{aligned} \tag{24}$$

where $u_b = u(\mathbf{x}_b) = 0$, $b \in \mathcal{B}_p \cap \mathcal{B}_D$.

Combining all derivations (19), (22), and (24), we have a complete discretization using the Soner boundary condition to solve (10):

$$\begin{aligned}0 = &-\epsilon \left(\sum_{q \in \mathcal{N}_p} \frac{|e_g|}{|\mathbf{d}_{p'q'}|} \left(u_q + \nabla u_q \cdot \mathbf{d}_{qq'} - u_p - \nabla u_p \cdot \mathbf{d}_{pp'}\right) \right) \\ &- \epsilon \left(\sum_{b \in \mathcal{B}_p \cap \mathcal{B}_D} \frac{|e_b|}{|\mathbf{d}_{p'b}|} \left(u_b - u_p - \nabla u_p \cdot \mathbf{d}_{pp'}\right) + \sum_{b \in \mathcal{B}_p^+ \setminus \mathcal{B}_D} \nabla u_p \cdot \mathbf{n}_b \right) \\ &+ \sum_{f \in \mathcal{F}_p^-} \left(u_q + \mathcal{D}_q u \cdot \mathbf{d}_{qf} - u_p\right) \mu_{pf} + \sum_{f \in \mathcal{B}_p^+ \cup \mathcal{F}_p^+} \left(\mathcal{D}_p u \cdot \mathbf{d}_{pf}\right) \mu_{pf} \\ &+ \sum_{b \in \mathcal{B}_p^- \cap \mathcal{B}_D} \left(u_b - u_p\right) \mu_{pb} - |\Omega_p|,\end{aligned} \tag{25}$$

where the gradient ∇u_p and the modified inflow-based gradient $\mathcal{D}_p u$ are defined by (12) and (20), respectively. On a regular cubic mesh, the displacement $\mathbf{d}_{pp'}$ and $\mathbf{d}_{qq'}$ are zero vectors and the equation above is a banded block diagonal matrix equation. In parallel computing using domain decomposition with the 1-ring neighborhood structure, if Ω_p is located in the domain D_1 and one of faces of Ω_p is located between two domains, D_1 and D_2, one of the second face neighbor cells on Ω_p, that is, Ω_r, $r \in \mathcal{N}_q \setminus \mathcal{N}_p$ and $q \in \mathcal{N}_p$, may not be accessible by the domain D_1 where Ω_p is located. Such a cell exists when we compute ∇u_q or $\mathcal{D}_q u$ in the formulation of (25) and then it is not possible to construct a correct linear system in the domains D_1 and D_2. To overcome the mentioned technical difficulties, we use a deferred correction method [51] to solve (25) iteratively:

$$
\begin{aligned}
0 = &-\epsilon\left(\sum_{q\in\mathcal{N}_p}\frac{|e_g|}{|\mathbf{d}_{p'q'}|}\left(u_q^k+\nabla u_q^{k-1}\cdot\mathbf{d}_{qq'}-u_p^k-\nabla u_p^{k-1}\cdot\mathbf{d}_{pp'}\right)\right)\\
&-\epsilon\left(\sum_{b\in\mathcal{B}_p\cap\mathcal{B}_D}\frac{|e_b|}{|\mathbf{d}_{p'b}|}\left(u_b-u_p^k-\nabla u_p^{k-1}\cdot\mathbf{d}_{pp'}\right)+\sum_{b\in\mathcal{B}_p^+\setminus\mathcal{B}_D}\nabla u_p^{k-1}\cdot\mathbf{n}_b\right)\\
&+\sum_{f\in\mathcal{F}_p^-}\left(u_q^k+\mathcal{D}_q^{k-1}u\cdot\mathbf{d}_{qf}-u_p^k\right)\mu_{pf}+\sum_{f\in\mathcal{B}_p^+\cup\mathcal{F}_p^+}\left(\mathcal{D}_p u^{k-1}\cdot\mathbf{d}_{pf}\right)\mu_{pf}\\
&+\sum_{b\in\mathcal{B}_p^-\cap\mathcal{B}_D}\left(u_b-u_p^k\right)\mu_{pb}-|\Omega_p|,
\end{aligned}
\tag{26}
$$

where $k\in\mathbb{N}$ and $u^0=u_{\epsilon'}$. Keeping in mind the formulation above, we continue to discuss a decreasing sequence of regularization parameters ϵ and a pre-computed function $u_{\epsilon'}$ in (10) in the next subsection.

2.3. The regularization parameter ϵ

The vanishing viscosity method expects that the solution u_ϵ of (4) becomes close to the viscosity solution of (1) when the regularization parameter $\epsilon > 0$ is smaller and smaller. Similarly, we would like to find a numerical solution of (4) and (5) converges to the viscosity solution when the characteristic length h_L (9) becomes smaller and smaller. That is, a numerical convergence is related to not only the characteristic length h_L but also the regularization parameter ϵ. An empirical relation between h_L and ϵ to obtain a numerical convergence is that when h_L becomes smaller, the regularization parameter ϵ must become smaller too. Such a relation is also observed in solving a variant of the phase field model of the simplified Stefan problem [52].

Another aspect of the regularization parameter ϵ is that it cannot be too small in a fixed discretized domain. The reason is similar to that the time step cannot be too large in the time-relaxed eikonal equation (3). The direct effect of time relaxation in a linear system is to add positive values on a diagonal element which brings more stable computation to solve the linear system; see more details in [40]. When the time step is too large, the positive value being added to the diagonal elements is too small and then we can observe oscillation over time as it is already shown in [40]. Similarly, if the regularization parameter ϵ is too small on a fixed discretized domain, then the numerical solution does not become close enough to the viscosity solution of (1). The same phenomenon of a regularization parameter η is also observed in [53,54] by solving a singularly perturbed boundary value problem in [55] or the screened Poisson equation [56],

$$
\begin{aligned}
-\eta^2 \triangle w(\mathbf{x}) + w(\mathbf{x}) &= 0, \quad \mathbf{x}\in\Omega,\\
w(\mathbf{x}) &= 1, \quad \mathbf{x}\in\partial\Omega,
\end{aligned}
\tag{27}
$$

which can be transformed to

$$
\begin{aligned}
-\eta \triangle v(\mathbf{x}) + |\nabla v(\mathbf{x})|^2 &= 1, \quad \mathbf{x}\in\Omega,\\
v(\mathbf{x}) &= 0, \quad \mathbf{x}\in\partial\Omega,
\end{aligned}
\tag{28}
$$

by the Hopf–Cole transformation [57,58].

The obvious effect of using regularization parameter ϵ is to eliminate singularities and compute a smooth solution. However, if the parameter is too large, the numerical solution is not accurate enough to be the distance function. If it is too small, the numerical computation is not stable enough. Therefore, a reasonable choice of the regularization parameter ϵ is from a large value to a small value in a certain range. We choose regularization parameters as a decreasing sequence:

$$
\epsilon_n = \left(h_L\right)^{\frac{1}{2}n}, \quad n\in\mathbb{N},
\tag{29}
$$

where $h_L < 1$ is the characteristic length (9).

2.4. Proposed algorithm

In this subsection, combining the iterative algorithm (26) and a decreasing sequence ϵ_n of the regularization parameter (29), we propose an algorithm to compute a sequential numerical solution:

$$
\begin{aligned}
0 = &-\epsilon_n\left(\sum_{q\in\mathcal{N}_p}\frac{|e_g|}{|\mathbf{d}_{p'q'}|}\left(u_q^{n,k}+\nabla u_q^{n,k-1}\cdot\mathbf{d}_{qq'}-u_p^{n,k}-\nabla u_p^{n,k-1}\cdot\mathbf{d}_{pp'}\right)\right)\\
&-\epsilon_n\left(\sum_{b\in\mathcal{B}_p\cap\mathcal{B}_D}\frac{|e_b|}{|\mathbf{d}_{p'b}|}\left(u_b-u_p^{n,k}-\nabla u_p^{n,k-1}\cdot\mathbf{d}_{pp'}\right)\right.\\
&\left.+\sum_{b\in\mathcal{B}_p^+\setminus\mathcal{B}_D}\nabla u_p^{n,k-1}\cdot\mathbf{n}_b\right)\\
&+\sum_{f\in\mathcal{F}_p^-}\left(u_q^{n,k}+\mathcal{D}_q u^{n,k-1}\cdot\mathbf{d}_{qf}-u_p^{n,k}\right)\mu_{pf}^{n-1}\\
&+\sum_{f\in\mathcal{B}_p^+\cup\mathcal{F}_p^+}\left(\mathcal{D}_p u^{n,k-1}\cdot\mathbf{d}_{pf}\right)\mu_{pf}^{n-1}\\
&+\sum_{b\in\mathcal{B}_p^-\cap\mathcal{B}_D}\left(u_b-u_p^{n,k}\right)\mu_{pb}^{n-1}-|\Omega_p|, \quad k=1,\dots,K_n
\end{aligned}
\tag{30}
$$

where $u^0=0$, $u^{n,0}=u^{n-1}$ is a pre-computed solution of (10), K_n is defined by (32). The solution $u^{n,k}$ of (30) is computed by u^{n-1}, the parameter ϵ_n, and the k^{th} number of iterations in (30). Note that we explain how to numerically implement Dirichlet boundary condition (4) in the linear system (30) at the end of this subsection. Rewriting (30) formally as a matrix equation,

$$
\mathbf{A}^{n-1}u^{n,k}=\mathbf{f}(u^{n,k-1}),
\tag{31}
$$

an algebraic multigrid method is used to solve the above equation. The k^{th} iteration is stopped at the smallest K_n such that a residual error is smaller than a chosen error bound $\eta = 10^{-8}$:

$$
K_n=\min\left\{k\in\mathbb{N} : \rho^{n,k}=\frac{1}{|\mathcal{I}|}\sum_{p\in\mathcal{I}}\left|\left(\mathbf{A}^{n-1}\phi^{n,k}-\mathbf{f}(\phi^{n,k})\right)_p\right|<\eta\right\}, \quad n\geq 2,
\tag{32}
$$

where the parenthesis above with a subscript $(\mathbf{r})_p$ denotes the p^{th} component of the vector $\mathbf{r}$. Then, we define $u^n\equiv u^{n,K_n}$ for $n\geq 2$. When $n=1$, we use $K_n=1$. The proposed algorithm is also presented step by step in Algorithm 1.

Remark 1. In the matrix of the linear system (30) on the p^{th} row, the diagonal element is the coefficient of $u_p^{n,k}$ and all off-diagonal elements are the coefficients of $u_q^{n,k}$, $q\in\mathcal{N}_p$. It means the system only uses neighbor cells across faces of Ω_p. Then, an implementation of (30) in a standard cell-centered FVM code is straightforwardly done for parallel computing using domain decomposition with 1-ring face neighborhood.

Algorithm 1 A procedure to compute a numerical solution of (4) and (5).

```
procedure
    Initialization $u^0 = 0$.
    Set $n = 1$ and $K_1 = 1$.
    Find a solution $u^1 = u^{1,1}$ of (30) with $u^{1,0} = u^0 = 0$.
    for $n \leftarrow 2$ to 5 do
        Set $u^{n,0} = u^{n-1}$.
        Set $k = 1$.
        while $\rho^{n,k} \geq \eta$ do                          ▷ See (32).
            Find a solution $u^{n,k}$ of (30) with $u^{n,k-1}$.
            $k \leftarrow k + 1$
        end while
    end for
end procedure
```

When $n = 1$ in the proposed algorithm (30), the linear system computes a solution of the equation below because all gradients are zero with the initial choice $u^{1,0} = u^0 = 0$:

$$
\begin{aligned}
-\epsilon \triangle \bar{u}(\mathbf{x}) &= 1 \quad \mathbf{x} \in \Omega \setminus \Gamma, \\
\bar{u}(\mathbf{x}) &= 0 \quad \mathbf{x} \in \Gamma, \\
\boldsymbol{\nu}(\mathbf{x}) \cdot \nabla \bar{u}(\mathbf{x}) &= 0 \quad \mathbf{x} \in \partial\Omega \setminus \Gamma.
\end{aligned} \tag{33}
$$

The direction of $\nabla \bar{u}$ is the same as the gradient of the viscosity solution in (1) because their zero level set Γ is identical. Then, for $n \geq 2$, the normalized vector $\mathbf{v}$ in (10) on Γ is already same as the vector $\mathbf{v}$ computed by the viscosity solution of (1). In the case of $\Gamma = \partial\Omega$, the solution of Poisson equation (33) is also used to approximate a distance function on a close neighborhood of Γ by a normalization scheme [59]. In [60], it is argued that there is a proximity in L^2 sense between the solution of (33) and the distance function from Γ.

In order to complete the description of the proposed algorithm, we need to explain how the boundary value on Γ is implemented in a polyhedron mesh because $\Gamma \subset \bar{\Omega}$ is generally located on a given mesh. To do so, we define index sets to select the cells where Γ is located in $\bar{\Omega}$:

$$
\begin{aligned}
\mathcal{I}^1 &= \left\{ p \in \mathcal{I} : \bar{\Omega}_p \cap \Gamma \neq \emptyset,\ \Gamma \subseteq \partial\Omega \right\}, \\
\mathcal{I}^2 &= \left\{ p \in \mathcal{I} : \bar{\Omega}_p \cap \Gamma \neq \emptyset,\ \Gamma \subsetneq \Omega,\ \Gamma \cap \partial\Omega = \emptyset \right\}.
\end{aligned} \tag{34}
$$

If Γ is a general shape, an octree search and point-in-cell algorithms are used to define the index sets above. Let us define a function $\mathcal{F} : K \subset \mathcal{I} \to \mathcal{I}$ by $\mathcal{F}(K) = \left\{ q \in \mathcal{I} : q \in \mathcal{N}_p,\ \forall p \in K \right\} \cup K$. Now, we use the set $\Gamma^0 = \mathcal{F}(\mathcal{F}(\mathcal{I}^2)) \cup \mathcal{F}(\mathcal{I}^1)$ and it is straightforward to compute the exact distance value from Γ at all points $\mathbf{x}_p$, $p \in \Gamma^0$. An octree search algorithm can find a short list of potential elements in Γ to compute the shortest distance from $\mathbf{x}_p$ to Γ and it is efficient enough because all points $\mathbf{x}_p$, $p \in \Gamma^0$, are close to Γ. Then, the computed distance value on Γ^0 is used in the proposed algorithm. That is, on the p^{th} row of the matrix (30), $p \in \Gamma^0$, we use the value and make all relevant off-diagonal element of Ω_p to be zero in the matrix.

3. Numerical results

We present various examples to show the numerical properties of the proposed algorithm (30). The meshes generated by `AVL FIRE`™ are illustrated in Fig. 2 and the number of polyhedral cells $|\mathcal{I}_{\text{L}}|$ and the characteristic length h_{L} (9) of the meshes are presented for four levels of meshes, $\text{L} \in \{1, 2, 3, 4\}$, in Table 1. Note that $h_{\text{L}+1} < h_{\text{L}}$. The test examples are basically to compute a distance function from Γ on a given computation domain Ω and all details are explained below with constants $\gamma_i = \frac{R_i}{15}$ for $i = 1, 2$, where $R_1 = 1.25$ and $R_2 = 10$.

EX1 Γ is a sphere with the center at the origin and the radius is 0.6 in the computational domain $\Omega = [-R_1, R_1]^3$. The mesh $\mathcal{M}_L^1$ is used in Table 1. The first level of mesh is shown in Fig. 2-(a).

EX2 Γ is a sphere at the origin with the radius is 0.3 in the computational domain $\Omega = [-8\gamma_1, 22\gamma_1] \times [-15\gamma_1, 15\gamma_1] \times [-15\gamma_1, 15\gamma_1] \setminus \Omega'$, where $\Omega' = [8\gamma_1, 15\gamma_1] \times [-15\gamma_1, 15\gamma_1] \times [-5\gamma_1, 5\gamma_1]$. The mesh $\mathcal{M}_L^2$ is used in Table 1. The first level of mesh is shown in Fig. 2-(b).

EX3 The computational domain is $\Omega = [-15\gamma_1, 15\gamma_1] \times [-15\gamma_1, 15\gamma_1] \times [-5\gamma_1, 5\gamma_1] \setminus \Omega'$, where $\Omega' = [-5\gamma_1, 15\gamma_1] \times [-5\gamma_1, 5\gamma_1] \times [-5\gamma_1, 5\gamma_1]$ and $\Gamma = \{15\gamma_1\} \times [5\gamma_1, 15\gamma_1] \times [-5\gamma_1, 5\gamma_1]$ is the upper right plane. The mesh $\mathcal{M}_L^3$ is used in Table 1. The first level of mesh is shown in Fig. 2-(c).

EX4 The computational domain Ω is same as EX3 and $\Gamma = \{15\gamma_1\} \times [5\gamma_1, 15\gamma_1] \times [-5\gamma_1, 5\gamma_1] \cup \{15\gamma_1\} \times [-15\gamma_1, -5\gamma_1] \times [-5\gamma_1, 5\gamma_1]$ is the upper right and the lower right planes in Fig. 2-(c). The mesh $\mathcal{M}_L^3$ is used in Table 1.

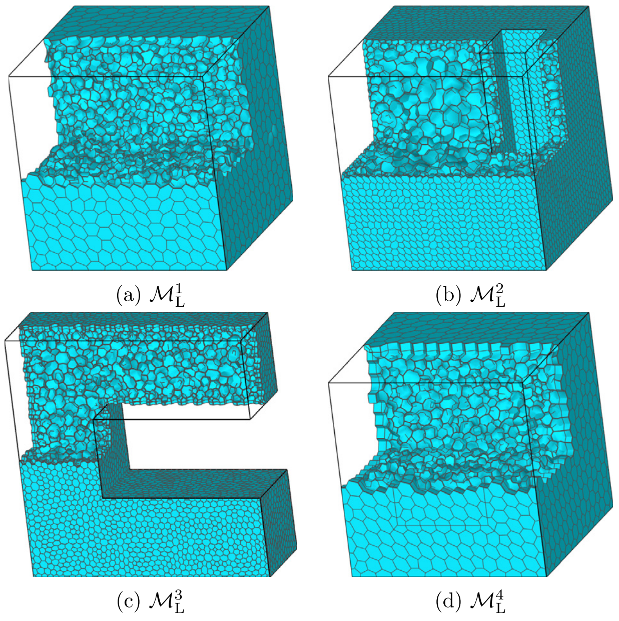

(a) $\mathcal{M}_{\text{L}}^1$
(b) $\mathcal{M}_{\text{L}}^2$
(c) $\mathcal{M}_{\text{L}}^3$
(d) $\mathcal{M}_{\text{L}}^4$

Fig. 2. It is an illustration of meshes for computational domains in Table 1 with $L = 1$. The bold black lines are the boundary of the computational domain. The polyhedral cells inside the domain are presented. Note that the right side is the positive direction of x axis, the top side is the positive direction of y axis, and the direction coming out of the paper is the positive direction of z axis.

Table 1
The numbers of polyhedral cells $\mathcal{I}_{\text{L}}$ and the characteristic length h_{L} (9) of the meshes are presented; see the shape of the computational domains at the level $\text{L} = 1$ in Fig. 2.

mesh	L	$\lvert\mathcal{I}_{\text{L}}\rvert$	h_{L}
$\mathcal{M}_{\text{L}}^1$	1	10 421	$1.54 \cdot 10^{-1}$
	2	48 516	$9.24 \cdot 10^{-2}$
	3	331 146	$4.87 \cdot 10^{-2}$
	4	2 301 237	$2.55 \cdot 10^{-2}$
$\mathcal{M}_{\text{L}}^2$	1	14 821	$1.17 \cdot 10^{-1}$
	2	70 859	$6.89 \cdot 10^{-2}$
	3	363 418	$4.08 \cdot 10^{-2}$
	4	2 153 388	$2.34 \cdot 10^{-2}$
$\mathcal{M}_{\text{L}}^3$	1	18 118	$7.02 \cdot 10^{-2}$
	2	74 301	$4.22 \cdot 10^{-2}$
	3	362 679	$2.44 \cdot 10^{-2}$
	4	1 868 820	$1.45 \cdot 10^{-2}$
$\mathcal{M}_{\text{L}}^4$	1	7863	$6.52 \cdot 10^{-1}$
	2	58 091	$3.42 \cdot 10^{-1}$
	3	457 436	$1.73 \cdot 10^{-1}$
	4	3 660 530	$8.71 \cdot 10^{-2}$

EX5 The computational domain Ω is same as EX3 and $\Gamma = \partial\Omega$. The mesh $\mathcal{M}_L^3$ is used in Table 1.

EX6 The computational domain is $\Omega = \left[-\frac{R_2}{2}, \frac{R_2}{2}\right]^3$ and $\Gamma = \partial\Omega$. The mesh $\mathcal{M}_L^4$ is used in Table 1.

EX7 The computational domain is the same as EX6 and Γ is a circle with the center at the origin and the radius 0.6, where the normal vector of the plane containing the circle is z axis. The mesh $\mathcal{M}_L^4$ is used in Table 1.

EX8 The computational domain is same as EX6 and Γ is a disk whose boundary is the circle in EX7. The mesh $\mathcal{M}_L^4$ is used in Table 1.

EX9 The computational domain is $\Omega = [-R_2, R_2]^3$ and Γ is a square with the center at the origin and the length of the side is $7r_2$, where the normal vector of the plane containing the square is z axis. The mesh $\mathcal{M}_L^4$ is used in Table 1. The first level of mesh is shown in Fig. 2-(d).

EX10 The computational domain is same as EX9 and Γ is two squares of the same size used in EX9. The center of the first and second square is located at $(0, 0, 7.5\gamma_2)$ and $(0, 0, -7.5\gamma_2)$, respectively. The mesh $\mathcal{M}_L^4$ is used in Table 1.

The exact solutions from EX1 to EX5 are already presented in [40] and the exact solutions from EX6 to EX10 can be analytically obtained. Note that the polyhedron meshes $\mathcal{M}^2$ and $\mathcal{M}^3$ are exactly the same as the one in [40]. A typical body-fitted surface mesh is used on two squares Γ in the case of EX10. In Fig. 2-(d), half of a small square is visible on the boundary of $\mathcal{M}_L^4$, $L = 1$. The same mesh is used to test cases from EX6 to EX10.

Prior to the numerical properties of the proposed algorithm, equidistant isosurfaces of numerical solutions computed by the proposed algorithm (30) are presented in Fig. 3 on the level $L = 4$ in Table 1. They are qualitatively shown as a distance function from a given Γ illustrated by the color of dark red. In the cases of EX1, EX2, and EX10, we use a transparency on Γ to visually observe isosurfaces behind Γ. In the cases of EX5 and EX6, the surface Γ is not presented because $\Gamma = \partial\Omega$.

The first numerical property is an experimental order of convergence (EOC). Since exact solutions for all examples are known, we compute the errors E_{L}^1 and E_{L}^∞ of L^1 and L^∞ norms between a numerical solution on the L^th level of mesh and an exact solution, respectively. Then, for each error, the corresponding EOC is calculated by

$$EOC_{\mathrm{L}} = \frac{\log\left(\frac{E_{\mathrm{L}+1}}{E_{\mathrm{L}}}\right)}{\log\left(\frac{h_{\mathrm{L}+1}}{h_{\mathrm{L}}}\right)}, \quad \mathrm{L} \in \{1, 2, 3\}. \tag{35}$$

In Table 2, we present EOCs of all examples for a numerical solution of the proposed algorithm (30) with ϵ_n, $n = 5$. For smooth solutions of EX8 and EX9, the EOCs with E^1 and E^∞ errors are close to 2. In EX1, the EOCs with E^1 is larger than 2, but the EOCs with E^∞ is close to 1 because of a singularity at the origin. For all non-smooth solutions, the EOCs with E^1 and E^∞ errors are close to 1. Compared to the EOCs in [40], the behavior of EOC is quite similar.

The second numerical property is the behavior of the errors versus the regularization parameter ϵ_n on a fixed level of meshes. For each n on the L^th level of mesh, the proposed algorithm (30) provides a numerical solution u^n with $\epsilon_n = (h_{\mathrm{L}})^{\frac{1}{2}n}$. For the next $n+1$, we use the solution u^n and then find the next solution u^{n+1} with ϵ_{n+1} $(< \epsilon_n)$. In Table 3, for the case of EX1, errors E^1 and E^∞ of numerical solutions u^n with ϵ_n from $n = 2$ to $n = 5$ are presented on all levels of meshes.

A crucial observation is that the choice of $\epsilon_5 = h_L^{\frac{5}{2}}$ brings a better result, that is, smaller errors, than the other regularization values ϵ_n for $1 \le n \le 4$. Since we use $K_1 = 1$ in (30), the results of $n = 1$ are far from the exact solution. On a fixed level of mesh, when the regularization parameter ϵ_n is smaller, that is, n becomes larger, the errors E^1 and E^∞ become smaller until $n = 5$. The mentioned property can be seen on the rows with the same gray color in Table 3. For example, when $\mathrm{L} = 1$, by the value on the second row of E^1 column, the error on every fourth row below in the same column decreases; see the error values shadowed by the darkest gray color in Table 3. Also, the EOCs on different levels of meshes become better from $n = 2$ to $n = 5$. In order to check similar phenomena for all other examples, we provide a graph version of Table 3 from Figs. 4 to 8. When $n \ge 6$, the effect of the Laplacian regularizer is too small to solve the linear system (30) stably enough. A similar instability of using too small regularization parameter is also observed in [17,53,61]. A relation between the regularization parameter and the order of the numerical scheme is also observed in [58]. A further numerical analysis is necessary to find an optimal regularization parameter to minimize an error between a numerical solution on a discrete space of (4) and (5) and a viscosity solution of (1), which is out of the scope of this paper.

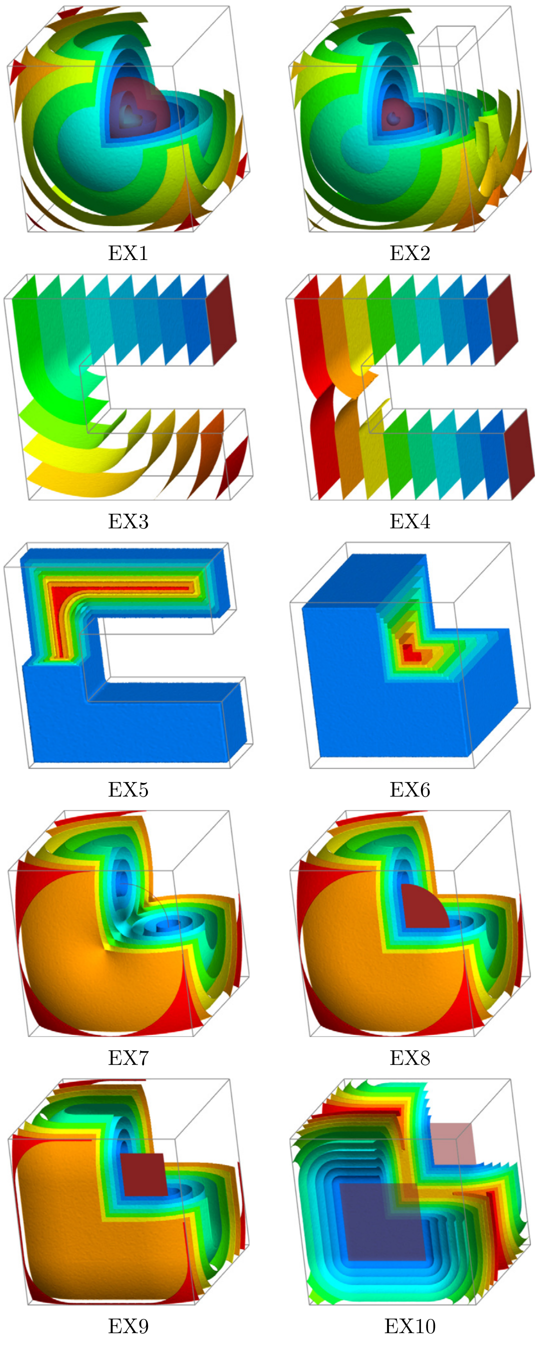


Fig. 3. Iso-surfaces of numerical solutions computed by the proposed algorithm (30) are presented on the level $L = 4$ in Table 1.

Table 2
The EOCs (35) of all examples for a numerical solution of (30) with ϵ_n, $n = 5$, are presented. L is the level of mesh listed in Table 1.

	L	E^1	EOC	E^∞	EOC
EX1	1	$6.37 \cdot 10^{-3}$	1.76	$4.95 \cdot 10^{-2}$	3.65
	2	$2.60 \cdot 10^{-3}$	2.10	$7.68 \cdot 10^{-3}$	1.41
	3	$6.77 \cdot 10^{-4}$	2.21	$3.10 \cdot 10^{-3}$	0.75
	4	$1.62 \cdot 10^{-4}$		$1.91 \cdot 10^{-3}$	
EX2	1	$1.04 \cdot 10^{-2}$	1.43	$1.08 \cdot 10^{-1}$	1.45
	2	$4.91 \cdot 10^{-3}$	1.75	$5.03 \cdot 10^{-2}$	0.92
	3	$1.96 \cdot 10^{-3}$	1.78	$3.11 \cdot 10^{-2}$	1.28
	4	$7.28 \cdot 10^{-4}$		$1.53 \cdot 10^{-2}$	
EX3	1	$1.34 \cdot 10^{-2}$	1.50	$3.88 \cdot 10^{-2}$	1.26
	2	$6.23 \cdot 10^{-3}$	1.22	$2.04 \cdot 10^{-2}$	1.24
	3	$3.19 \cdot 10^{-3}$	1.10	$1.04 \cdot 10^{-2}$	1.06
	4	$1.81 \cdot 10^{-3}$		$5.97 \cdot 10^{-3}$	
EX4	1	$3.33 \cdot 10^{-3}$	1.27	$5.61 \cdot 10^{-2}$	1.02
	2	$1.74 \cdot 10^{-3}$	1.17	$3.33 \cdot 10^{-2}$	1.03
	3	$9.20 \cdot 10^{-4}$	1.25	$1.90 \cdot 10^{-2}$	1.40
	4	$4.81 \cdot 10^{-4}$		$9.19 \cdot 10^{-3}$	
EX5	1	$5.95 \cdot 10^{-3}$	1.04	$5.78 \cdot 10^{-2}$	0.91
	2	$3.49 \cdot 10^{-3}$	1.92	$3.64 \cdot 10^{-2}$	1.12
	3	$1.22 \cdot 10^{-3}$	2.24	$1.97 \cdot 10^{-2}$	0.78
	4	$3.82 \cdot 10^{-4}$		$1.32 \cdot 10^{-2}$	
EX6	1	$9.29 \cdot 10^{-2}$	4.33	$7.35 \cdot 10^{-1}$	2.95
	2	$5.73 \cdot 10^{-3}$	1.78	$1.10 \cdot 10^{-1}$	1.76
	3	$1.70 \cdot 10^{-3}$	1.69	$3.32 \cdot 10^{-2}$	0.61
	4	$5.34 \cdot 10^{-4}$		$2.18 \cdot 10^{-2}$	
EX7	1	$3.12 \cdot 10^{-1}$	1.40	$6.67 \cdot 10^{-1}$	1.63
	2	$1.26 \cdot 10^{-1}$	1.91	$2.33 \cdot 10^{-1}$	2.09
	3	$3.44 \cdot 10^{-2}$	1.93	$5.60 \cdot 10^{-2}$	2.03
	4	$9.09 \cdot 10^{-3}$		$1.38 \cdot 10^{-2}$	
EX8	1	$2.89 \cdot 10^{-1}$	1.47	$6.53 \cdot 10^{-1}$	1.62
	2	$1.12 \cdot 10^{-1}$	1.94	$2.30 \cdot 10^{-1}$	2.08
	3	$2.99 \cdot 10^{-2}$	1.95	$5.61 \cdot 10^{-2}$	2.00
	4	$7.78 \cdot 10^{-3}$		$1.42 \cdot 10^{-2}$	
EX9	1	$2.60 \cdot 10^{-1}$	1.39	$7.81 \cdot 10^{-1}$	1.38
	2	$1.06 \cdot 10^{-1}$	1.78	$3.22 \cdot 10^{-1}$	1.93
	3	$3.17 \cdot 10^{-2}$	1.84	$8.64 \cdot 10^{-2}$	1.80
	4	$8.95 \cdot 10^{-3}$		$2.49 \cdot 10^{-2}$	
EX10	1	1.01	1.86	2.06	1.60
	2	$3.05 \cdot 10^{-1}$	1.73	$7.37 \cdot 10^{-1}$	1.62
	3	$9.45 \cdot 10^{-2}$	1.51	$2.44 \cdot 10^{-1}$	1.41
	4	$3.34 \cdot 10^{-2}$		$9.26 \cdot 10^{-2}$	

Table 3
For the case of EX1, errors E^1 and E^∞ of numerical solutions (30) with ϵ_n from $n = 2$ to $n = 5$ are presented on all levels of meshes. From a fixed ϵ_n, the EOCs are also shown on different levels of meshes.

L	ϵ_n	E^1	EOC	E^∞	EOC
1	h_L^1	9.87E-02	−0.23	2.52E-01	−0.14
2		1.11E-01	0.44	2.70E-01	0.23
3		8.40E-02	0.59	2.34E-01	0.24
4		5.72E-02		2.00E-01	
1	$h_L^{\frac{3}{2}}$	3.63E-02	0.74	1.20E-01	0.56
2		2.48E-02	1.10	9.05E-02	0.81
3		1.23E-02	1.18	5.40E-02	1.05
4		5.73E-03		2.74E-02	
1	h_L^2	1.52E-02	1.33	7.47E-02	1.79
2		7.72E-03	1.66	3.00E-02	1.36
3		2.67E-03	1.87	1.26E-02	1.10
4		7.98E-04		6.20E-03	
1	$h_L^{\frac{5}{2}}$	6.37E-03	1.76	4.95E-02	3.65
2		2.60E-03	2.10	7.68E-03	1.41
3		6.77E-04	2.21	3.10E-03	0.75
4		1.62E-04		1.91E-03	

The third numerical property is a comparison of computational cost. To minimize a systematical bias, we purposely choose the time-relaxed bidirectional eikonal equation [40] already implemented in AVL FIRE™. The proposed algorithm is also implemented in the same language (Fortran 2003) and all algorithms are compiled by the same compiler options.

Since the time-relaxed bidirectional eikonal equation is time-dependent and the governing equation in this paper is time-independent, we stop the time evolution in (3) right before the E^1 error of (3) becomes smaller than the E^1 error of the proposed algorithm. That is, we measure a computational cost until two methods reach the same error bound. In Table 4, such a final time T is shown on the column labeled by "Final T" for all examples. On that column, T_M means that E^1 error of (3) is not smaller than the E^1 error of the proposed algorithm until the predetermined final time T_M, specified in [40]. Time^1 and Time^2 are the computation time in seconds for the proposed algorithm (30) and the algorithm in [40], respectively, and the corresponding total number of iterations are shown right next to the computational time. The calculations of using $2 \cdot$ L numbers of CPUs for all examples in the L$^{\text{th}}$ level of mesh are repeated five times in a shared memory system (Intel® Core™ Processor i7-8700K CPU 3.70 GHz 12 CPUs and 62 gigabyte memory). The computational time (Time^1 and Time^2) in Table 4 is the average of five measurements. Since the distance information in (3) is evolved from Γ over time, the time-relaxed bidirectional equation has certainly a disadvantage in computational time whenever it is necessary to compute a distance further away from Γ. The last column shows how much the proposed algorithm is faster than the algorithm to solve the time-relaxed bidirectional eikonal equation to reach the same E^1 error. In the case of EX5, the time ratio is quite different from other examples because $\Gamma = \partial\Omega$ makes the traveling distance much shorter than other examples. In other words, the computational time of the proposed algorithm becomes faster than the previous approach [40] as long as the region of interest to find distance values is far away from Γ.

In the last example, we would like to show the sequential results along the decreasing regularization parameter ϵ_n and we present a qualitative comparison of the proposed algorithm (30) between polyhedron and hexahedron mesh on a box shape of the computational domain $[-0.101, 0.111] \times [-0.061, 0.091] \times [-0.049, 0.053]$. The number of cells and the characteristic length on polyhedron mesh are $N = 2452429$ and $h = 1.49 \cdot 10^{-3}$. For the hexahedron mesh, we have $N = 3105000$ and $h = 1.02 \cdot 10^{-3}$. We use the residual error bound $\eta = 10^{-6}$ in (32). The given surface Γ, the Dragon, is illustrated in Fig. 10 from the Stanford 3D scanning repository.[1] In Fig. 10, from the top to the bottom, we present equidistance isosurfaces from the solutions of (30) on the polyhedron mesh with the decreasing regularization parameters ϵ_n, $n = 1, \dots, 4$. The results of the first ϵ_1 are far away from the distance function of the Dragon surface. However, the results of the second ϵ_2 are dramatically improved because the normalized gradient vectors of the first result on the Dragon surface are already the same as the vectors computed by the viscosity solution of the eikonal equation. In Fig. 11, the results in the first row are computed on the polyhedron mesh with $\epsilon_5 = h^{\frac{5}{2}}$ and they are almost the same as the results in the last row ($\epsilon_4 = h^{\frac{4}{2}}$) in Fig. 10. The results in the second row in Fig. 11 are computed on the hexahedron mesh with $\epsilon_5 = h^{\frac{5}{2}}$. They are nearly the same results on the polyhedron mesh because the characteristic lengths of two meshes are deliberately chosen to be a similar size.

4. Conclusion

We present a cell-centered finite volume method to solve a Laplacian regularized eikonal equation with Soner boundary condition on

[1] http://graphics.stanford.edu/data/3Dscanrep.

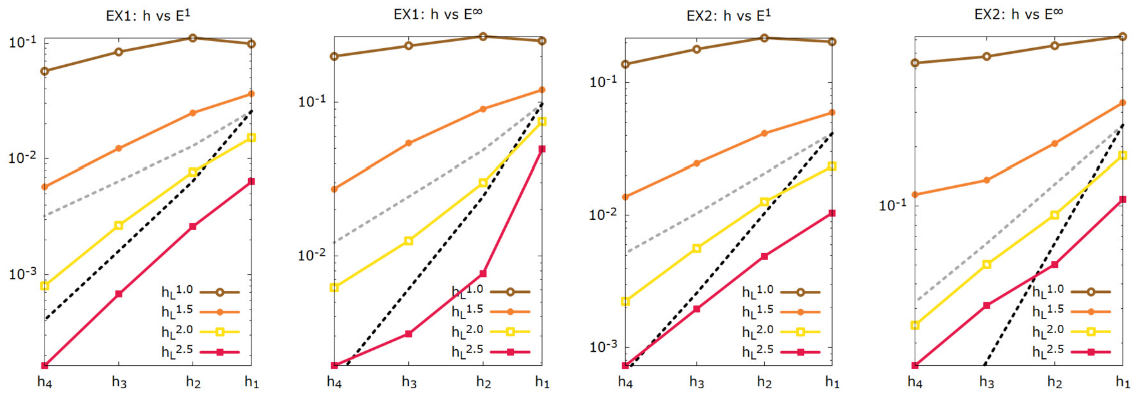


Fig. 4. For the cases of EX1 and EX2, the graphs h_L versus E^1 (or E^∞) are presented in the log-log scale. The gray and black dotted lines show the first and second order convergence, respectively.

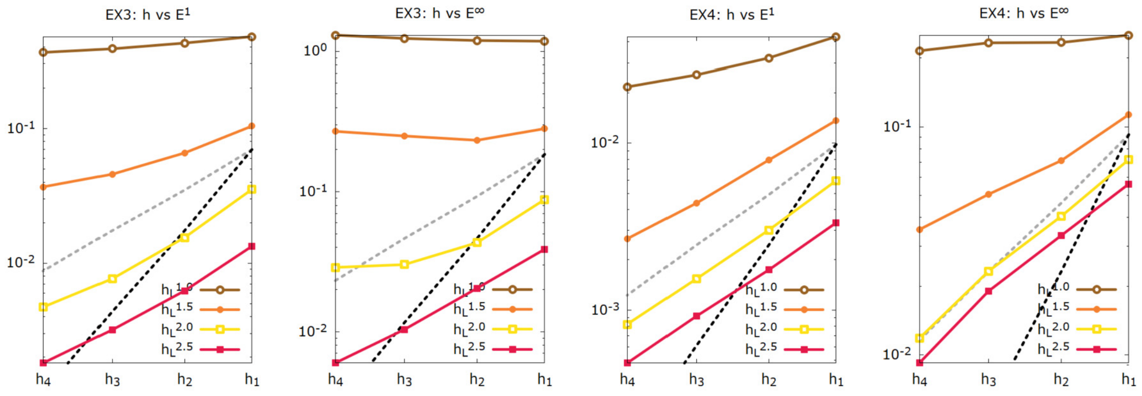


Fig. 5. For the cases of EX3 and EX4, the graphs h_L versus E^1 (or E^∞) are presented in the log-log scale. The gray and black dotted lines show the first and second order convergence, respectively.

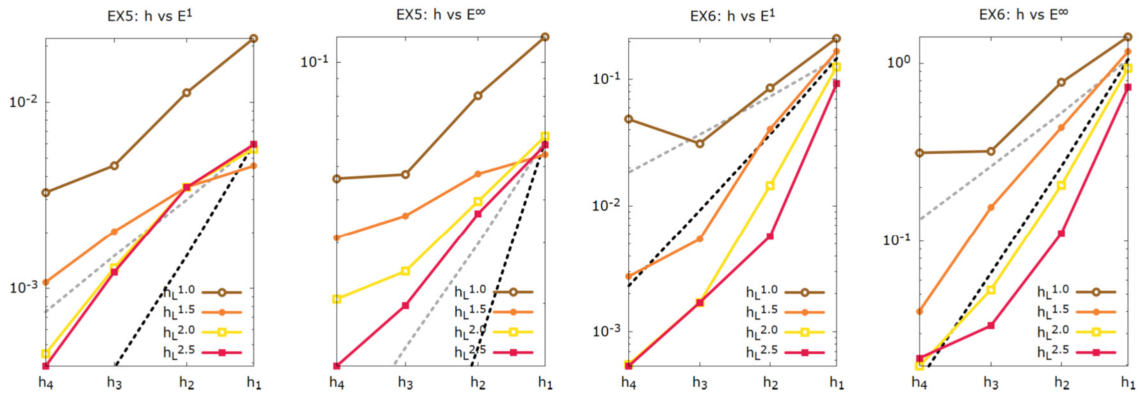


Fig. 6. For the cases of EX5 and EX6, the graphs h_L versus E^1 (or E^∞) are presented in the log-log scale. The gray and black dotted lines show the first and second order convergence, respectively.

polyhedral meshes in order to compute a distance function from given objects. Using a linearized form of the equation, a numerical solution is sequentially updated by a decreasing sequence of the regularization parameters depending on a characteristic length of discretized domain. The normalized gradient field of the first solution in the sequence is substantially improved on the given objects. As the characteristic length becomes smaller, the regularization parameter becomes smaller and a convergence to the viscosity solution is numerically verified. The EOC of L^1 norm of the error is shown to be the second order for tested smooth solutions. Compared to the computational time of solving the time-relaxed bidirectional eikonal equation, the proposed algorithm has the advantage to dramatically reducing the time when a larger number of cells is used or a region of interest is far away from where the distance measurement starts. The implementation of parallel computing using domain decomposition with the 1-ring face neighbor structure can be done straightforwardly by a standard cell-centered finite volume code.

Data availability

Data will be made available on request.

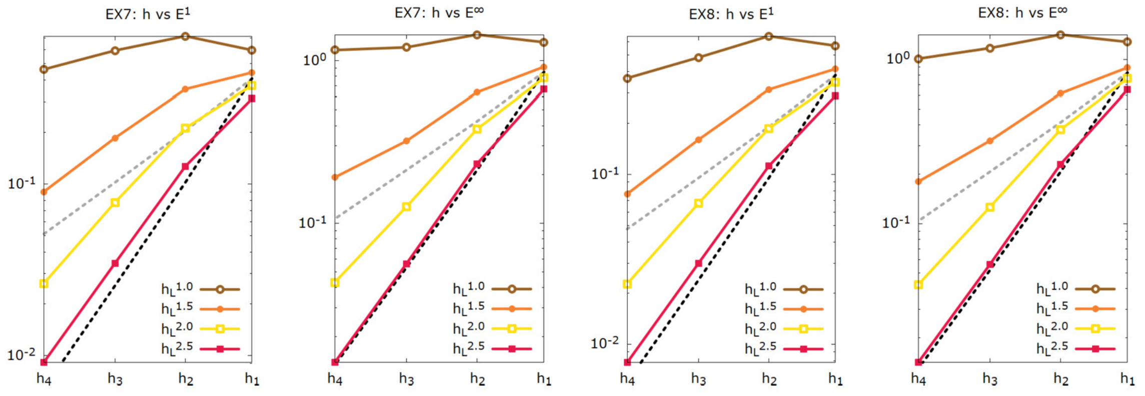


Fig. 7. For the cases of EX7 and EX8, the graphs h_L versus E^1 (or E^∞) are presented in the log-log scale. The gray and black dotted lines show the first and second order convergence, respectively.

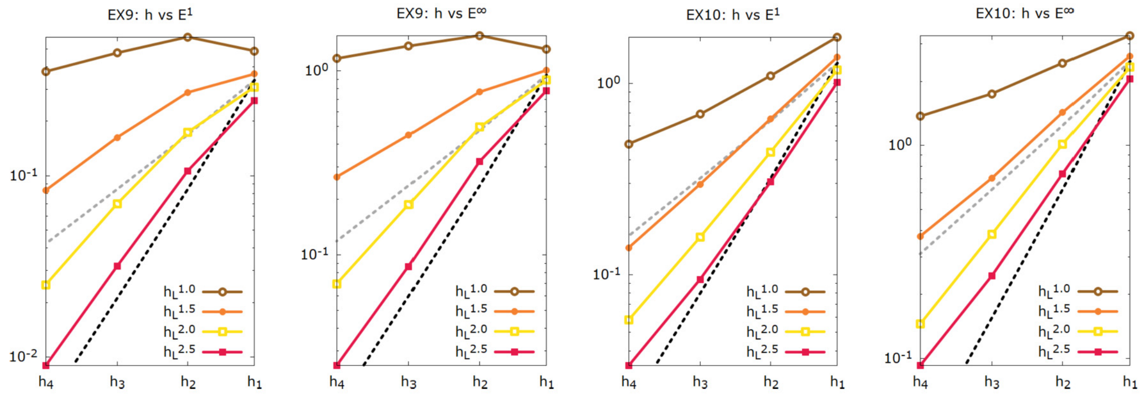


Fig. 8. For the cases of EX9 and EX10, the graphs h_L versus E^1 (or E^∞) are presented in the log-log scale. The gray and black dotted lines show the first and second order convergence, respectively.

Table 4
For all examples, a comparison of computational cost is presented by using $2 \cdot \mathrm{L}$ numbers of CPUs on the L^{th} level of mesh. Time^1 and Time^2 are the computation time in seconds of the proposed algorithm (30) and the algorithm in [40], respectively, and the corresponding total number of iterations are shown right next to the computational time. The final T to solve (3) is decided by the same E^1 error value as the proposed method; see more details in Section 3.

	L	Time^1 (s)	$\sum_{n=1}^{5} K_n$	Final T	Time^2 (s)	N_{tot}	Ratio
EX1	1	5.04	33	1.200	6.59	30	1.31
	2	10.72	24	1.280	38.42	64	3.58
	3	66.37	16	1.360	625.15	136	9.42
	4	438.30	12	1.430	7378.86	286	16.84
EX2	1	7.95	43	1.640	10.58	41	1.33
	2	20.25	35	1.680	67.18	84	3.32
	3	92.46	25	1.730	774.62	173	8.38
	4	488.79	15	1.775	8250.24	355	16.88
EX3	1	9.73	42	4.080	24.97	102	2.56
	2	19.23	27	4.220	126.51	105	6.58
	3	75.46	13	4.210	1473.19	421	19.52
	4	304.69	8	4.225	13 159.04	845	43.19
EX4	1	7.69	29	2.680	16.54	67	2.15
	2	15.72	19	2.980	90.00	149	5.73
	3	63.34	10	2.940	1031.72	294	16.29
	4	233.38	6	2.870	8929.88	574	38.26
EX5	1	5.68	20	$T_M = 2$	12.44	50	2.19
	2	12.63	16	$T_M = 2$	60.38	100	4.78
	3	54.87	12	$T_M = 2$	708.74	200	12.92
	4	173.09	7	0.630	1951.68	126	11.28

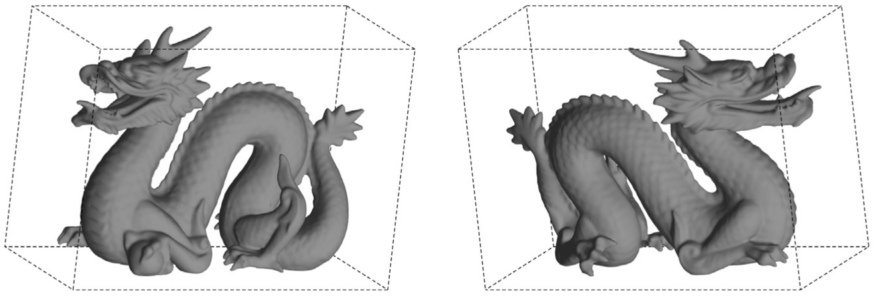
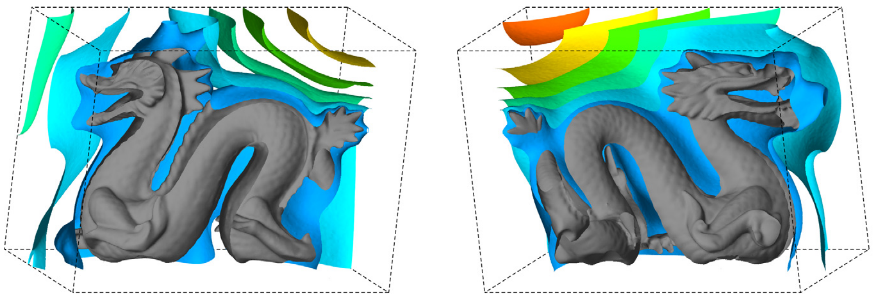
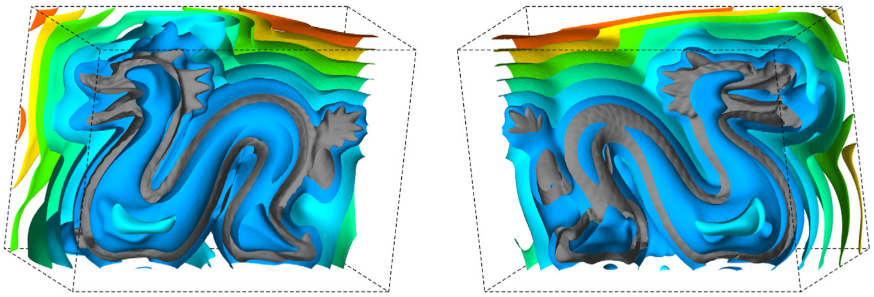


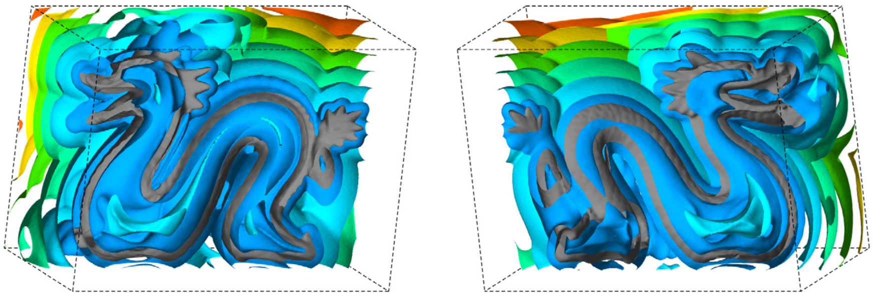
Fig. 9. Two sides of dragon surface Γ.



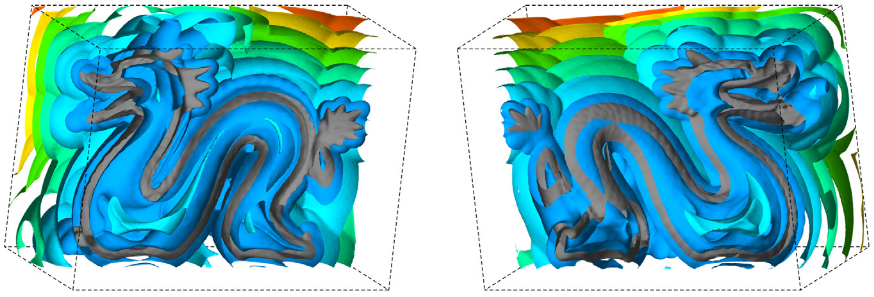
$\epsilon_1 = h^{\frac{1}{2}}$



$\epsilon_2 = h^{\frac{2}{2}}$

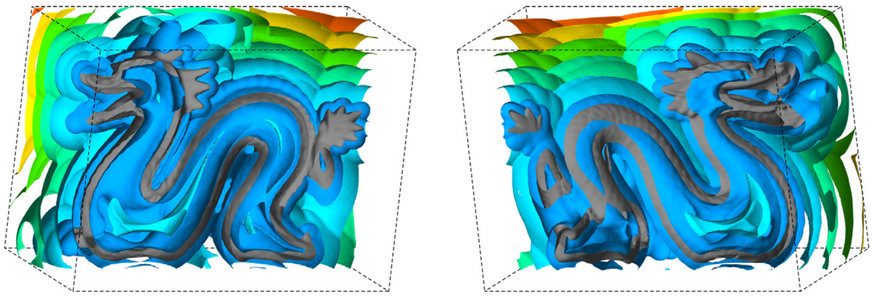


$\epsilon_3 = h^{\frac{3}{2}}$



$\epsilon_4 = h^{\frac{4}{2}}$

Fig. 10. From the top to the bottom, we present equidistance isosurfaces from the solutions of (30) on the polyhedron mesh with the decreasing regularization parameters.


$\epsilon_5 = h^{\frac{5}{2}}$, polyhedron mesh

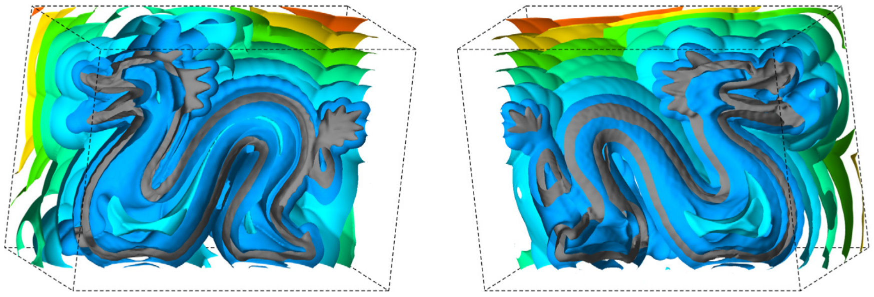

$\epsilon_5 = h^{\frac{5}{2}}$, hexahedron mesh

Fig. 11. The qualitative comparison of equidistance isosurfaces from the dragon surface in Fig. 9 is shown by the results computed on polyhedron and hexahedron mesh with a similar size of the characteristic length.

Acknowledgements

The authors thank the anonymous reviewers for their valuable suggestions. The authors also thank Prof. Silvia Tozza in University of Bologna, Italy, for the comments of the Soner boundary condition. We also sincerely thank Dr. Branislav Basara and Dr. Reinhard Tatschl in AVL List GmbH, Austria, for supporting the University Partnership Program.[2]

References

[1] J.A. Sethian, Level Set Methods and Fast Marching Methods, Evolving Interfaces in Computational Geometry Fluid Mechanics, Computer Vision, and Materials Science, Cambridge University Press, New York, 1999.
[2] S. Osher, R. Fedkiw, Level Set Methods and Dynamic Implicit Surfaces, Springer, Berlin, 2000.
[3] N. Peters, Turbulent Combustion, Cambridge Monographs on Mechanics, Cambridge University Press, 2000.
[4] D. Suckart, D. Linse, E. Schutting, H. Eichlseder, Experimental and simulative investigation of flame-wall interactions and quenching in spark-ignition engines, Automot. Engine Technol. 2 (1) (2017) 25–38.
[5] A. Manz, Modeling of End-Gas Autoignition for Knock Prediction in Gasoline Engines, Logos Verlag Berlin, 2016.
[6] B. Baldwin, H. Lomax, Thin-layer approximation and algebraic model for separated turbulentflows, in: American Institute of Aeronautics and Astronautics 16th Aerospace Sciences Meeting, 1978, 78-257.
[7] B. Baldwin, T. Barth, A one-equation turbulence transport model for high Reynolds number wall-bounded flows, in: American Institute of Aeronautics and Astronautics 29th Aerospace Sciences Meeting, 1991, 91-0610.
[8] P. Spalart, S. Allmaras, A one-equation turbulence model for aerodynamic flows, AIAA 1992-439, in: 30th Aerospace Sciences Meeting and Exhibit, January 1992.
[9] E. Fares, W. Schröder, A differential equation for approximate wall distance, Int. J. Numer. Methods Fluids 39 (2002) 743–762.
[10] P.G. Tucker, Differential equation-based wall distance computation for DES and RANS, J. Comput. Phys. 190 (2003) 229–248.
[11] H. Xia, P.G. Tucker, Finite volume distance field and its application to medial axis transforms, Int. J. Numer. Methods Eng. 82 (2010) 114–134.
[12] H. Xia, P.G. Tucker, Fast equal and biased distance fields for medial axis transform with meshing in mind, Appl. Math. Model. 35 (2011) 5804–5819.
[13] M.A. Price, C.G. Armstrong, M.A. Sabin, Hexahedral mesh generation by medial surface subdivision: Part I. Solids with convex edges, Int. J. Numer. Methods Eng. 38 (1995) 3335–3359.
[14] W.R. Quadros, K. Ramaswami, F.B. Prinz, B. Gurumoorthy, Laytracks: a new approach to automated geometry adaptive quadrilateral mesh generation using medial axis transform, Int. J. Numer. Methods Eng. 61 (2004) 209–237.
[15] P. Colli-Franzone, L. Guerri, Spreading of excitation in 3-D models of the anisotropic cardiac tissue. I. Validation of the eikonal model, Math. Biosci. 113 (1993) 145–209.
[16] J.P. Keener, An eikonal-curvature equation for action potential propagation in myocardium, J. Math. Biol. 29 (1991) 629–651.
[17] K.A. Tomlinson, P.J. Hunter, A.J. Pullan, A finite element method for an eikonal equation model of myocardial excitation wavefront propagation, SIAM J. Appl. Math. 63 (2002) 324–350.
[18] N. Rawlinson, M. Sambridge, The fast marching method: an effective tool for tomographic imaging and tracking multiple phases in complex layered media, Explor. Geophys. 36 (2005) 341–350.
[19] M. Perič, Flow simulation using control volumes of arbitrary polyhedral shape, ERCOFTAC Bull. 62 (2004) 25–29.
[20] J. Hahn, K. Mikula, P. Frolkovič, B. Basara, Inflow-based gradient finite volume method for a propagation in a normal direction in a polyhedron mesh, J. Sci. Comput. 72 (2017) 442–465.
[21] J.A. Sethian, A fast marching level set method for monotonically advancing fronts, Proc. Natl. Acad. Sci. 93 (1996) 1591–1595.
[22] T.J. Barth, J.A. Sethian, Numerical schemes for the Hamilton-Jacobi and level set equations on triangulated domains, J. Comput. Phys. 145 (1998) 1–40.
[23] R. Kimmel, J.A. Sethian, Computing geodesic paths on manifolds, Proc. Natl. Acad. Sci. 95 (1998) 8431–8435.
[24] H.-K. Zhao, Fast sweeping method for eikonal equations, Math. Comput. 74 (2005) 603–627.
[25] H.-K. Zhao, Parallel implementations of the fast sweep method, J. Comput. Math. 25 (2007) 421–429.
[26] J.-L. Qian, Y.-T. Zhang, H.-K. Zhao, Fast sweeping methods for eikonal equations on triangular meshes, SIAM J. Numer. Anal. 31 (2007) 83–107.
[27] Y.-H.R. Tsai, L.-T. Cheng, S. Osher, H.-K. Zhao, Fast sweeping algorithms for a class of Hamilton-Jacobi equations, SIAM J. Numer. Anal. 41 (2003) 673–694.
[28] S.-R. Hysing, S. Turek, The eikonal equation: numerical efficiency vs. algorithmic complexity on quadrilateral grids, in: Proceedings of ALGORITMY, 2005, pp. 22–31.
[29] P.A. Gremaud, C.M. Kuster, Computational study of fast methods for the eikonal equation, SIAM J. Sci. Comput. 27 (2006) 1803–1816.
[30] W.-K. Jeong, R.T. Whitaker, A fast iterative method for eikonal equations, SIAM J. Sci. Comput. 30 (2008) 2512–2534.
[31] Z. Fu, W.-K. Jeong, Y. Pan, R.M. Kirby, R.T. Whitaker, A fast iterative method for solving the eikonal equation on triangulated surfaces, SIAM J. Sci. Comput. 33 (2011) 2468–2488.
[32] Z. Fu, R.M. Kirby, R.T. Whitaker, A fast iterative method for solving the eikonal equation on tetrahedral domains, SIAM J. Sci. Comput. 35 (2013) C473–C494.

[2] See more details in AVL Advanced Simulation Technologies University Partnership Program: https://www.avl.com/documents/10138/3372587/AVL_UPP_Flyer.pdf.

[33] S.F. Potter, M.K. Cameron, R. Duraiswami, Numerical geometric acoustics: an eikonal-based approach for modeling sound propagation in 3D environments, J. Comput. Phys. 486 (2023) 112111.

[34] S.F. Potter, M.K. Cameron, Jet marching methods for solving the eikonal equation, SIAM J. Sci. Comput. 43 (6) (2021) A4121–A4146.

[35] M.G. Crandall, L.C. Evans, P.-L. Lions, Some properties of viscosity solutions of Hamilton-Jacobi equations, Trans. Am. Math. Soc. 282 (487–502) (1984).

[36] I. Capuzzo-Dolcetta, P.-L. Lions, Hamilton-Jacobi equations with state constraints, Trans. Am. Math. Soc. 318 (1990) 643–683.

[37] K. Deckelnick, C.M. Elliott, V. Styles, Numerical analysis of an inverse problem for the eikonal equation, Numer. Math. 119 (2011) 245–269.

[38] H.M. Soner, Optimal control with state-space constraint. II, SIAM J. Control Optim. 24 (1986) 1110–1122.

[39] M. Falcone, C. Truini, A level-set algorithm for front propagation in the presence of obstacles, Rend. Mat. Appl. 29 (2009) 29–50.

[40] J. Hahn, K. Mikula, P. Frolkovič, B. Basara, Finite volume method with the Soner boundary condition for computing the signed distance function on polyhedral meshes, Int. J. Numer. Methods Eng. 123 (2022) 1057–1077.

[41] P.G. Tucker, C.L. Rumsey, P.R. Spalart, R.E. Bartels, R.T. Biedron, Computations of wall distances based on differential equations, AIAA J. 43 (2005) 539–549.

[42] P.G. Tucker, Hybrid Hamilton-Jacobi-Poisson wall distance function model, Comput. Fluids 44 (2011) 130–142.

[43] A.G. Belyaev, P.-A. Fayolle, On variational and PDE-based distance function approximations, Comput. Graph. Forum 34 (8) (2015) 104–118.

[44] A. Caboussat, R. Glowinski, T.-W. Pan, On the numerical solution of some eikonal equations: an elliptic solver approach, Chin. Ann. Math., Ser. B 36 (2015) 689–702.

[45] A.G. Belyaev, P.-A. Fayolle, A variational method for accurate distance function estimation, in: Numerical Geometry, Grid Generation and Scientific Computing, Springer International Publishing, 2019, pp. 175–181.

[46] H. Ennaji, N. Igbida, V.T. Nguyen, Augmented Lagrangian methods for degenerate Hamilton-Jacobi equations, Calc. Var. Partial Differ. Equ. 60 (2021) 238.

[47] P. Frolkovič, K. Mikula, J. Hahn, D. Martin, B. Basara, Flux balanced approximation with least-squares gradient for diffusion equation on polyhedral mesh, in: Discrete & Continuous Dynamical Systems - S, 2020.

[48] P.-A. Fayolle, A.G. Belyaev, An ADMM-based scheme for distance function approximation, Numer. Algorithms 84 (2020) 983–996.

[49] J. Hahn, K. Mikula, P. Frolkovič, B. Basara, Semi-implicit level set method with inflow-based gradient in a polyhedron mesh, in: C. Cancès, P. Omnes (Eds.), Finite Volumes for Complex Applications VIII - Hyperbolic, Elliptic and Parabolic Problems, Springer International Publishing, 2017, pp. 81–89.

[50] J. Hahn, K. Mikula, P. Frolkovič, M. Medl'a, B. Basara, Iterative inflow-implicit outflow-explicit finite volume scheme for level-set equations on polyhedron meshes, Comput. Math. Appl. 77 (2019) 1639–1654.

[51] K. Böhmer, P.W. Hemker, H.J. Stetter, The defect correction approach, in: Defect Correction Methods, Springer, 1984, pp. 1–32.

[52] P. Strachota, M. Beneš, Design and verification of the MPFA scheme for three-dimensional phase field model of dendritic crystal growth, in: Numerical Mathematics and Advanced Applications 2011, Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 459–467.

[53] K. Crane, C. Weischedel, M. Wardetzky, Geodesics in heat: a new approach to computing distance based on heat flow, ACM Trans. Graph. 32 (2013) 152.

[54] A.G. Churbanov, P.N. Vabishchevich, Numerical solution of boundary value problems for the eikonal equation in an anisotropic medium, J. Comput. Appl. Math. 362 (2019) 55–67.

[55] S.R.S. Varadhan, On the behavior of the fundamental solution of the heat equation with variable coefficients, Commun. Pure Appl. Math. 20 (1967) 431–455.

[56] P. Areias, N. Sukumar, J. Ambrósio, Continuous gap contact formulation based on the screened Poisson equation, Comput. Mech. (2023), https://doi.org/10.1007/s00466-023-02309-8.

[57] L.C. Evans, Partial Differential Equations, American Mathematical Society, Providence, R.I., 1998.

[58] K.S. Gurumoorthy, A. Rangarajan, A Schrödinger equation for the fast computation of approximate Euclidean distance functions, in: Scale Space and Variational Methods in Computer Vision, SSVM 2009, in: Lecture Notes in Computer Science, vol. 5567, Springer, Berlin, Heidelberg, 2009, pp. 100–111.

[59] P.G. Tucker, Assessment of geometric multilevel convergence robustness and a wall distance method for flows with multiple internal boundaries, Appl. Math. Model. 22 (1998) 293–311.

[60] G. Aubert, J.-F. Aujol, Poisson skeleton revisited: a new mathematical perspective, J. Math. Imaging Vis. 48 (2014) 149–159.

[61] K. Crane, C. Weischedel, M. Wardetzky, The heat method for distance computation, Commun. ACM 60 (2017) 90–99.