Massively parallel computation of globally optimal shortest paths with curvature penalization

Jean-Marie Mirebeau, Lionel Gayraud, Rémi Barrère, Da Chen, François Desquilbet

Massively parallel computation of globally optimal shortest paths with curvature penalization

Jean-Marie Mirebeau*, Lionel Gayraud†, Remi Barrere†, Da Chen‡, François Desquilbet§

October 4, 2022

Abstract

We address the computation of paths globally minimizing an energy involving their curvature, with given endpoints and tangents at these endpoints, according to models known as the Reeds-Shepp car (reversible and forward variants), the Euler-Mumford elasticae, and the Dubins car. For that purpose, we numerically solve degenerate variants of the eikonal equation, on a three dimensional domain, in a massively parallel manner on a graphical processing unit. Due to the high anisotropy and non-linearity of the addressed PDE, the discretization stencil is rather wide, has numerous elements, and is costly to generate, which leads to subtle compromises between computational cost, memory usage, and cache coherency. Accelerations by a factor 30 to 120 are obtained w.r.t a sequential implementation. The efficiency and the robustness of the method is illustrated in various contexts, ranging from motion planning to vessel segmentation and radar configuration.

Keywords: Eikonal equation, Curvature penalization, GPU acceleration

1 Introduction

The eikonal Partial Differential Equation (PDE) characterizes the minimal travel time of an omni-directional vehicle, from a fixed source point to an arbitrary target point, and allows to backtrack the corresponding globally optimal shortest path. The numerical solution of the eikonal PDE is at the foundation of numerous applications ranging from path planning to image processing or seismic tomography [Set99]. Real vehicles however are usually not omni-directional, but are subject to maneuverability constraints: cars cannot perform side motions, planes cannot stop, etc. In this paper we focus on the Reeds-Shepp, Euler-Mumford and Dubins vehicle models, which account for these constraints by increasing the cost of highly curved path sections, or even forbidding them. The variants of the eikonal PDE corresponding to these models are *non-holonomic* (a degenerate form of anisotropy) and are posed on the three dimensional state space $\mathbb{R}^2 \times \mathbb{S}^1$, which makes their numerical solution challenging. A dedicated variant of the fast marching method is presented in [Mir18, MP19], and together with earlier prototypes it has found applications in medical image segmentation [CMC16, CMC17, DMMP18] as well as the configuration of surveillance systems [MD17, DDBM19]. However, a weakness of the fast marching algorithm is its sequential nature: the points of the discretized domain are *accepted* one by one in a specific order, namely by ascending values of the front arrival times, which imposes the use of a single CPU thread managing a priority queue.

*Centre Borelli, ENS Paris-Saclay, CNRS, University Paris-Saclay, 91190, Gif-sur-Yvette, France
†Thales Research & Technology, Campus Polytechnique, 91767 Palaiseau
‡Qilu University of Technology (Shandong Acad. of Sciences), Shandong Artificial Intelligence Institute, China
§Univ. Grenoble-Alpes, LJK, F-38000, Grenoble, France

In this paper, we present a massively parallel solver of the non-holonomic eikonal PDEs associated with the Reeds-Shepp, Euler-Mumford and Dubins models of curvature penalized shortest paths. We use the same finite difference discretization as [Mir18, MP19], on a Cartesian discretization grid, but solve the resulting coupled system of equations using an iterative method implemented on a massively parallel computational architecture, namely a Graphics Processing Unit (GPU), following [WDB$^+$08, JW08, FKW13, GHZ18]. Our numerical schemes involve finite difference offsets which are often numerous (30 for Euler-Mumford), rather wide (up to 7 pixels), and whose construction requires non-trivial techniques from lattice geometry [Mir18]. This is in sharp contrast with the standard isotropic eikonal equation addressed by existing GPU solvers, which only requires few and small finite difference offsets when it is discretized on Cartesian grids [WDB$^+$08, JW08], and depends on unrelated geometric data when the domain is an unstructured mesh [FKW13, GHZ18]. Due to these differences, the compromises needed to achieve optimal efficiency - a delicate balance between the cost of computations and of memory accesses - strongly differ between previous works and ours, and even between the different models considered in this paper.

Our study provides the opportunity to inspect these compromises as the stencil of the finite difference scheme grows in width, number of elements and complexity, from 2 offsets of width 1 pixel (isotropic model in 2D), to 30 offsets of width up to 7 pixels (elastica model). Alternative finite difference discretizations may benefit from shorter stencils, but fail structural properties such as causality, leading to different compromises whose investigation is an opportunity for future research. Specifically, our observations regarding the models with *wider* finite difference stencils are the following: (i) They work best, somewhat counter intuitively, with a more finely grained parallelization, in our case obtained with smaller tiles and a smaller number of fixed point iterations within them, see §2.1 and Table 1. (ii) Precomputing and storing the stencil weights and offsets offers a significant speedup, up to 40% in our case, but the memory cost is prohibitive unless one can take advantage of symmetries in the equation to share this data between grid points, see §2.2. (iii) The scheme update operation involves a sort of the solution values fetched at the neighbors defined by the stencil, whose cost becomes dominant in the wide stencil case unless implemented in a GPU friendly manner, see §2.3. We expect our findings to transfer to other *wide stencil finite difference methods*, a class of numerical schemes commonly used to address Hamilton-Jacobi-Bellman PDEs arising in various applications, including deterministic (as here) and stochastic optimal control, optimal transport and optics design via the Monge-Ampere equation [Obe08], etc. Eventually, our GPU accelerated eikonal solver is 30× to 120× faster than the CPU fast marching method from [Mir18], see Table 2. In the numerical experiments §3, which include applications to medical image segmentation, boat routing and radar configuration, computations times on typical problem instances are often reduced from 30 seconds to less than one, enabling convenient user interaction.

Outline. We describe §1.1 the curvature penalized path optimization problems addressed, and §1.2 the eikonal equation formalism and the corresponding finite difference scheme. Our numerical solver is presented § 2, distinguishing routines acting at the grid scale § 2.1, the tile scale § 2.2, and the pixel scale § 2.3, see also Algorithms 1 to 3. Numerical experiments § 3 illustrate the method's efficiency in various applications, corresponding to the best case scenario § 3.1 or to various difficulties such as obstacles §3.2, strongly inhomogeneous cost functions §3.3, asymmetric perturbations of the curvature penalization §3.4, and optimization problems §3.5.

Remark 1.1 (Absence of conflict of interest, data availability, intellectual property)**.** *All authors declare that they have no conflicts of interest. The numerical methods presented in this paper are available as a public and open source library*[1]*, licensed under the Apache License 2.0, and whose development is led by J.-M. Mirebeau. The specific inputs needed to reproduce the numerical experiments are available on demand to the first author. In a preliminary and unpublished work, accelerations of the same order were obtained with an earlier independent GPU implementation of the HFM [MP19] method (limited to the Dubins model) developed by L. Gayraud with the support of R. Barrere, and in informal collaboration with J.-M. Mirebeau. The two libraries are written in*

[1] `www.github.com/Mirebeau/AdaptiveGridDiscretizations`

different languages (Python/CUDA versus C++/OpenCL), do not share a single line of code, use different implementation tricks, and offer distinct functionality.

Notations. We use square brackets to denote real intervals, such as $]a,b[$, $]a,b]$, $[a,b[$ and $[a,b] \subset \mathbb{R}$, whose bounds a and b are either contained or excluded following the usual convention. We denote $\mathbb{e}_\theta := (\cos\theta, \sin\theta)$ for all $\theta \in \mathbb{R}$.

1.1 Curvature penalized path models

Throughout this paper we fix a bounded and closed domain $\Omega \subset \mathbb{R}^2$, and a continuous and positive cost function $\rho : \overline{\Omega} \times \mathbb{S}^1 \to]0, \infty[$, where $\mathbb{S}^1 := [0, 2\pi[$ with periodic boundary conditions. The general objective of this paper is to compute paths $(\boldsymbol{x}, \boldsymbol{\theta}) : [0, L] \to \overline{\Omega} \times \mathbb{S}^1$ in the position-orientation state space, which globally minimize the energy

$$\mathcal{E}(\boldsymbol{x}, \boldsymbol{\theta}) := \int_0^L \rho(\boldsymbol{x}, \boldsymbol{\theta})\, \mathcal{C}(\dot{\boldsymbol{\theta}})\, \mathrm{d}l, \qquad \text{subject to } \dot{\boldsymbol{x}} = \mathbb{e}_{\boldsymbol{\theta}}, \tag{1}$$

where we denoted $\mathbb{e}_\theta := (\cos\theta, \sin\theta)$ and $\dot{\boldsymbol{\theta}} := \frac{\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}l}$ and $\dot{\boldsymbol{x}} := \frac{\mathrm{d}\boldsymbol{x}}{\mathrm{d}l}$. An additional constraint to (1) is that the initial and final configurations $\boldsymbol{x}(0)$, $\boldsymbol{\theta}(0)$ and $\boldsymbol{x}(L)$, $\boldsymbol{\theta}(L)$ are imposed, in other words the endpoints of the physical path and the tangents at these endpoints. The path is parametrized by Euclidean length in the physical space Ω, and the total length L is a free optimization parameter. The constraint (1, right) requires that the path physical velocity $\dot{\boldsymbol{x}}(l)$ matches the direction defined by the angular coordinate $\mathbb{e}_{\boldsymbol{\theta}(l)} := (\cos\boldsymbol{\theta}(l), \sin\boldsymbol{\theta}(l))$, for all $l \in [0, L]$. This constraint is said *non-holonomic* because it binds together the first order derivatives of the path $(\dot{\boldsymbol{x}}(l), \dot{\boldsymbol{\theta}}(l))$ for each $l \in [0, L]$.

The choice of curvature penalty function $\mathcal{C}(\kappa)$, where $\kappa := \dot{\boldsymbol{\theta}}$ is the derivative of the path direction in (1), is limited to three possibilities in our approach, in contrast with the state dependent penalty ρ which is essentially arbitrary. The considered curvature penalties are defined by the following expressions, which correspond to the Reeds-Shepp, Euler-Mumford, and Dubins models respectively: we define $\mathcal{C}(\kappa)$, for all $\kappa \in \mathbb{R}$, as either

$$\sqrt{1+\kappa^2}, \qquad 1+\kappa^2, \qquad 1+\infty_{|\kappa|>1}, \tag{2}$$

where ∞_{cond} stands for $+\infty$ where *cond* holds, and 0 elsewhere. The Reeds-Shepp model penalizes curvature in a roughly linear manner; because this is a rather weak regularization, smooth minimizers of (2) may not exist [BCR10], and a relaxed formulation allowing for singularities is required, introducing either cusps or in-place rotations, and leading to two variants referred to as the Reeds-Shepp reversible and the Reeds-Shepp forward models, see Remark 1.3. The quadratic curvature penalty of the Euler-Mumford model corresponds to the energy of an elastic bar, hence minimal paths follow the rest position of those objects. Finally the Dubins model forbids any path section whose curvature exceeds that of the unit disk, by assigning to it the cost $+\infty$. Minimal paths for these models are qualitatively distinct, as illustrated on Figure 1. The curvature penalty may also be scaled and shifted, so as to control its strength and symmetry, see Remark 1.2 and §3.4.

In the following, we fix a seed point $(x_*, \theta_*) \in \Omega \times \mathbb{S}^1$ in the state space, and denote by $u(x, \theta)$ the minimal cost of a path from this seed to an arbitrary target $(x, \theta) \in \Omega \times \mathbb{S}^1$:

$$\begin{aligned} u(x,\theta) := \inf\{\mathcal{E}(\boldsymbol{x},\boldsymbol{\theta});\, L \geq 0,\, (\boldsymbol{x},\boldsymbol{\theta}) : [0,L] \to \Omega \times \mathbb{S}^1,\, \dot{\boldsymbol{x}} = \mathbb{e}_{\boldsymbol{\theta}},\\ \boldsymbol{x}(0) = x_*,\ \boldsymbol{\theta}(0) = \theta_*,\ \boldsymbol{x}(L) = x,\ \boldsymbol{\theta}(L) = \theta\}. \end{aligned} \tag{3}$$

For the Reeds-Shepp models, the path cost is modified as in Remark 1.3. Once the map $u : \Omega \times \mathbb{S}^1 \to \mathbb{R}$ is numerically computed, as described in §1.2, a standard backtracking technique [Mir18] allows to extract the path $(\boldsymbol{x}, \boldsymbol{\theta}) : [0, L] \to \overline{\Omega} \times \mathbb{S}^1$ globally minimizing (1), from the seed state (x_*, θ_*) to any given target $(x^*, \theta^*) \in \Omega \times \mathbb{S}^1$.

Remark 1.2 (Scaling and shifting the curvature penalty)**.** *The curvature penalty $\mathcal{C}(\dot{\boldsymbol{\theta}})$ appearing in our path models* (1) *can be generalized into $\mathcal{C}(\xi(\dot{\boldsymbol{\theta}} - \varphi))$. The parameter $\xi > 0$ dictates the*

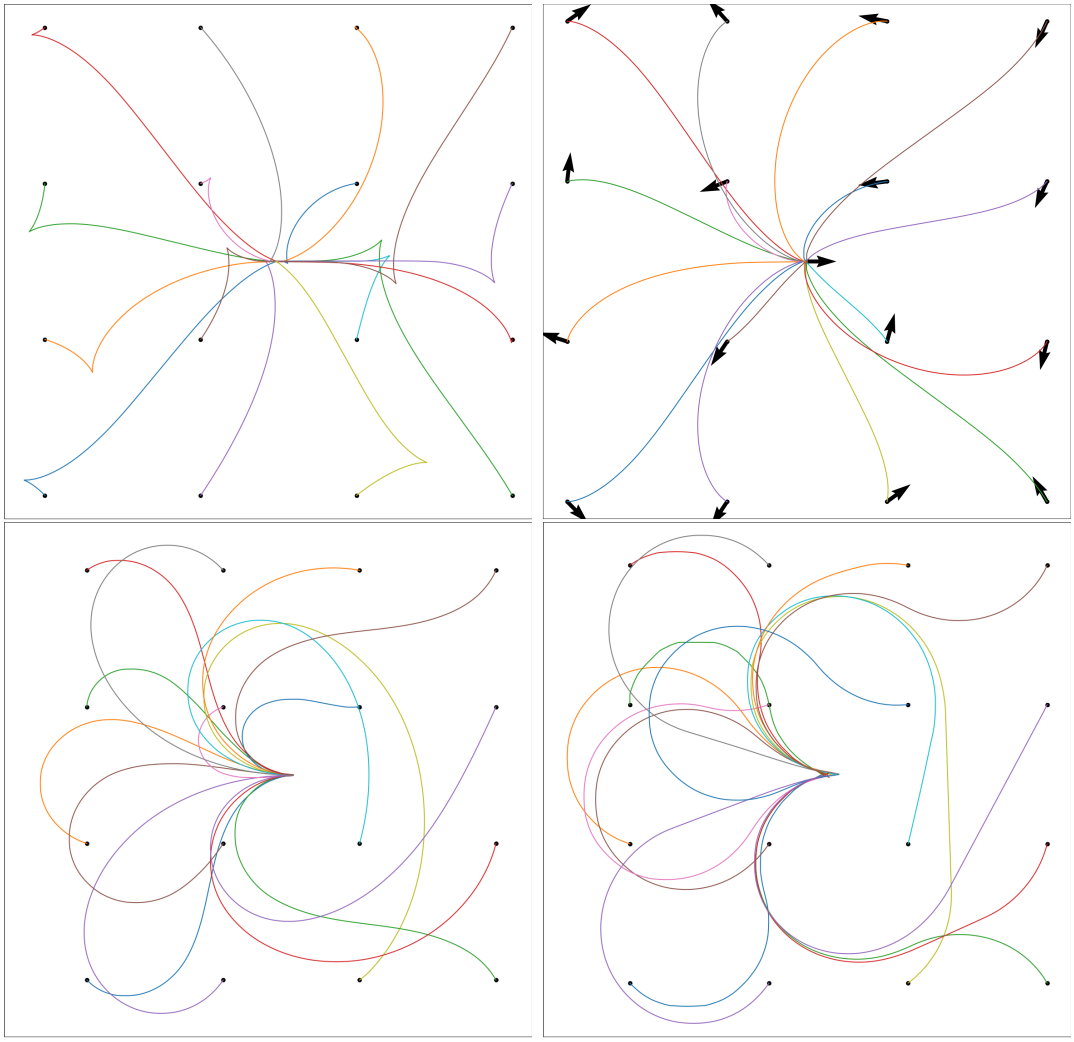

Figure 1: Planar projections of minimal geodesics for the Reeds-Shepp, Reeds-Shepp forward, Elastica and Dubins models (left to right). Seed point $(0, 0)$ with horizontal tangent, regularly spaced tip point with random tangent (but identical for all models).

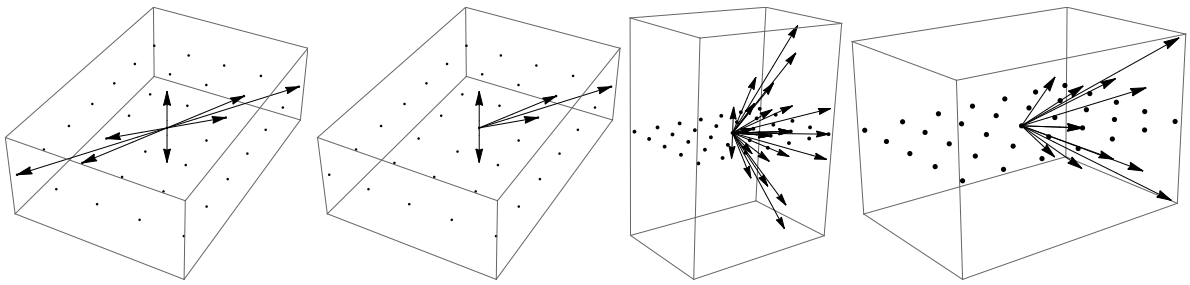

Figure 2: Discretization stencils used for the Reeds-Shepp reversible, Reeds-Shepp forward, Euler-Mumford, and Dubins models. They respectively involve $N_{\text{neigh}} = 8$, 5, 30 and 12 neighbors, a number which strongly contributes to the numerical method's cost. Note the sparseness and the anisotropy of the stencils. Model parameters: $\theta = \pi/3$, $\xi = 0.2$, $\varepsilon = 0.1$.

intensity of curvature penalization, whereas $\varphi \in \mathbb{R}$ can introduce asymmetric penalty. Optionally, $\xi = \xi(x,\theta)$ and $\varphi = \varphi(x,\theta)$ may depend on the current state $(x,\theta) \in \Omega \times \mathbb{S}^1$.

Remark 1.3 (Well posed formulation of the Reeds-Shepp model)**.** *The minimization of the energy* (1) *associated with the Reeds-Shepp cost $\mathcal{C}(\kappa) := \sqrt{1+\kappa^2}$ is* not *a well posed optimization problem in general [BCR10, DBRS13], since minimizing sequences may develop non-differentiable singularities in the limit, see Figure 1. To obtain a well posed problem, following [DMMP18], we reject the convenient planar Euclidean arclength parametrization $l \in [0, L]$ used in* (1)*, in favor of an arbitrary time parametrization $t \in [0,1]$, thus allowing for times where the spatial velocity vanishes $\dot{\boldsymbol{x}}(t) = 0$ but the angular velocity does not $\dot{\boldsymbol{\theta}}(t) \neq 0$.*

The classical Reeds-Shepp sub-Riemannian model asks for a Lipschitz path $(\boldsymbol{x}, \boldsymbol{\theta}) : [0,1] \to \Omega \times \mathbb{S}^1$, with prescribed initial and final configurations $\boldsymbol{x}(0)$, $\boldsymbol{\theta}(0)$ and $\boldsymbol{x}(1)$, $\boldsymbol{\theta}(1)$, and minimizing

$$\int_0^1 \rho(\boldsymbol{x},\boldsymbol{\theta}) \sqrt{\dot{\boldsymbol{x}}^2 + \dot{\boldsymbol{\theta}}^2} \mathrm{d}t \qquad \text{subject to } \langle \dot{\boldsymbol{x}}, \mathbb{e}_{\boldsymbol{\theta}}^{\perp} \rangle = 0. \tag{4}$$

The constraint (4, right) ensures that $\dot{\boldsymbol{x}}(t) = \lambda(t) \mathbb{e}_{\boldsymbol{\theta}}(t)$ for each $t \in [0,1]$, with a proportionality constant $\lambda(t) \in \mathbb{R}$. In the special case where $\lambda(t) > 0$ for all $t \in [0,1]$, one recovers (1) *using a reparametrization of the path by the planar Euclidean arclength.*

The formulation (4) *allows the proportionality constant $\lambda(t)$ to be* positive or negative*, and for this reason we refer to* (4) *as the Reeds-Shepp* reversible *model. A cusp is observed when λ changes sign.*

One may also introduce in (4) *the additional constraint $\langle \dot{\boldsymbol{x}}, \mathbb{e}_{\boldsymbol{\theta}} \rangle \geq 0$, thus ensuring that the proportionality constant $\lambda(t) \geq 0$ is non-negative, and leading to the Reeds-Shepp* forward *model [DMMP18]. A rotation in-place is observed when λ vanishes over some time interval, which often happens at the start and at the end of the path.*

1.2 Non-holonomic eikonal equations, and their discretization

The minimal travel cost (3), from a given source point to an arbitrary target, is the value function of a deterministic optimal control problem. As such, it obeys a first order static non-linear PDE, a variant of the eikonal equation, of the generic form

$$\mathcal{F}u(x,\theta) = \rho(x,\theta) \qquad \text{where} \quad \mathcal{F}u(x,\theta) = \mathfrak{F}(x,\theta,\, \nabla_x u(x,\theta),\, \partial_\theta u(x,\theta)),$$

where $\nabla_x u(x,\theta) \in \mathbb{R}^2$ and $\partial_\theta u(x,\theta) \in \mathbb{R}$ denote the partial derivatives of the unknown $u : \Omega \times \mathbb{S}^1 \to \mathbb{R}$ w.r.t. the physical position x and the angular coordinate θ. This PDE holds in $\Omega \times \mathbb{S}^1 \setminus \{(x_*, \theta_*)\}$, while the constraint $u(x_*, \theta_*) = 0$ is imposed at the seed point (x_*, θ_*), and outflow boundary conditions are applied on $\partial\Omega$. The detailed arguments and adequate concepts of optimal control, Hamilton-Jacobi-Bellman equations, and discontinuous viscosity solutions, are non-trivial and unrelated to the object of this paper (which is GPU acceleration), hence we simply refer the interested reader to [BCD08, Mir18]. For comparison, the standard isotropic eikonal equation [RT92, Set99] on $\mathbb{R}^d$, which corresponds to an omni-directional vehicle not subject to maneuverability constraints or curvature penalization, is defined by the operator $\mathcal{F}u = \|\nabla u\|$.

The considered variants of the eikonal PDE involve the following non-linear and anisotropic operators $\mathcal{F}u(x,\theta)$, see [Mir18].

$$\sqrt{\langle \nabla_x u, \mathbb{e}_\theta \rangle^2 + |\partial_\theta u|^2}, \qquad \sqrt{\max\{0, \langle \nabla_x u, \mathbb{e}_\theta \rangle\}^2 + |\partial_\theta u|^2}, \tag{5}$$

$$\frac{1}{2}(\langle \nabla_x u, \mathbb{e}_\theta \rangle + \sqrt{\langle \nabla_x u, \mathbb{e}_\theta \rangle^2 + |\partial_\theta u|^2}), \qquad \langle \nabla_x u, \mathbb{e}_\theta \rangle + |\partial_\theta u|. \tag{6}$$

They respectively correspond to the Reeds-Shepp reversible (5, left), Reeds-Shepp forward (5, right), Euler-Mumford (6, left) and Dubins (6, right) models.

We rely on a finite differences discretization Fu of the operator $\mathcal{F}u$, on the Cartesian grid

$$X_h := (\Omega \times \mathbb{S}^1) \cap h\mathbb{Z}^3, \tag{7}$$

where the physical domain is usually rectangular $\Omega = [a, b] \times [c, d]$ (or padded as such), and where the grid scale $h > 0$ is such that $2\pi/h \in \mathbb{N}$ so that the sampling of $\mathbb{S}^1 := [0, 2\pi[$ is compatible with the periodic boundary conditions. By convention, the value function u is extended by $+\infty$ outside Ω, thus implementing the desired outflow boundary conditions on $\partial\Omega$. For any discretization point $p = (x, \theta) \in X_h$, the finite differences operator $Fu(p)$ is defined as the square root of the following expression [MP19]

$$\max_{1\leq k\leq K} \Big(\sum_{1\leq i\leq I} \alpha_{ik} \max\big\{0, \frac{u(p) - u(p + he_{ik})}{h}\big\}^2 + \sum_{1\leq j\leq J} \beta_{jk} \max_{\sigma=\pm 1} \big\{0, \frac{u(p) - u(p + \sigma h f_{jk})}{h}\big\}^2 \Big), \quad (8)$$

where I, J, K are fixed integers, $\alpha_{ik}, \beta_{jk} \geq 0$ are non-negative weights, and $e_{ik}, f_{jk} \in \mathbb{Z}^3$ are finite difference offsets, for all $1 \leq i \leq I$, $1 \leq j \leq J$, $1 \leq k \leq K$. The weights and offsets may depend on the current point p. Before turning to the variants (6) and (5), let us mention that the standard discretization [RT92] of the isotropic eikonal equation ($\mathcal{F}u = \|\nabla u\|$) fits within this framework, with meta-parameters $J = d$ (and $I = 0$, $K = 1$), choosing unit weights $w_{j1} = 1$, $1 \leq j \leq d$, and letting $(f_{j1})_{i=1}^d$ be the canonical basis of $\mathbb{R}^d$. Riemannian eikonal PDEs can also be addressed in this framework, with $J = d(d + 1)/2$ (and $I = 0$, $K = 1$) and using weights and offsets defined by an appropriate decomposition of the inverse metric tensor, see [Mir19, MP19]. The anisotropy of the Riemannian metric is not bounded a-priori, but strong anisotropy leads to large stencils : specifically $\|f_{j1}\| \leq C\sqrt{\|M\|\|M^{-1}\|}$ for all $1 \leq j \leq J$ in dimension $d \leq 3$, where M denotes the Riemannian metric tensor, see [Mir19, Proposition 1.1]. Excessively large stencils in turn lead to longer execution time due to cache misses, slower convergence of the iterative method, and less precise boundary conditions.

In the curvature penalized case, the weights and offsets in (8) implicitly depend on the base point $p = (x, \theta)$, at least through the angular coordinate θ, in view of the continuous PDEs (5) and (6). They may also depend on the physical position x if the strength or the symmetry of the curvature penalty varies from point to point, see Remark 1.2. We refer to [Mir18, MP19] for details on the construction of the weights and offsets, and simply report here the meta-parameters for the Reeds-Shepp reversible ($I = 0$, $J = 4$, $K = 1$), Reeds-Shepp forward ($I = 3$, $J = 1$, $K = 1$), Euler-Mumford ($I = 30$, $J = 0$, $K = 1$), and Dubins ($I = 6$, $J = 0$, $K = 2$) models, see Figure 2. The construction also involves a relaxation parameter $\varepsilon > 0$ for the non-holonomic constraint (1, right), which dictates the size of the stencils : specifically $\|e_{ik}\|, \|f_{jk}\| \leq C/\varepsilon$ for all i, j, k, see [Mir18, Proposition 1.1]. Convergence is established in the limit where $\varepsilon \to 0$ and $h/\varepsilon \to 0$, see [Mir18], yet in practice the fixed choice $\varepsilon = 0.1$ appears suitable for the considered applications.

A fundamental property of discretization schemes of the form (8) is that they can be solved in a single pass over the domain, using a generalization of the fast-marching algorithm [Mir18, MP19, Mir19], thanks to a property known as *causality*. This is highly desirable when implementing CPU solver, but anecdotical for a GPU eikonal solver whose massive parallelism forbids taking advantage of this property. See Appendix A for more discussion of the properties of these schemes and of their relevance to GPU implementations. Nevertheless, those schemes are robust and well tested. Alternative approaches offering different compromises and possibly more suited to GPUs will be considered in future works.

2 Implementation

We describe the implementation of our massively parallel solver of generalized eikonal PDEs, assumed to be discretized in the form (8). The bulk of the method is split in three procedures, Algorithms 1 to 3, discussed in detail in the corresponding sections.

For simplicity, Algorithms 2 and 3 are written in the special case where the meta parameters of the discretization (8) are $J = 0$ and $K = 1$, whereas I is arbitrary. The case of arbitrary J and K is discussed in §2.3. The assignment of a value *val* to a scalar (resp. array) variable *var* is denoted $var \leftarrow val$ (resp. $var \Leftarrow val$).

Algorithm 1 Parallel iterative solver (Python)

Variables:

$u : X_h \to [0, \infty]$ (The problem unknown)

$active, next : B_h \to \{0, 1\}$. (Blocks marked for current and next update)

Initialization:

$u \Leftarrow \infty$; $active, next \Leftarrow 0$.

$u[p_*] \leftarrow 0$; $active[b_*] \leftarrow 1$. (Set seed point value, and mark its block for update)

While an *active* block remains:

  For all *active* blocks b in parallel: (CUDA kernel lauch)

    For all $p \in X_h^b$ in parallel: (Block of threads)

      BlockUpdate$(u, next, b, p)$

  $active \Leftarrow next$; $next \Leftarrow 0$.

Algorithm 2 BlockUpdate$(u, next, b, p)$, where $p \in X_h^b$ (CUDA)

Global variables: $u : X_h \to [0, \infty]$, $next : B_h \to \{0, 1\}$, $\rho : X_h \to \mathbb{R}$ (the r.h.s).

Block shared variable: $u_b : X_h^b \to [0, \infty]$.

Thread variables: $\alpha_i \geq 0$, $e_i \in \mathbb{Z}^d$, $u_i \in \mathbb{R}$, for all $1 \leq i \leq I$.

$u_b(p) \leftarrow u(p)$; __syncthreads() (Load main memory values into shared array)

Load or compute the stencil weights $(\alpha_i)_{i=1}^I$ and offsets $(e_i)_{i=1}^I$.

$u_i \leftarrow u(p + he_i)$, for all $1 \leq i \leq I$ such that $p + he_i \notin X_h^b$. (Load the neighbor values)

For r from 1 to R:

  $u_i \leftarrow u_b(p + he_i)$, for all $1 \leq i \leq I$ such that $p + he_i \in X_h^b$. (Load shared values)

  $u_b(p) \leftarrow \Lambda(\rho(p), \alpha_i, u_i, 1 \leq i \leq I)$ (Update $u_b(p)$, unless p is the seed point)

  __syncthreads() (Sync shared values)

$u(p) \leftarrow u_b(p)$ (Export shared array values to main memory)

If appropriate, $next[b] \leftarrow 1$ and/or $next[b'] \leftarrow 1$ for each neighbor block b' of b. (Thread 0 only)

Algorithm 3 Local update operator $\Lambda(\rho, \alpha_i, u_i, 1 \leq i \leq I)$ (C++)

Variables $a \leftarrow 0$, $b \leftarrow 0$, $c \leftarrow -h^2\rho^2$, $\lambda \leftarrow \infty$.

Sort the indices, so that $u_{i_1} \leq \cdots \leq u_{i_I}$.

For r from 1 to I:

  If $\lambda \leq u_{i_r}$ **then** break.

  $a \leftarrow a + \alpha_{i_r}$; $b \leftarrow b + \alpha_{i_r} u_{i_r}$; $c \leftarrow c + \alpha_{i_r} u_{i_r}^2$

  $\lambda \leftarrow (b + \sqrt{b^2 - ac})/a$

return λ

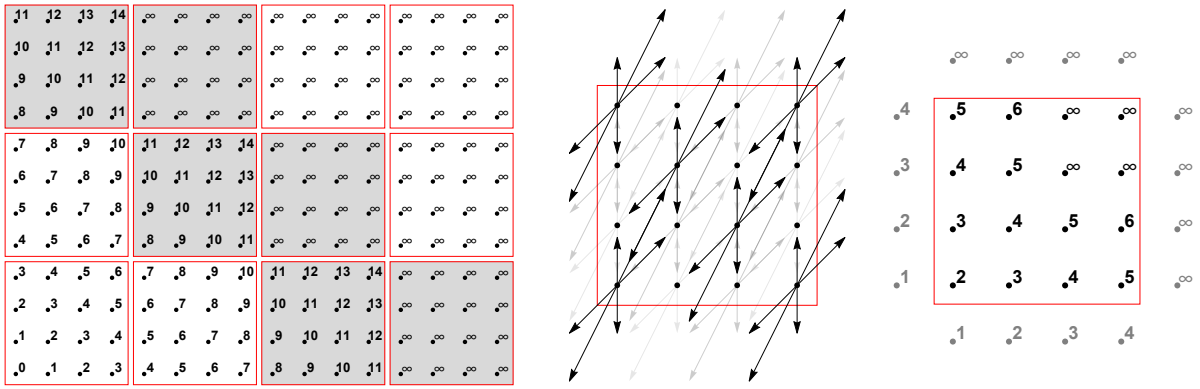

Figure 3: *Left:* Decomposition of the Cartesian grid X_h into tiles X_h^b, with block index $b \in B_h$. Grayed blocks are tagged *active*. *Center:* Updating a block $b \in B_h$ requires loading the unknown values $u : X_h \to \mathbb{R}$, both within X_h^b and at some neighbor points. *Right:* The solution values within a block are updated several times. (Here, after only two updates, the block values have not yet stabilized).

2.1 Parallel iterative solver

Massively parallel architectures divide computational tasks into *threads* which, in the case of graphics processing units, are grouped into *blocks* (indexed by b) following a common sequence of instructions, and able to take advantage of shared data, see Remark 2.1. Following [WDB+08, JW08, GHZ18], the main loop of our iterative eikonal equation solver is designed to take advantage of this computational architecture, see Algorithm 1. It is written in the Python programming language, which is also used for the pre- and post-processing tasks, and launches Algorithm 2 as a CUDA kernel via the cupy[2] library.

The discretization domain X_h, which is a three dimensional Cartesian grid (7), is split into rectangular tiles X_h^b, indexed by $b \in B_h$, see Figure (3, left). The update of a tile X_h^b is handled by a block of threads, and the tile should therefore contain no less than 32 points in view of Remark 2.1. The best shape of the tiles X_h^b was found to be $4 \times 4 \times 4$ for the Reeds-Shepp models (forward and reversible), and $4 \times 4 \times 2$ for the Euler-Mumford and Dubins models, see §2.2 and Table 1. One of the findings of our work is indeed that the schemes featuring wider stencils work best with smaller tile sizes, see also the discussion in the second paragraph of §2.2. Some padding is introduced if the dimensions of the tiles X_h^b do not divide those of the grid X_h.

A boolean table $\mathit{active} : B_h \to \{0, 1\}$ records all tiles tagged for update. Denote by $N_h := \#(B_h)$ the total number of tiles, and by $N_b := \#(X_h^b)$ the number of grid points in a tile, which is independent of $b \in B_h$, so that $\#(X_h) = N_h N_b$ by construction. Let also $N_{act} = \#\{b \in B_h;\ \mathit{active}[b]\}$ be the number of active tiles in a typical iteration of Algorithm 1. Since we are implementing a front propagation in a three dimensional domain, one generally expects that $N_{act} \approx N_h^{2/3}$ (in d-dimensions, $N_{act} \approx N_h^{1-1/d}$).

In each iteration of Algorithm 1, the *active* table is checked for emptiness, in which case the program terminates. More importantly, the indices of all non-zero entries of the *active* table are extracted, so as to update only the relevant blocks. The complexity $\mathcal{O}(N_h \ln N_h)$ of this operation is in practice negligible w.r.t. the cost of the block updates themselves $\mathcal{O}\big(N_{act} N_b R K (I + J)\big)$ where R is the number of inner loops in Algorithm 2 and I, J, K are the scheme parameters (8). A second boolean table $\mathit{next} : B_h \to \{0, 1\}$, is used to mark the blocks which are to be updated in the subsequent iteration.

A single array $u : X_h \to [0, \infty[$ holds the solution values. Indeed, the block update operator benefits from a monotony property, see Appendix A, which guarantees that the values of $(u_n)_{n \geq 0}$

[2] A NumPy-compatible array library accelerated by CUDA. `https://cupy.dev`

of the numerical solution *decrease* along the iterations of Algorithm 1, toward a limit u_∞. As a result, load/store data races in u between the threads are innocuous.

Remark 2.1 (SIMT architecture)**.** *A block of threads is under the hood handled by a GPU device in a Single Instruction Multiple Threads (SIMT) manner : the same instructions are applied on 32 threads of a same block (also called a warp) simultaneously. For this reason, the number of threads within a block should preferably be a multiple of the width of a warp. For the same reason, thread divergence (threads within a warp going along different execution paths, due to conditional branching statements, which is implemented in a sequential manner by "muting" the threads of the inactive branch) should be avoided for best efficiency.*

Our numerical solver reflects these properties through the choice of the tile size X_h^b, and in the choice of an Eulerian discretization scheme on a Cartesian grid (8) *whose solution by Algorithm 3 involves little branching and yields an even load between threads, as opposed to an unstructured mesh where these properties are by design less ensured [FKW13].*

2.2 Block update

The BlockUpdate procedure, presented in Algorithm 2, is the most complex part of our numerical method. It is executed in parallel by a block of threads, each handling a given point $p \in X_h^b$ of a tile of the computational grid, where the tile index $b \in B_h$ is fixed.

An array $u_b : X_h^b \to [0, \infty]$ *shared* between the threads of the block is initialized with the values of the unknown $u : X \to [0, \infty]$ at the same positions. Throughout the execution of the BlockUpdate procedure, the values of u_b are updated several times, and then finally they are exported back into the main array u. If the number R of updates of u_b is sufficiently large, then this procedure amounts to solving a local eikonal equation on X_h^b, with $u_{|X_h \setminus X_h^b}$ treated as boundary conditions. A similar approach is used in [WDB$^+$08, JW08, GHZ18]. We empirically observe that stencil schemes using a wide stencil work best with a small number R of iterations, and a small tile size, see Table 1. Our interpretation is the following: using several iterations is meant to propagate the front through the tile X_h^b and to stabilize the local solution u_b within the tile [JW08], but this objective looses relevance when the stencil is so wide that the scheme update at a point $x \in X_h^b$ involves fewer values of the local array u_b in X_h^b than of the global array u in $X_h \setminus X_h^b$, which is cached and frozen throughout the iterations in Algorithm 2. In addition, each of the R iterations has a higher cost when the stencil is wide and has numerous elements, see the description of Algorithm 3 in §2.3.

The finite difference scheme (8) used for curvature penalized fast marching is built using non-trivial tools from lattice geometry [Mir18], whose numerical cost cannot be ignored. Empirical tests show that precomputing the weights and offsets usually reduces overall computation time by 30% to 50%. If the scheme structure only depends on the angular coordinate θ of the point, then the precomputed stencils can be shared across all physical coordinates x and thus have a negligible memory usage, so that these precomputations are a pure benefit. On the other hand, if the scheme stencils depend on all coordinates (x, θ) of the current point, typically for a model whose curvature penalty function depends on the current point as discussed in Remark 1.2 and § 3.4, then the storage cost of the weights and offsets significantly exceeds the amount of problem data. (Stencils are defined by $N = K(I+J)$ scalars and offsets per grid point, see (8), where typically $4 \leq N \leq 30$. In comparison, the problem data u, ρ and optionally ξ, φ consists of 2 to 4 scalars per grid point, see Remark 1.2.) Stencil recomputation is preferred in these cases, in order to avoid crippling the ability of the numerical method to address large scale problems on memory limited GPUs.

The values of the unknown $u : X_h \to \mathbb{R}$ needed for the evaluation of the scheme (8) and lying outside X_h^b are loaded once and for all at the beginning of the BlockUpdate procedure Algorithm 2, and treated as fixed boundary conditions so as to minimize memory bandwidth usage. Contrary to what could be expected, such boundary values are an overwhelming majority in comparison with the values located within the tile X_h^b. For instance the three dimensional isotropic eikonal equation, using standard tiles of $64 = 4 \times 4 \times 4$ points, involves $96 = 6 \times 4 \times 4$ boundary values. Boundary values are even more numerous with the curvature penalization schemes, which involve many wide finite difference offsets, as illustrated on Figure (3, center).

Each thread of a block, associated to a discretization point $p \in X_h^b$ where $b \in B_h$ is the block index, goes through R iterations of a loop where the local unknown value $u_b(p)$ is updated via Algorithm 3, see §2.3. The threads are synchronized at each iteration of this loop, to ensure that the front propagates through the tile X_h^b. Since the values of $u_b : X_h^b \to [0, \infty]$ are decreasing along the iterations, by monotony of the scheme see Appendix A, no additional protection of u_b against data races between the threads of the block is required. The number R of iterations is discussed in §2.3.

Last but not least, the block b and its immediate neighbors b' need to be tagged for update in the next iteration of the eikonal solver Algorithm 1, if appropriate, via the boolean array *next* : $B_h \to \{0, 1\}$. This step is *not* fully described in Algorithm 2, and in particular the *neighbors* of a tile and the *appropriate* condition for marking them are not specified. Indeed, a variety of strategies can be plugged in here, and our numerical solver is not tied to any of them. Good results were obtained using Adaptive Gauss Siedel Iteration (AGSI) [BR06, GHZ18] and with the Fast Iterative Method (FIM) [JW08], while other variants were not tested [WDB$^+$08].

Remark 2.2 (Walls and thin obstacles)**.** *Our finite differences scheme involves rather wide stencils, see Figure 2, raising the following issue: the update of a point p may involve neighbor values $u(p + he_i)$ across a thin obstacle. In order to avoid propagating the front through the obstacles, if any are present, an additional* walls *array is introduced in Algorithm 2, as well as and intersection test between the segment $[p, p + he_i]$ and the obstacles. For computational efficiency, the array* walls $: X_h \to \{0, \cdots, 255\}$ *is not boolean, but* walls$[p]$ *instead encodes the Manhattan distance in pixels (capped at 255) from the current point p to the nearest obstacle. If $\|e_i\|_1 <$ walls$[p]$, then $[p, p + he_i]$ does not meet the obstacles, and the intersection test can be bypassed.*

2.3 Local update

This section is devoted to the local update operator presented in Algorithm 3. From the mathematical standpoint, one defines $\Lambda u(p)$ as the solution to the equation $Fu(p) = \rho(p)$ w.r.t. the variable $u(p)$, regarding all neighbor values as constants, see [Mir19, Appendix A]. This process of solving a discretized PDE at a single point p, and w.r.t. the corresponding unknown $u(p)$, is often referred to as a Gauss-Siedel update. We prove in this subsection that Algorithm 3 does compute the correct update value $\Lambda u(p)$, and comment on its numerical complexity and efficient implementation. These results generalize and abstract the update step of the standard fast marching method for isotropic eikonal equations [Set96], whose discretization is a special case of (8) as mentioned in §1.2.

For simplicity, and consistently with the presentation of Algorithm 3, we first assume a numerical scheme of the following form : denoting $a_+ := \max\{0, a\}$,

$$(Fu(p))^2 := h^{-2} \sum_{i=1}^{I} \alpha_i \big(u(p) - u(p + he_i)\big)_+^2, \tag{9}$$

in other words $J = 0$ and $K = 1$ in (8). Lemma 2.3 shows that the equation $Fu(p) = \rho(p)$ admits a unique solution w.r.t. the variable $u(p)$, where p is a fixed discretization point, and Proposition 2.4 validates Algorithm 3 for computing this root. They should be applied with the parameters $u_i := u(p + he_i)$ for all $1 \le i \le I$, and $\rho := \rho(p)$.

Lemma 2.3. *Consider values $u_1, \cdots, u_I \in \mathbb{R}$, weights $\alpha_1, \cdots, \alpha_I > 0$, and let $h\rho > 0$. Then there exists a unique solution $\lambda_* \in \mathbb{R}$ to the equation $f(\lambda) = 0$, where*

$$f(\lambda) := \sum_{1 \le i \le I} \alpha_i (\lambda - u_i)_+^2 - h^2\rho^2. \tag{10}$$

Proof. Assume w.l.o.g. that $u_1 \le \cdots \le u_I$. Observing that $f(\lambda) = -h^2\rho^2 < 0$ for all $\lambda \in]-\infty, u_1]$, and that $f(\lambda) \ge \alpha_1(\lambda - u_1)^2 - h^2\rho^2 \to \infty$ as $\lambda \to \infty$, we see that f admits a root $\lambda_* \in [u_1, \infty[$ by the intermediate value theorem. This root is unique since f is increasing on $[u_1, \infty[$, which concludes the proof. □

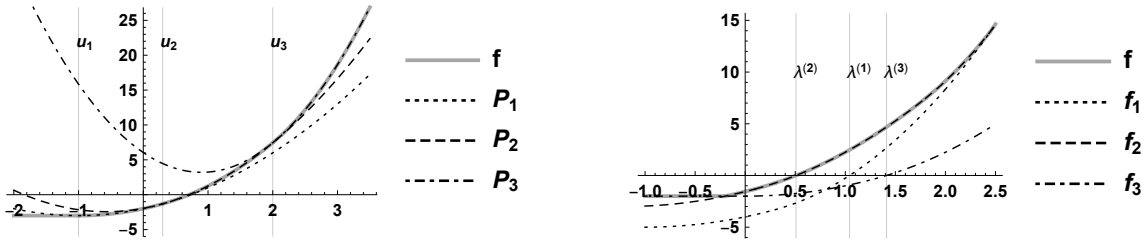

Figure 4: Left : illustration of Proposition 2.4, with $I = 3$ and $u_1 < u_2 < u_3$. The function f is piecewise quadratic : $f = P_1$ on $[u_1, u_2]$, $f = P_2$ on $[u_2, u_3]$, and $f = P_3$ on $[u_3, \infty[$, see (11). In this case, the unique root of f is also the largest root of P_2, which belongs to the interval $[u_2, u_3]$. Right : illustration of Proposition 2.5, with $K = 3$. The function f is the maximum (a.k.a. the upper enveloppe) of the non-decreasing functions f_1, f_2, f_3, whose unique root is $\lambda^{(1)}$, $\lambda^{(2)}$, $\lambda^{(3)}$ respectively. The unique root of f is thus $\lambda^* = \min\{\lambda^{(k)}; 1 \leq k \leq 3\}$ (here $\lambda^* = \lambda^{(2)}$).

Proposition 2.4. *With the notations of Lemma 2.3. Introduce a permutation $i_1, \cdots, i_I$ of $\{1, \cdots, I\}$ such that $u_{i_1} \leq \cdots \leq u_{i_I}$, and define for all $1 \leq r \leq I$*

$$a_r := \sum_{1 \leq s \leq r} \alpha_{i_s}, \qquad b_r := \sum_{1 \leq s \leq r} \alpha_{i_s} u_{i_s}, \qquad c_r := \sum_{1 \leq s \leq r} \alpha_{i_s} u_{i_s}^2 - h^2 \rho^2,$$

Then f admits the piecewise quadratic expression

$$f(\lambda) = P_r(\lambda) := a_r \lambda^2 - 2 b_r \lambda + c_r, \qquad \text{for all} \quad \lambda \in J_r := [u_{i_r}, u_{i_{r+1}}], \tag{11}$$

with the convention $J_I := [u_{i_I}, \infty[$. Denoting $r_* := \max\{r;\, 1 \leq r \leq I,\, f(u_{i_r}) < 0\}$, one has

- *$b_r^2 - a_r c_r > 0$ for all $1 \leq r \leq r_*$, so that $\lambda_r := (b_r + \sqrt{b_r^2 - a_r c_r})/a_r$ is well defined.*
- *$\lambda_{r_*} \in J_{r_*}$ is the unique root of f, whereas $\lambda_r > u_{i_{r+1}}$ for all $r < r_*$.*

Proof. The proof is illustrated on Figure (4, left). Assume w.l.o.g. that $u_1 \leq \cdots \leq u_I$, so that $i_r = r$ for all $1 \leq r \leq I$. The piecewise quadratic expression (11) on $J_r := [u_r, u_{r+1}]$, is obtained by inserting $(\lambda - u_i)_+ = \lambda - u_i$ if $i \leq r$, and $(\lambda - u_i)_+ = 0$ if $i > r$, in the defining expression (10).

Recalling that f is increasing on $[u_1, \infty[$, we obtain that $f(u_1) \leq \cdots \leq f(u_{r_*}) < 0$. The polynomial P_r thus takes a negative value $P_r(u_r) = f(u_r) < 0$, for any $r \leq r_*$, and has a positive dominant coefficient $a_r \geq \alpha_1 > 0$. It follows that P_r admits two distinct real roots, for any $r \leq r_*$, the largest being λ_r. When $r < r_*$ one has $P_r(u_{r+1}) < 0$, and thus $\lambda_r > u_{r+1}$ as announced. On the other hand $P_r(u_{r_*}) = f(u_{r_*}) < 0$ and $P_r(u_{r_*+1}) = f(u_{r_*+1}) \geq 0$ (or $P_r(\lambda) \to \infty$ as $\lambda \to \infty$ in the case $r_* = I$), so that P_r admits a root in the interval J_{r_*} by the intermediate value theorem. Since $P_{r_*} = f$ is increasing on the interval J_{r_*}, this is the largest root of P_{r_*}. We have found a root $\lambda_{r_*} \in J_{r_*}$ of f, and it is unique by Lemma 2.3, which concludes the proof. □

The complexity of Algorithm 3 is $s(I) + \mathcal{O}(I)$, where the first term accounts for the sorting step, and the second term for the floating point operations. In standard isotropic fast-marching, the number of terms is the space dimension $I \in \{2, 3\}$ and the sorting step has a negligible cost; however wide stencil schemes behave differently, especially the Euler-Mumford model for which $I = 30$. In the latter case, a naive bubble sort involving $s(I) = \mathcal{O}(I^2/2)$ compare and swap operations becomes prohibitive, whereas some other methods with optimal asymptotic complexity $s(I) = \mathcal{O}(I \ln I)$ performed poorly to the GPU due to their complex control flow, see Remark 2.1. Best results were obtained by applying a network sort [Knu98] (an efficient branchless sorting method) independently to the 15 first and the 15 last values, followed by a merge operation, cutting the overall computation time by more than half. In the general case, discussed below, the complexity becomes $Ks(I + J) + K\mathcal{O}(I + J)$.

Proposition 2.5 below shows how to characterize and compute the Gauss-Siedel update $\lambda^* = \Lambda u(p)$ when the numerical scheme has the general form (8), with arbitrary parameters I, J, K. It is obtained as the unique root of $f : \mathbb{R} \to \mathbb{R}$ defined by

$$f(\lambda) := \max_{1 \leq k \leq K} f_k(\lambda), \quad \text{where } f_k(\lambda) := \sum_{1 \leq i \leq I} \alpha_{ik} (\lambda - u_{ik})_+^2 + \sum_{1 \leq j \leq J} \beta_{jk} (\lambda - u'_{jk})_+^2 - \rho^2 h^2, \quad (12)$$

and where $u_{ik} := u(p + he_{ik})$ and $u'_{jk} := \min\{u(p - hf_{jk}), u(p + hf_{jk})\}$. Note that the weights α_{ik} and β_{jk} are non-negative by design for all i, j, k, yet for simplicity we assume that they are positive up to dropping the corresponding terms in the sums.

Proposition 2.5. *The function f_k defined by* (12)*, where $1 \leq k \leq K$, admits a unique root denoted $\lambda^{(k)}$. It can be characterized and computed by applying Lemma 2.3, Proposition 2.4, and Algorithm 3 to the $I + J$ weights $(\alpha_{1k}, \cdots, \alpha_{Ik}, \beta_{1k}, \cdots, \beta_{Jk})$ and values $(u_{1k}, \cdots, u_{Ik}, u'_{1k}, \cdots, u'_{Jk})$.*

The function f defined by (12) *admits a unique root, which is $\lambda^* = \min\{\lambda^{(k)}; 1 \leq k \leq K\}$.*

Proof. The function f_k has a structure similar to (10), up to gathering the two sums featuring I and J terms respectively into a single sum with $I + J$ terms, which establishes the first point.

The proof of the second point is illustrated on Figure (4, right). Since f_k is non-decreasing and admits the unique root $\lambda^{(k)}$, one has $f_k < 0$ on $]-\infty, \lambda^{(k)}[$, $f_k(\lambda^{(k)}) = 0$, and $f_k > 0$ on $]\lambda^{(k)}, \infty[$. It immediately follows that $f < 0$ on $]-\infty, \lambda^*[$, $f(\lambda^*) = 0$, and $f > 0$ on $]\lambda^*, \infty[$, which concludes the proof. □

3 Numerical experiments

We illustrate our numerical solver of curvature penalized shortest paths in variety of contexts ranging from motion planning with obstacles or drift, to image segmentation, and to the configuration of radar systems. Some of the test cases are new, whereas others are closely related to previous works [CMC16, CMC17, DDBM19, DMMP18, MD17, Mir18] and are meant to illustrate the benefits of the GPU solver over an earlier CPU implementation in common use cases. Test data is synthetic except for the medical image segmentation problem §3.3.

We report in Table 2 the running times of the GPU eikonal solver presented in this paper, and of the CPU solver introduced in [Mir18], as well as the GPU/CPU speedup which varies significantly depending on the experiment. Indeed, the running time of the GPU eikonal solver, which is an iterative method, depends on the presence and layout of obstacles or slow regions in the test case as noted in [WDB+08]. This in contrast with the fast-marching-like method [Mir18] implemented on the CPU, which is guaranteed to update each discretization point at most $N_{\text{neigh}} = K(I + 2J)$ times where I, J, K are the scheme parameters (8) (for this reason, slightly abusively, fast-marching is referred to as a single pass method), and whose complexity $\mathcal{O}(N_{\text{neigh}} N \ln N)$ is independent of the specific test case, where N is the total number of discretization points. The logarithmic factor in the fast-marching complexity comes from the overhead of managing a priority queue of the trial discretization points, sorted by the solution values, which is implemented using a binary heap. This complexity can be slightly improved to $\mathcal{O}(N_{\text{neigh}} N + N \ln N)$ using the more advanced Fibonacci heap data structure [FT87], but with little practical benefit in our experience [Mir14b]. Indeed, the sequential nature of the fast marching method is the main obstacle to improving its computation time.

The numerical experiments presented in the following sections are designed to illustrate the following features of the eikonal solver introduced in this paper:

1. *Geodesics in an empty domain.* Illustrates the qualitative properties of the different path models, and the GPU/CPU speedup in the ideal case.
2. *Fastest exit from a building.* Illustrates the implementation of walls and thin obstacles, which is non-trivial with wide stencils as described in Remark 2.2.
3. *Retinal vessel segmentation.* Illustrates a realistic application to image processing, based on the choice of a carefully designed cost function.

4. *Boat routing.* Illustrates a curvature penalty whose strength and asymmetry properties vary over the PDE domain, as described in Remark 1.2.
5. *Radar configuration.* Illustrates the automatic differentiation of the eikonal PDE solution u w.r.t. the cost function ρ, for the optimization of a complex objective.

Remark 3.1 (Computation time and hardware characteristics)**.** *Program runtime is dependent on the hardware characteristics of each machine. The reported CPU and GPU times were obtained on the Blade® Shadow cloud computing service, using the provided Nvidia® GTX 1080 graphics card for the GPU eikonal solver, and an Intel® Xeon E5-2678 v3 for the CPU eikonal solver (a single thread was used, with turbo frequency 3.1Ghz).*

3.1 Geodesics in an empty domain

We compute minimal geodesics for the Reeds-Shepp, Reeds-Shepp forward, Euler-Mumford elastica and Dubins model, in the domain $[-1,1]^2 \times \mathbb{S}^1$ without obstacles. The front is propagated from the seed point (x_*, θ_*) placed at the origin $x_* = (0,0)$ and imposing a horizontal initial tangent $\theta_* = 0$. Geodesics are backtracked from several tips (x^*, θ^*) where x^* is placed at 16 regularly spaced points in the domain, whereas θ^* is chosen randomly (but consistently across all models).

This experiment is meant to illustrate the qualitative differences between minimal geodesic paths associated with the four curvature penalized path models, see Figure 1. The Reeds-Shepp car can move both forward and backward, and reverse gear along its path, as evidenced by the *cusps* along several trajectories. The Reeds-Shepp forward variant cannot move backward, but has the ability to rotate in place (with a cost), and such behavior can often be observed at the endpoints of the trajectories [DMMP18]. The Elastica model produces pleasing smooth curves, which have a physical interpretation as the rest positions of elastic bars. Trajectories of the Dubins model have a bounded radius of curvature, and can be shown to be concatenations of straight segments and of arcs of circles, provided the cost function is constant as here.

The generalized eikonal PDE (5) or (6) is discretized on a $300 \times 300 \times 96$ Cartesian grid, following (8), thus producing a coupled system of equations featuring 8.6 million unknowns[3]. Computation time for the GPU eikonal solver ranges from 0.28s (Reeds-Shepp forward) to 1.54s (Euler-Mumford elastica), reflecting the complexity of the discretization stencil, see Figure 2. A substantial speedup ranging from $60\times$ to $120\times$ is obtained over the CPU implementation; let us nevertheless acknowledge that, as noticed in [WDB+08], the absence of obstacles and of a position dependent speed function is usually the best case scenario for an iterative eikonal solver such as our GPU implementation.

3.2 Fastest exit from a building

We compute minimal paths within a museum map, for the four curvature penalized models under consideration in this paper, as illustrated on Figure 5. Due to the use of rather wide stencils, often 7 pixels long see Figure 2, some intersection tests are needed to avoid propagating the front through the walls, which are one pixel thick only. A careful implementation, as described in Remark 2.2, allows to bypass most of these intersection tests and limits their impact on computation time. In contrast with [JW08], we do not consider "slightly permeable walls", since they would not be correctly handled with our wide stencils, and since as far as we know they have little relevance in applications. A closely related experiment is presented in [DMMP18] for the Reeds-Shepp models, using a CPU eikonal solver.

The front propagation starts from two seed points located at the exit doors, and a tip is placed in each room for geodesic backtracking, with an arbitrary orientation. The extracted paths are smooth (Euler-Mumford case) or have a bounded curvature radius (Dubins case), but minimize a functional (1) which is unrelated with safety and thus may not be directly suitable for motion

[3]For this particularly simple problem (with a constant cost function, without walls), results visually quite similar can be obtained at a fraction of the cost using a smaller discretization grid, eg. of size $100 \times 100 \times 64$.

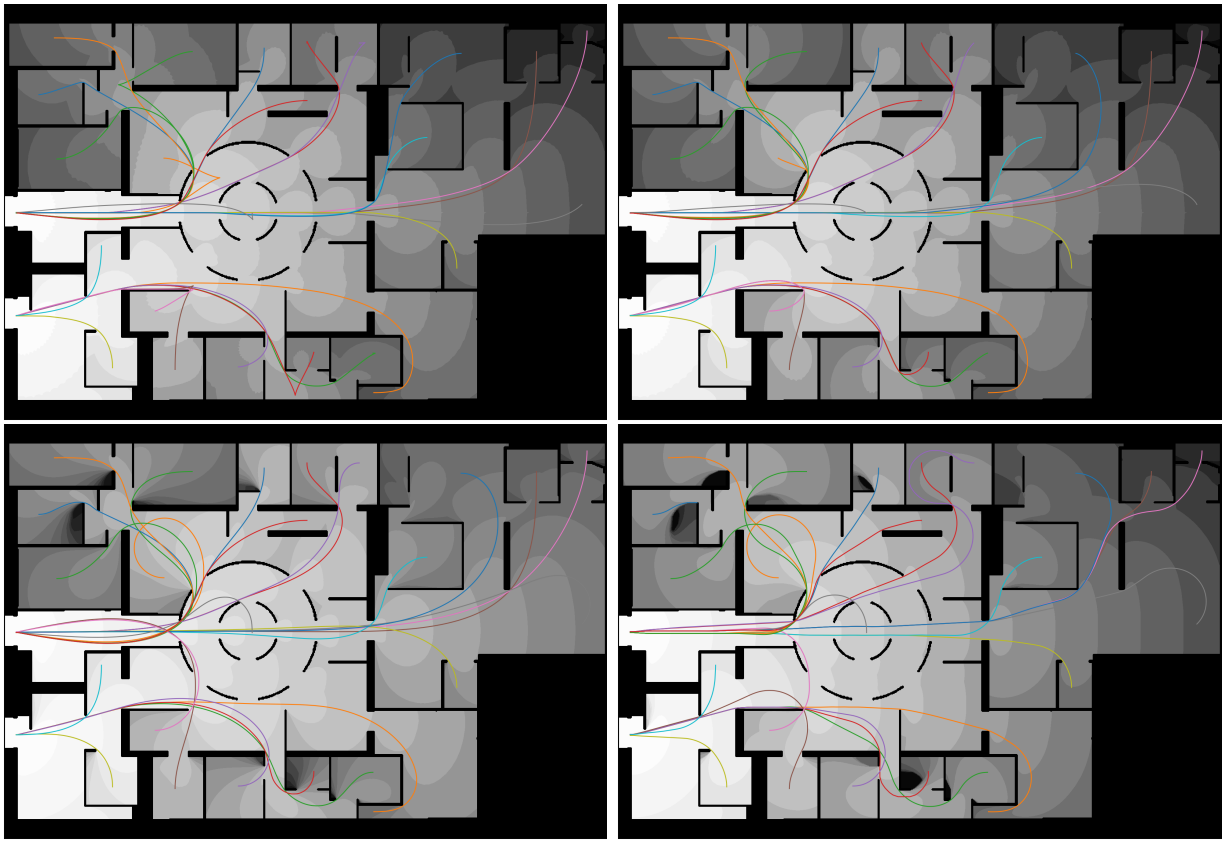

Figure 5: Planar projections of minimal geodesics for the Elastica and Dubins model (left to right). Two seed points at the exits, with horizontal tangents. Geodesics are backtracked from one tip point in each room, with a given but arbitrary tangent.

planning. Indeed, in many places they are tangent to the obstacles, walls, and doorposts, without any visibility behind, which is a hazardous way to move.

The PDE is discretized on a Cartesian grid of size $705 \times 447 \times 60$, where the first two factors are the museum map dimensions, and the third factor is the number of angular orientations, for a total of 19 million unknowns. Computation time on the GPU ranges from 0.59s (Reeds-Shepp forward) to 3.2s (Euler-Mumford elastica), a reduction by approximately $50\times$ over the CPU eikonal solver.

3.3 Tubular structure segmentation

A popular approach for segmenting tubular structures in medical images, such as blood vessels on the retinal background as illustrated on Figure 6, is to devise a geometric model whose minimal paths (between suitable endpoints) are the centerlines of the desired structures. For that purpose a key ingredient, not discussed here, is the careful design of a cost function $\rho : \mathbb{R}^2 \times \mathbb{S}^1 \to]0, \infty]$ which is small along the vessels of interest in their tangent direction, and large elsewhere [PKP09]. Curvature penalization, and in particular the Reeds-Shepp forward and Euler-Mumford elastica models [CMC16, CMC17, DMMP18], helps avoid a classical artifact where the minimal paths do not follow a single vessel but jump form one to another at crossings.

The test cases have size $512 \times 512 \times 60$, $387 \times 449 \times 60$ and $398 \times 598 \times 60$ respectively, and the computation time of the GPU eikonal solver ranges from 1s (Reeds-Shepp forward) to 3s (Euler-Mumford elastica) on the GPU. This is compatible with user interaction, in contrast with CPU the run time which is $30\times$ to $80\times$ longer, see Table 2. Note that by construction, the front propagation is fast along the blood vessels, and slower in the rest of the domain. This specificity

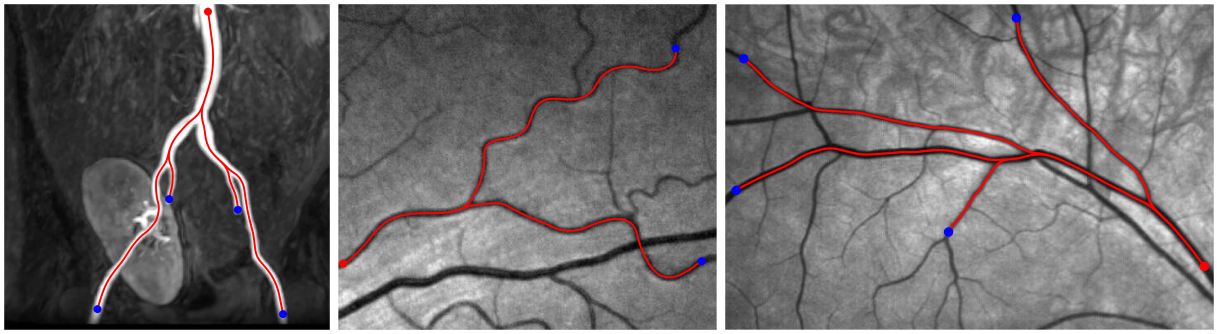

Figure 6: Segmentation of tubular structure centerlines using the Reeds-Shepp forward and Euler-Mumford elastica models, following [CMC17]. Left : Blood vessels in Magnetic Resonance Angiography (MRA) data. Center and right : Blood vessels on an image of the retina.

plays against iterative methods, which are most efficient when velocity is uniform [JW08], yet the speedup achieved by the GPU solver remains very substantial. Computation time could in principle be further reduced, both on the CPU and the GPU, by using advanced stopping criteria and restriction methods [CCV13] to avoid solving the eikonal PDE on the whole domain.

3.4 Boat routing with a trailer

The Dubins-Zermelo-Markov model [BT13] describes a vehicle subject to a drift, and whose speed and turning radius *as measured before the drift is applied* are bounded. This problem was introduced to us in the context of maritime seismic prospection, where boats drag long trails of acoustic sensors, and are subject to water currents. Optimal Dubins-Zermelo-Markov trajectories, with drift defined by the water flow, may help avoid entangling and damaging these trails, and reduce the prospection times. In this synthetic experiment we use the drift velocity $V(x) = 0.6\sin(\pi x_0)\sin(\pi x_1)x/\|x\|$ at each point $x = (x_0, x_1) \in [-1, 1]^2$. Our vehicle has unit speed, and turning radius $\xi = 0.3$. Our numerical results are illustrated on Figure 7.

From the mathematical standpoint, the Dubins-Zermelo-Markov model can be rephrased in the form of the original Dubins model, but with a curvature penalty which is scaled, shifted (asymmetric), and depends on the current point, as described in Remark 1.2. This does not raise particular issues for discretization, except that the weights and offsets of the numerical scheme (8) depend on the full position $(x, \theta) \in \mathbb{R}^2 \times \mathbb{S}^1$, rather than the orientation $\theta \in S^1$ alone.

The boat routing problem is discretized on a grid of size $151 \times 151 \times 96$. Computation time on the GPU is 0.34s if stencils are pre-computed and stored, and 0.52s if they are recomputed on the fly when needed. The second approach (recomputation) uses significantly less GPU memory, which is usually a scarce ressource, hence we regard it as default despite the longer runtime, see the discussion §2.2; it is nevertheless 59× faster than the CPU implementation.

3.5 Optimization of a radar configuration

We consider the optimization of a radar system, so as to maximize the probability of detection of an intruder vehicle. The intruder has full knowledge of the radar configuration, and does its best to avoid detection, but is subject to maneuverability constraints as does a fast plane. Following [MD17, DDBM19] the intruder is modeled as a Dubins vehicle, traveling at unit speed with a turning radius of 0.2, whose trajectory starts and ends at a given point $x_* \in \Omega$ and which must visit a target keypoint $x^* \in \Omega$ in between[4]. The problem takes the generic form

$$\sup_{\xi \in \Xi} \inf_{\gamma \in \Gamma} \mathcal{E}(\xi; \gamma), \tag{13}$$

[4]This is achieved by concatenating a trajectory $(x_*, \theta_0) \in \Omega \times \mathbb{S}^1$ to (x^*, φ), with a reversed trajectory from (x_*, θ_1) to $(x^*, \varphi + \pi)$, where $\theta_0, \theta_1, \varphi \in \mathbb{S}^1$ are arbitrary, see [MD17].

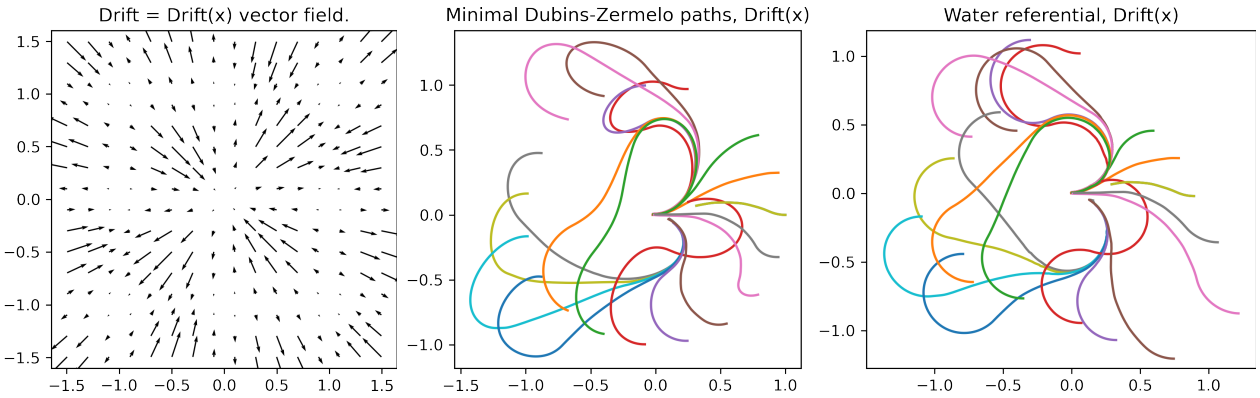

Figure 7: Illustration of the Dubins-Zermelo-Markov problem. Left : drift velocity (water current). Center : shortest paths for this model, from the seed $(0,0)$ to some arbitrary targets, with horizontal tangents imposed at the endpoints. Right : the same shortest paths, to which the drift associated with the vector field is added. In this referential, the radius of curvature does not exceed the prescribed bound ξ.

where Ξ is the set of radar configurations, and Γ is the set of admissible trajectories. A trajectory γ escapes detection from a radar configured as ξ with probability $\exp(-\mathcal{E}(\xi;\gamma))$. Following (1), a trajectory is represented as a pair $\gamma = (\boldsymbol{x},\boldsymbol{\theta}) : [0,L] \to \Omega \times \mathbb{S}^1$, and its cost is defined as

$$\mathcal{E}(\xi;\gamma) = \int_0^L \rho(\boldsymbol{x},\boldsymbol{\theta};\xi)\,\mathcal{C}(\dot{\boldsymbol{\theta}})\,\mathrm{d}l$$

where $\mathcal{C}$ denotes the Dubins cost (2, right), and $\rho(x,\theta;\xi)$ is an instantaneous probability of detection depending on the radar configuration ξ, and on the intruder position x and orientation θ. We refer to [DDBM19] for a discussion of the detection probability model, and settle for a synthetic and simplified yet already non-trivial construction. The detection probability is the sum of three terms $\rho(x,\theta;\xi) = \sum_{i=1}^{3} \tilde{\rho}(x,\theta;y_i,r_i,v_i)$, corresponding to as many radars, each of the following form illustrated on Figure (8, top left)

$$\tilde{\rho}(x,\theta;\,y,r,v) := \frac{1}{1+2\|x-y\|^2}\,\sigma\Big(\frac{\|x-y\|}{r}\Big)\,\sigma\Big(\frac{\langle \mathfrak{e}(\theta), x-y\rangle}{v\|x-y\|}\Big). \tag{14}$$

where y is the radar position, $\sigma(s) = 1-((1+\cos(2\pi s))/2)^4$ is a function vanishing periodically, r is the ambiguous distance period, and v is the ambiguous radial velocity period. Optimal trajectories for escaping detection by one or several of these radars are shown on Figure (8, top right and bottom left). The ambiguous periods r and v are related to the *pulse repetition interval* and *frequency* used by the radar, and their product is bounded below. In this experiment, we choose to optimize the position of the first radar x_1 within the disk D_1 centered at $(-0.4,0.4)$ and of radius 0.3, and the position of the second radar x_2 within the segment S_2 of endpoints $(-0.2,0.8)$ and $(0.5,0.8)$, whereas the third radar is fixed at $X_3 = (0.6,0.3)$. We also optimize the blind distances $r_1,r_2,r_3 \in]0,\infty[$, and we define the bling velocities as $v_i = 0.2/r_i$ for $1 \le i \le 3$, reflecting the fact that the product $r_i v_i$ is constrained by the physics of the radar system. Summarizing, the collection of radar configuration parameters ξ, and their domain Ξ, are defined for this experiment as :

$$\xi = (x_1,x_2,x_3,r_1,r_2,r_3), \qquad \Xi := D_1 \times S_2 \times \{X_3\} \times]0,\infty[^3.$$

The trajectory of the intruder has a curvature radius bounded below by 0.2 (Dubins model), stays within the rectangular domain $[-1,1]\times[0,1]$, starts and ends at $(-0.9,0.5)$, and visits the target point $(0.8,0.5)$, which completes the description of the test case.

Minimization over the parameter $\gamma \in \Gamma$ in (13) is solved numerically using the eikonal solver presented in this paper, thus defining a function $\mathcal{E}(\xi) := \inf\{\mathcal{E}(\xi;\gamma);\ \gamma \in \Gamma\}$ depending on the radar

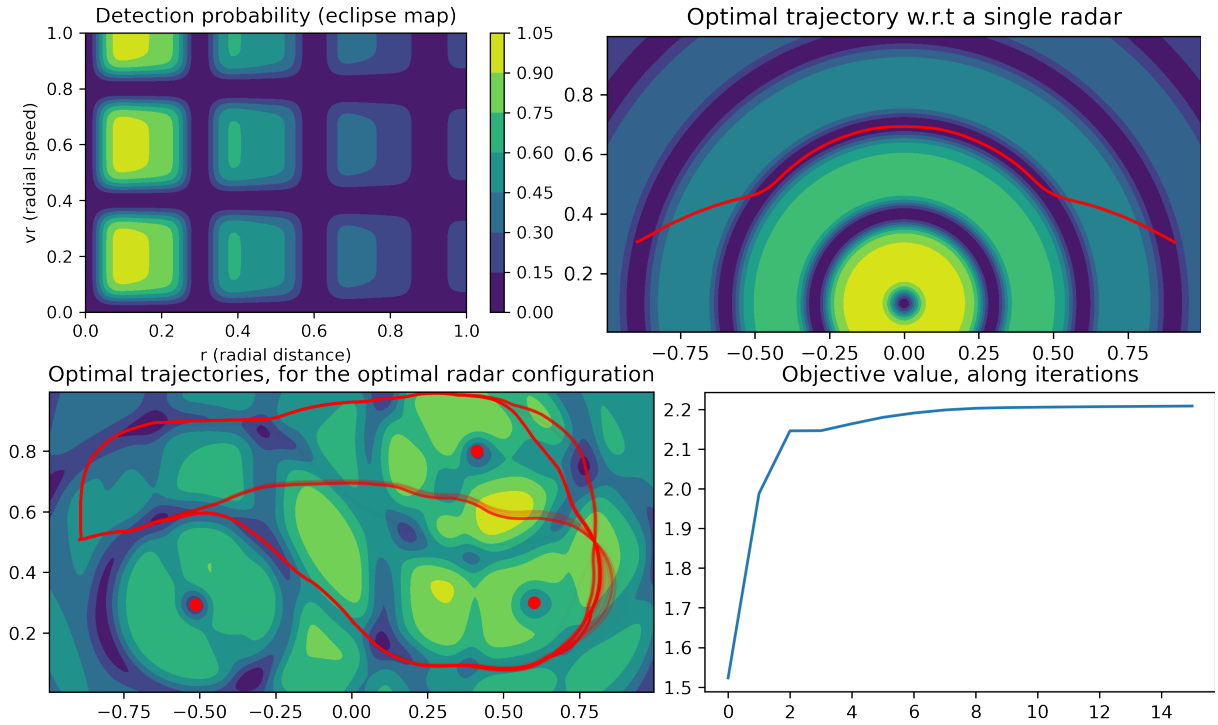

Figure 8: (Top left) Instantaneous detection probability of a vehicle by a radar, depending on the radial distance r and radial velocity v of the vehicle, in addition to the radar parameters, see (14). Note the blind distance and blind velocity periods. (Top right) Trajectory minimizing the probability of detection between two points in the presence of a single radar. It approximates a concatenation of circles, at a multiple of the blind radial distance, and spirals, corresponding to a multiple of the blind radial velocity. (Bottom left) Configuration of three radars locally optimized, see text, to detect trajectories from the left seed point to the right tip and back. Best adverse trajectories. (Bottom right) Objective value, $\mathcal{E}(\xi)$ see text, along the iterations of gradient ascent.

configuration alone $\xi \in \Xi$. We differentiate $\mathcal{E}(\xi)$ in an automatic manner as described in [MD17], and optimize this quantity by gradient ascent, with a projection of the updated parameters ξ onto the admissible domain Ξ at each gradient step. Using these tools, a *local* maximum of $\mathcal{E}(\xi)$ is reached in a dozen iterations approximately, see Figure (8, bottom right). Computation time is dominated by the cost of solving a generalized eikonal equation in each iteration, which takes 0.26s on the GPU and 9.6s on the CPU (Dubins model on a $200 \times 100 \times 96$ grid). Since the optimization landscape is highly non-convex, obtaining the global maximum w.r.t. ξ would require a non-local optimization method in complement or replacement of local gradient ascent, thus requiring many more iterations and benefitting even more from GPU acceleration.

4 Conclusion and perspectives

Geodesics and minimal paths are ubiquitous in mathematics, and their efficient numerical computation has countless applications. In this paper, we present a numerical method for computing paths which globally minimize a variety of energies featuring their curvature, by solving a generalized anisotropic eikonal PDE, and which takes advantage of the massive parallelism offered by GPU hardware for computational efficiency. In comparison with previous CPU implementations, a computation time speed up by 30× to 120× is achieved, which enables convenient user interaction in the context of image processing and segmentation, and reasonable run-times for applications

such as radar configuration which solve these problems within an inner optimization loop.

Future work will be devoted to additional applications, to efficient implementations of wide stencil schemes associated with other classes of Hamilton-Jacobi-Bellman PDEs, and to the study of numerical schemes based on different compromises in favor of e.g. allowing grid refinement or using shorter finite different offsets.

References

[BCD08] Martino Bardi and Italo Capuzzo-Dolcetta. *Optimal control and viscosity solutions of Hamilton-Jacobi-Bellman equations*. Springer Science & Business Media, 2008.

[BCR10] Ugo Boscain, Grégoire Charlot, and Francesco Rossi. Existence of planar curves minimizing length and curvature. *Proceedings of the Steklov institute of mathematics*, 270(1):43–56, 2010.

[BR06] Folkmar Bornemann and Christian Rasch. Finite-element Discretization of Static Hamilton-Jacobi Equations based on a Local Variational Principle. *Computing and Visualization in Science*, 9(2):57–69, June 2006.

[BT13] Efstathios Bakolas and Panagiotis Tsiotras. Optimal synthesis of the Zermelo–Markov–Dubins problem in a constant drift field. *Journal of Optimization Theory and Applications*, 156(2):469–492, 2013.

[CCV13] Zachary Clawson, Adam Chacon, and Alexander Boris Vladimirsky. Causal Domain Restriction for Eikonal Equations. *arXiv.org*, September 2013.

[CMC16] Da Chen, Jean-Marie Mirebeau, and Laurent D. Cohen. A New Finsler Minimal Path Model With Curvature Penalization for Image Segmentation and Closed Contour Detection. *Computer Vision and Pattern Recognition (CVPR)*, pages 355–363, June 2016.

[CMC17] Da Chen, Jean-Marie Mirebeau, and Laurent D. Cohen. Global Minimum for a Finsler Elastica Minimal Path Approach. *International Journal of Computer Vision*, 122(3):458–483, 2017.

[DBRS13] Remco Duits, U Boscain, F Rossi, and Y Sachkov. Association Fields via Cuspless Sub-Riemannian Geodesics in SE(2). *Journal of Mathematical Imaging and Vision*, 49(2):384–417, December 2013.

[DCC+21] François Desquilbet, Jian Cao, Paul Cupillard, Ludovic Métivier, and Jean-Marie Mirebeau. Single pass computation of first seismic wave travel time in three dimensional heterogeneous media with general anisotropy. 2021.

[DDBM19] Johann Dreo, François Desquilbet, Frédéric Barbaresco, and Jean-Marie Mirebeau. Netted multi-function radars positioning and modes selection by non-holonomic fast marching computation of highest threatening trajectories. In *International RADAR'19 conference*, 2019.

[DMMP18] Remco Duits, Stephan PL Meesters, Jean-Marie Mirebeau, and Jorg M Portegies. Optimal paths for variants of the 2D and 3D Reeds-Shepp car with applications in image analysis. *Journal of Mathematical Imaging and Vision*, pages 1–33, 2018.

[FKW13] Zhisong Fu, Robert M Kirby, and Ross T Whitaker. A fast iterative method for solving the eikonal equation on tetrahedral domains. *SIAM Journal on Scientific Computing*, 35(5):C473–C494, 2013.

[FT87] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[GHZ18] Daniel Ganellari, Gundolf Haase, and Gerhard Zumbusch. A massively parallel Eikonal solver on unstructured meshes. *Computing and Visualization in Science*, 19(5-6):3–18, 2018.

[JW08] Won-Ki Jeong and Ross T Whitaker. A Fast Iterative Method for Eikonal Equations. *SIAM Journal on Scientific Computing*, 30(5):2512–2534, July 2008.

[Knu98] Donald E Knuth. *The art of computer programming: Volume 3: Sorting and Searching*. Addison-Wesley Professional, 1998.

[MD17] Jean-Marie Mirebeau and Johann Dreo. Automatic differentiation of non-holonomic fast marching for computing most threatening trajectories under sensors surveillance. In *International Conference on Geometric Science of Information*, pages 791–800. Springer, 2017.

[Mir14a] Jean-Marie Mirebeau. Anisotropic Fast-Marching on cartesian grids using Lattice Basis Reduction. *SIAM Journal on Numerical Analysis*, 52(4):1573–1599, January 2014.

[Mir14b] Jean-Marie Mirebeau. Efficient fast marching with Finsler metrics. *Numerische Mathematik*, 126(3):515–557, 2014.

[Mir18] Jean-Marie Mirebeau. Fast-marching methods for curvature penalized shortest paths. *Journal of Mathematical Imaging and Vision*, 60(6):784–815, 2018.

[Mir19] Jean-Marie Mirebeau. Riemannian Fast-Marching on Cartesian Grids, Using Voronoi's First Reduction of Quadratic Forms. *SIAM Journal on Numerical Analysis*, 57(6):2608–2655, 2019.

[MP19] Jean-Marie Mirebeau and Jorg Portegies. Hamiltonian fast marching: A numerical solver for anisotropic and non-holonomic eikonal pdes. *Image Processing On Line*, 9:47–93, 2019.

[Obe08] A M Oberman. Wide stencil finite difference schemes for the elliptic Monge-Ampere equation and functions of the eigenvalues of the Hessian. *Discrete Contin Dyn Syst Ser B*, 2008.

[PKP09] M Pechaud, R Keriven, and G Peyré. Extraction of tubular structures over an orientation domain. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPR Workshops)*, pages 336–342. IEEE, 2009.

[RT92] Elisabeth Rouy and Agnès Tourin. A Viscosity Solutions Approach to Shape-From-Shading. *SIAM Journal on Numerical Analysis*, 29(3):867–884, July 1992.

[Set96] James A. Sethian. A fast marching level set method for monotonically advancing fronts. *Proceedings of the National Academy of Sciences*, 93(4):1591–1595, 1996.

[Set99] James A. Sethian. *Level Set Methods and Fast Marching Methods: Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press, 1999.

[SV03] James A. Sethian and Alexander Boris Vladimirsky. Ordered upwind methods for static Hamilton-Jacobi equations: theory and algorithms. *SIAM Journal on Numerical Analysis*, 41(1):325–363, 2003.

[Tsi95] J.N. Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE transactions on Automatic Control*, 40(9):1528–1538, September 1995.

[WDB+08] Ofir Weber, Yohai S Devir, Alexander M Bronstein, Michael M Bronstein, and Ron Kimmel. Parallel algorithms for approximation of distance maps on parametric surfaces. *ACM Transactions on Graphics (TOG)*, 27(4):104–16, October 2008.

A Monotony and causality

We briefly present in this appendix the concepts of monotony and causality, which are two key properties for the analysis of discretization schemes of eikonal equations, and we discuss their relevance to a GPU implementation. We refer the interested reader to [MP19, §2.1 and §A] for additional discussion, and for the proofs of the results presented below.

A (finite differences) scheme on a finite set X is a mapping $F : \mathbb{R}^X \to \mathbb{R}^X$ defined as

$$Fu(p) = \mathfrak{F}(p, u(p), [u(p) - u(q)]_{q \in X \setminus \{p\}}),$$

for each $u \in X$, where $\mathfrak{F}$ is continuous. The scheme is said discrete degenerate elliptic (DDE) if $\mathfrak{F}$ is non-decreasing w.r.t. the second and third variables. It is said causal if $\mathfrak{F}$ only depends on the positive part $[\max\{0, u(p) - u(q)\}]_{q \in X \setminus \{p\}}$ of the third variable. The schemes considered in this paper (8) obey these two properties.

Given such a scheme, one is often able to prove that the univariate non-linear equation

$$\mathfrak{F}(p, \lambda, [\lambda - u(q)]_{q \in X \setminus \{p\}}) = 0 \tag{15}$$

admits a unique root $\lambda \in \mathbb{R}$, thus defining the Gauss-Siedel update operator $\Lambda u(p) := \lambda$, see Lemma 2.3 for the class of schemes considered in this paper. This operator is monotonous, in the sense that $\Lambda u \leq \Lambda v$ if $u \leq v$ on X, provided the original scheme F is DDE. It is causal, in the sense that $\Lambda u(p)$ *may depend on* $u(q)$ *only if* $\Lambda u(p) > u(q)$, provided the original scheme F is in addition causal in the previously defined sense. See [MP19, Proposition A.4] for a complete statement and proof.

A solution $Fu = 0$ to a finite difference scheme defines a fixed point $\Lambda u = u$ of the corresponding Gauss-Siedel update operator, and conversely. Alternatively, semi-Lagrangian discretizations of the eikonal equation directly define an operator Λ whose fixed point must be computed, and which is obtained via a variational principle [BR06] rather than from a finite difference scheme as in (15). In that case, monotony holds by construction, whereas the causality property can be derived from a geometrical acuteness property of the discretization stencils [Tsi95, SV03, Mir14a, Mir14b, DCC$^+$21].

The fixed point of a monotonous operator Λ can be obtained using a variety of iterative methods such as [RT92, BR06], or [FKW13, GHZ18, JW08, WDB$^+$08] on the GPU. For concreteness, global iteration is the simplest approach (but usually not the most efficient): let formally $u_0 := +\infty$, and define $u_{n+1} := \Lambda u_n$ for all $n \geq 0$. Then using monotony and an immediate induction argument one obtains that $u_{n+1} \leq u_n$ for all $n \geq 0$; under mild additional assumptions, one finds that u_n converges decreasingly to a fixed point of Λ, as desired, see e.g. [DCC$^+$21, Proposition D.3]. Monotony is of key significance for the GPU implementation, since it ensures stability along the iterations, and also allows to use a single array for reading and writing the solution values (data races are indeed innocuous, since the latest computed value is always the smallest and the closest to the fixed point).

The fixed point of an operator Λ which is both monotonous and causal can be computed in a single pass over the domain using the extremely efficient fast marching method (FMM), see [Set96] or [MP19, Proposition A.2]. The FMM however has little parallelization potential (notably, the parallel implementation [Tsi95] does not scale well due to the excessive overhead of message passing), hence it is less relevant to GPU eikonal solvers which instead rely on iterative methods as discussed above.

B Tables

Model	N_{fd}	Best tile	Best R
Isotropic (d=2)	2	24×24	48
Isotropic (d=3)	3	$4 \times 4 \times 4$	8
Reeds-Shepp (both)	4	$4 \times 4 \times 4$	6
Dubins	12	$4 \times 4 \times 4$	2
–	–	$4 \times 4 \times 2$	1
Euler-Mumford	30	$4 \times 4 \times 2$	1

Table 1: Number $N_{fd} = K(I + J)$ of finite differences terms in (8) for a variety of path models. Tile shape and number of iterations R in Algorithm 2, producing the smallest running time, found experimentally. Two sets of parameters are reported for Dubins model, since the corresponding running times results are close which and one is fastest depends on the test case. Simple models, whose stencil involves few and short finite differences, work best with large tile sizes and numerous iterations allowing the front to propagate within the tile, whereas complex models involving many wide finite differences and a costly update operator benefit from small tiles and few iterations.

Exp.	model	GPU(s)	CPU(s)	accel
Empty	RS rev	0.28	34.3	120×
	RS fwd	0.25	15.7	62×
	EM	1.53	117	76×
	Dubins	0.44	46.5	105×
Building	RS rev	1.37	50.5	37×
	RS fwd	0.59	29	49×
	EM	3.21	174	54×
	Dubins	1.02	55.4	54×
Boat	Dubins	0.52	30.2	59×
MRI	RS fwd	0.93	30.8	33×
	EM	3.32	275.9	83×
Retina1	RS fwd	0.66	21.1	32×
	EM	2.22	171.3	77×
Retina2	RS fwd	0.98	32.8	33×
	EM	3.21	256.1	80×
Radar	Dubins	0.26	9.57	37×

Table 2: Running time of the CPU and GPU eikonal solver, for the experiments presented §3.