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§

March 13, 2021

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 or forward only), 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 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 ten to hundred are obtained w.r.t sequential implementation. The efficiency and robustness of the method is illustrated in various contexts, ranging from motion planning to vessel segmentation and radar configuration.

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 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. Eventually, the GPU accelerated eikonal solver is often $50\times$ faster than the CPU fast marching method from [Mir18], see Table 2. In the considered applications, computations times on typical problem instances are often reduced from 30 seconds to less than one, which enables convenient user interaction.

Remark 1.1 (Intellectual property)**.** *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. Accelerations of the same order were first 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 different languages (Python/CUDA versus C++/OpenCL), do not share a single line of code, use different implementation tricks, and offer distinct functionality.*

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 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 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 some of the first order derivatives of the path $(\dot{\boldsymbol{x}}, \dot{\boldsymbol{\theta}})$.

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. This is in contrast with the state dependent penalty ρ which is essentially arbitrary. The considered curvature penalties are defined

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

by the three following expressions, which correspond to the Reeds-Shepp[2], 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, which allows in-place rotations[3]. The quadratic curvature penalty of the Euler-Mumford model corresponds to the energy of an elastic bar, hence minimal paths follow their rest position. 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 $(x,\theta) \in \Omega \times \mathbb{S}^1$:

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

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 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 $(x,\theta) \in \Omega \times \mathbb{S}^1$.*

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 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 feature curvature penalization, is defined by the operator $\mathcal{F}u = \|\nabla u\|$.

[2]The following description applies to the *forward* only variant of the Reeds-Shepp model, see Remark 1.3 for a discussion of the *reversible* variant.

[3]In full rigor, a parametrization by Euclidean length in the full state space (both physical and angular), or an arbitrary Lipschitz parametrization, is necessary to ensure the existence of a minimizer of (1) for the Reeds-Shepp forward model. Indeed, in-place rotations are path sections where the the physical velocity vanishes, but the angular velocity does not. See [Mir18] for a discussion of well posedness.

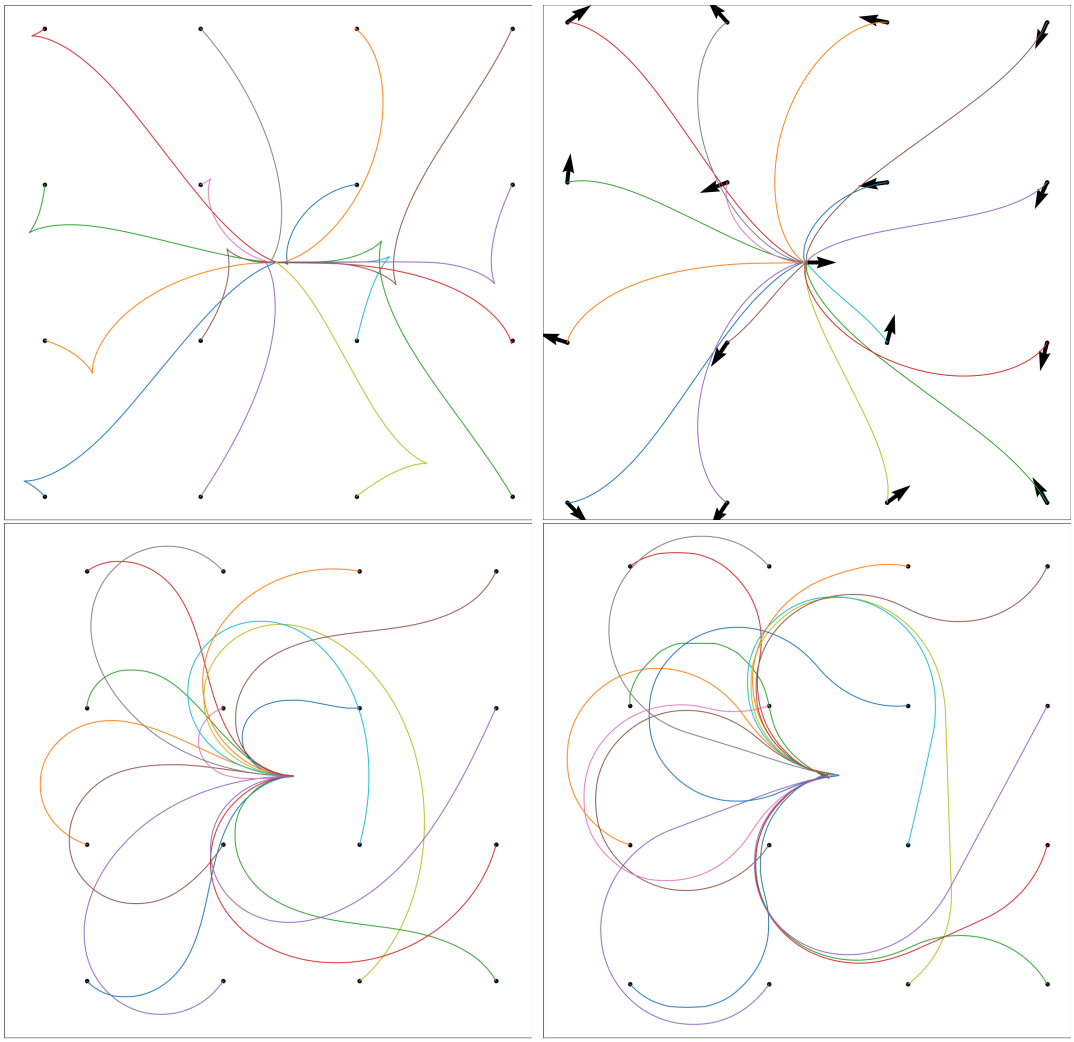

Figure 1: Planar projections of minimal geodesics for the Reeds-Shepp, Reeds-Shepp forward, Elastica and Dubins models (left to right). Seed point $(0, 0)$ with horizontal tangent, regularly spaced tip point with random tangent (but identical for all models).

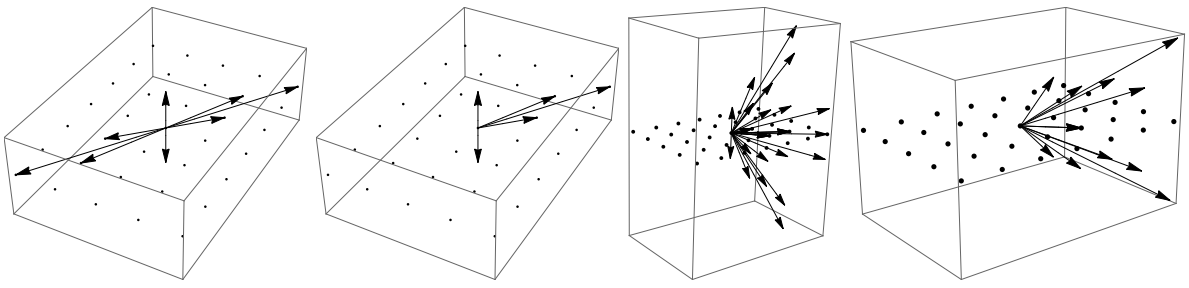

Figure 2: Discretization stencils used for the Reeds-Shepp reversible, Reeds-Shepp forward, Euler-Mumford, and Dubins models. Note the sparseness and anisotropy of the stencils. Model parameters: $\theta = \pi/3$, $\xi = 0.2$, $\varepsilon = 0.1$.

The variants of the eikonal PDE associated with the Reeds-Shepp forward, Euler-Mumford and Dubins models, involve the following non-linear and anisotropic operators [Mir18]: define $\mathcal{F}u(x,\theta)$ as, respectively

$$\sqrt{\max\{0,\langle\nabla_x u,\mathbb{e}_\theta\rangle\}^2+|\partial_\theta u|^2},\quad \frac{1}{2}(\langle\nabla_x u,\mathbb{e}_\theta\rangle+\sqrt{\langle\nabla_x u,\mathbb{e}_\theta\rangle^2+|\partial_\theta u|^2}),\quad \langle\nabla_x u,\mathbb{e}_\theta\rangle+|\partial_\theta u|. \tag{4}$$

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

$$X_h := (\Omega\times\mathbb{S}^1)\cap h\mathbb{Z}^3, \tag{5}$$

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 samping 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 an expression of the following form [MP19]

$$\max_{1\le k\le K}\Big(\sum_{1\le i\le I}\alpha_{ik}\max\big\{0,\frac{u(p)-u(p+he_{ik})}{h}\big\}^2+\sum_{1\le j\le J}\beta_{jk}\max\big\{0,\frac{u(p)-u(p\pm hf_{jk})}{h}\big\}^2\Big), \tag{6}$$

where I,J,K are fixed integers, $\alpha_{ik},\beta_{jk}\ge 0$ are non-negative weights, and $e_{ik},f_{jk}\in\mathbb{Z}^3$ are finite difference offsets, for all $1\le i\le I$, $1\le j\le J$, $1\le k\le K$. The weights and offsets may depend on the current point p. Before turning to the variants (4), 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 $w_{j1}=1$, $1\le j\le d$, 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 using weights and offsets defined by an appropriate decomposition of the inverse metric tensor, see [Mir19, MP19].

In the curvature penalized case, the weights and offsets in (6) implicitly depend on the base point $p=(x,\theta)$, at least through the angular coordinate θ consistently with the continuous PDE (4), and possibly the physical position x as well if the strength or 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, which involves a relaxation parameter $\varepsilon>0$ for the non-holonomic constraint (1, right), and simply report the meta-parameters for the Reeds-Shepp ($I=3$, $J=1$, $K=1$), Euler-Mumford ($I=30$, $J=0$, $K=1$), and Dubins ($I=6$, $J=0$, $K=2$) models, see Figure 2.

A fundamental property of discretization schemes of the form (6) is that they can be solved in a single pass over the domain, using a generalization of the fast-marching algorithm [Mir18, MP19, Mir19]. This is highly desirable when implementing CPU solver, but anecdotical for a GPU eikonal solver whose massive parallelism forbids taking advantage of this property. Nevertheless, those schemes are robust and well tested. Alternative approaches offering different compromises possibly more suited to GPUs will be considered in future works.

Remark 1.3 (Forward and reversible Reeds-Shepp models)**.** *The Reeds-Shepp model comes in two flavors [DMMP18]: the* forward *variant, presented above, and the (more standard)* reversible *variant, modeling a vehicle equipped with a reverse gear additionally. The latter is obtained by relaxing the constraint (1, right) into $\dot{\boldsymbol{x}}=\pm\mathbb{e}_{\boldsymbol{\theta}}$. In turn the eikonal PDE (4, left) is replaced with $\sqrt{\langle\nabla_x u,\mathbb{e}_\theta\rangle^2+|\nabla_\theta u|^2}$, whose discretization* (6) *uses the meta-parameters $I=0$, $J=4$, $K=1$.*

Remark 1.4 (Monotony and degenerate ellipticity)**.** *The discrete operator* (6) *is* degenerate elliptic*: $Fu(p)$ is a non-decreasing function of the finite differences $[u(p) - u(q)]_{q \in X_h \setminus \{p\}}$. This property implies comparison principles, used in the proof of convergence of the numerical method [Mir18]. In addition, degenerate ellipticity implies a monotony property of the local update operator Λ, see Proposition A.4 in [Mir19], implemented in Algorithm 3 below. As a result, the sequence of approximate solutions $(u_n)_{n \geq 0}$, $u_n : X_h \to [0, \infty]$, produced along the iterations of our numerical method are pointwise non-increasing.*

2 Implementation

We describe the implementation of our massively parallel solver of generalized eikonal PDEs, discretized under the form (6). 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 (6) 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$(u, next, b, p)$
- $active \Leftarrow next$; $next \Leftarrow 0$.

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* 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[4] library.

The discretization domain X_h, which is a three dimensional cartesian grid (5), 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

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

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

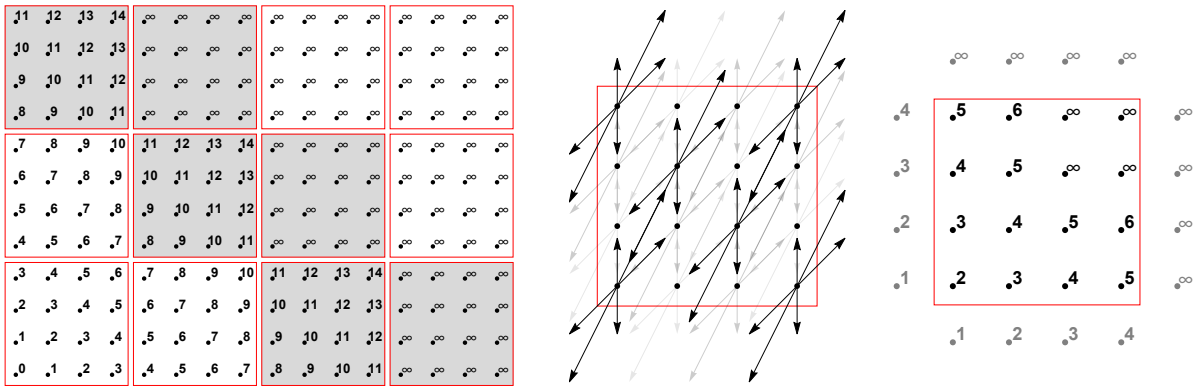

Figure 3: *Left:* Decomposition of the Cartesian grid X_h into tiles X_h^b, with block index $b \in B_h$. Grayed blocks are tagged *active*. *Center:* Updating a block $b \in B_h$ requires loading the unknown values $u : X_h \to \mathbb{R}$, both within X_h^b and at some neighbor points. *Right:* Several local updates are performed within a block (two here).

§2.2 and Table 1. 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 $active : B_h \to \{0,1\}$ 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 $N_{act} = \#\{b \in B_h;\ active[b]\}$ 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 typically expects that $N_{act} \approx N_h^{2/3}$ (in d-dimensions, $N_{act}^b \approx N_h^{1-1/d}$).

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 $\mathcal{O}\big(N_{act} N_b RK(I+J)\big)$ where R is the number of inner loops in Algorithm 2 and I, J, K are the scheme parameters (6). A second boolean table $next : B_h \to \{0,1\}$, is used to mark which blocks 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 Remark 1.4, which guarantees that the values of $(u_n)_{n\geq 0}$ of the approximation 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, implemented by "muting" the threads of the inactive branch) should be avoided for best efficiency.*

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.

A 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 stored by in 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].

The finite difference scheme (6) 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 of the point, then the precomputed stencils have a negligible memory usage, and these precomputations are used. 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 problem data. (Stencils are defined by $N = K(I + J)$ scalars and offsets per grid

point, see (6), 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 not cripple 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 (6) 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 Remark 1.4, 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, if appropriate, the block b and its immediate neighbors b' need to be tagged for update in the next iteration of the eikonal solver Algorithm 1, 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, it is customary to define $\Lambda u(p)$ as the solution to equation $Fu(p) = \rho(p)$ w.r.t the variable $u(p)$, regarding all neighbor values as constants, see [Mir19, Appendix A]. We prove in this subsection that Algorithm 3 does compute this value, and comment on its numerical complexity and efficient implementation. A closely related method is used in the update step of the standard fast marching method for isotropic eikonal equations [Set96], whose discretization is a special case of (6).

For simplicity, and consistently with the presentation of Algorithm 3, we assume a numerical

scheme of the form

$$(Fu(x))^2 := h^{-2} \sum_{i=1}^{I} \alpha_i \big(u(x) - u(x+he_i)\big)_+^2, \tag{7}$$

in other words $J = 0$ and $K = 1$ in (6). Denote $u_i := u(x + he_i)$ for all $1 \leq i \leq I$, and let $\rho := \rho(p)$. The update value $\lambda = \Lambda u(x)$ is by construction the unique root $\lambda \in \mathbb{R}$ of

$$f(\lambda) := \sum_{1 \leq i \leq I} \alpha_i (\lambda - u_i)_+^2 - h^2 \rho^2, \tag{8}$$

where $a_+ := \max\{0, a\}$. Note that $f(\lambda) = -h^2\rho^2 < 0$ on $]-\infty, \lambda_*]$ where $\lambda_* := \min\{u_i; 1 \leq i \leq I\}$, that f increasing on $[\lambda_*, \infty[$, and that $f(\lambda) \to \infty$ as $\lambda \to \infty$. Thus f does indeed admit a unique root λ, by the intermediate value theorem.

The numerical solution of the non-linear equation (8) takes advantage of its piecewise quadratic structure. For that purpose, introduce a permutation $i_1, \cdots, i_I$ of $\{1, \cdots, I\}$ such that $u_{i_1} \leq \cdots \leq u_{i_I}$. Then for any $1 \leq r \leq I$ on has

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

with the abuse of notations $[u_I, u_{I+1}] := [u_I, \infty[$. The coefficients of this quadratic function are

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

Algorithm 3 solves the quadratic equations $a_r\lambda^2 - 2b_r\lambda + c_r = 0$ successively, for increasing values of $1 \leq r \leq I$. Only the largest of the two quadratic roots is relevant, denoted $\lambda_r := (b_r + \sqrt{b_r^2 - a_r c_r})/a_r$, and it is returned if $\lambda_r \in [u_{i_r}, u_{i_{r+1}}]$, at some rank denoted $r = r_*$. By construction, the root λ_r exists and is real for all $1 \leq r \leq r_*$, since $a_r > 0$ and $f(u_{i_r}) < 0$.

From the implementation standpoint, some attention must be paid to the sorting step, especially in the Euler-Mumford case where $I = 30$ neighbor values are used. Indeed, a naive bubble sort has a complexity $\mathcal{O}(I^2)$ which dominates the rest of the computations and severely slows down the numerical method in that case. Best results were obtained applying a network sort [Knu98] (an efficient branchless sorting method) to the 15 first (resp. last) values, followed by a merge operation.

In the case of a general scheme (6), where I, J, K are arbitrary, the update $\lambda = \Lambda u(p)$ is by construction the unique root of $f : \mathbb{R} \to \mathbb{R}$ defined by

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

and where $u_{ik} := u(p + he_{ik})$ and $u'_{jk} := \min\{u(p - hf_{jk}),\, u(p + hf_{jk})\}$. For each $1 \leq k \leq K$, the unique root $\lambda^{(k)}$ of f_k is computed as for (8), by grouping the two sums defining f_k into a single one over $1 \leq l \leq I + J$. The root of f is $\lambda = \min_{1 \leq k \leq K} \lambda^{(k)}$.

The numerical cost of the local update operator Λ is roughly[5] proportional to the number $N_{fd} := K(I + J)$ of terms involved in the finite difference scheme (6). The variations of N_{fd} among models lead to interesting compromises in the choice of the number R of iterations in Algorithm 2 and of the tile size, see Table 1.

[5] Strictly speaking the complexity is $\mathcal{O}\big(K(I+J)\ln(I+J)\big)$, due to the sorting step.

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

Table 1: Number $N_{fd} = K(I + J)$ of finite differences terms in (6) 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 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×

Exp.	model	GPU(s)	CPU(s)	ratio
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.

3 Numerical experiments

We illustrate our numerical solver of curvature penalized shortest paths, in (mostly) synthetic experiments, related to a variety of contexts ranging from motion planning with obstacles or drift, to image segmentation, and the configuration of radar systems. Some of the test cases are new, whereas others reproduce or are strongly inspired by previous works [CMC16, CMC17, DDBM19, DMMP18, MD17, Mir18] which used an earlier CPU implementation.

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 of obstacles or slow regions in the test case, and their layout, as noted in [WDB$^+$08]. This in contrast with the fast-marching-like method [Mir18] implemented on the CPU, which is guaranteed update each discretization point at most $N_{fd} = K(I + J)$ times where I, J, K are the scheme parameters (6) (for this reason, slightly abusively, fast-marching is referred to as a single pass method), and whose complexity $\mathcal{O}(N_{fd} N \ln N)$ is independent of the specific test case, where N is the total number of discretization points. However, fast-marching is limited in speed by its sequential nature.

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 its ideal case.

2. *Fastest exit from a building.* Illustrates the implementation of walls and thin obstacles, 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, see (1) and (3), 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. The Reeds-Shepp car can move both forward and backward, and reverse gear along its path, which is evidenced by *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 be observed at the endpoints of trajectories displayed [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 (4) is discretized on a $300 \times 300 \times 96$ Cartesian grid, following (6), thus producing a coupled system of equations featuring 8.6 million unknowns[6]. Computation time for the GPU eikonal solver ranges from 0.28s (Reeds-Shepp forward) to 1.54s (Euler-Mumford elastica), reflecting (among other things) the complexity of the discretization stencil, see Figure 2. A substantial speedup ranging from 60× to 120× 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. 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

[6]For this simple problem, results visually quite similar can be obtained at a fraction of the cost using a smaller discretization grid, eg. of size $100 \times 100 \times 64$.

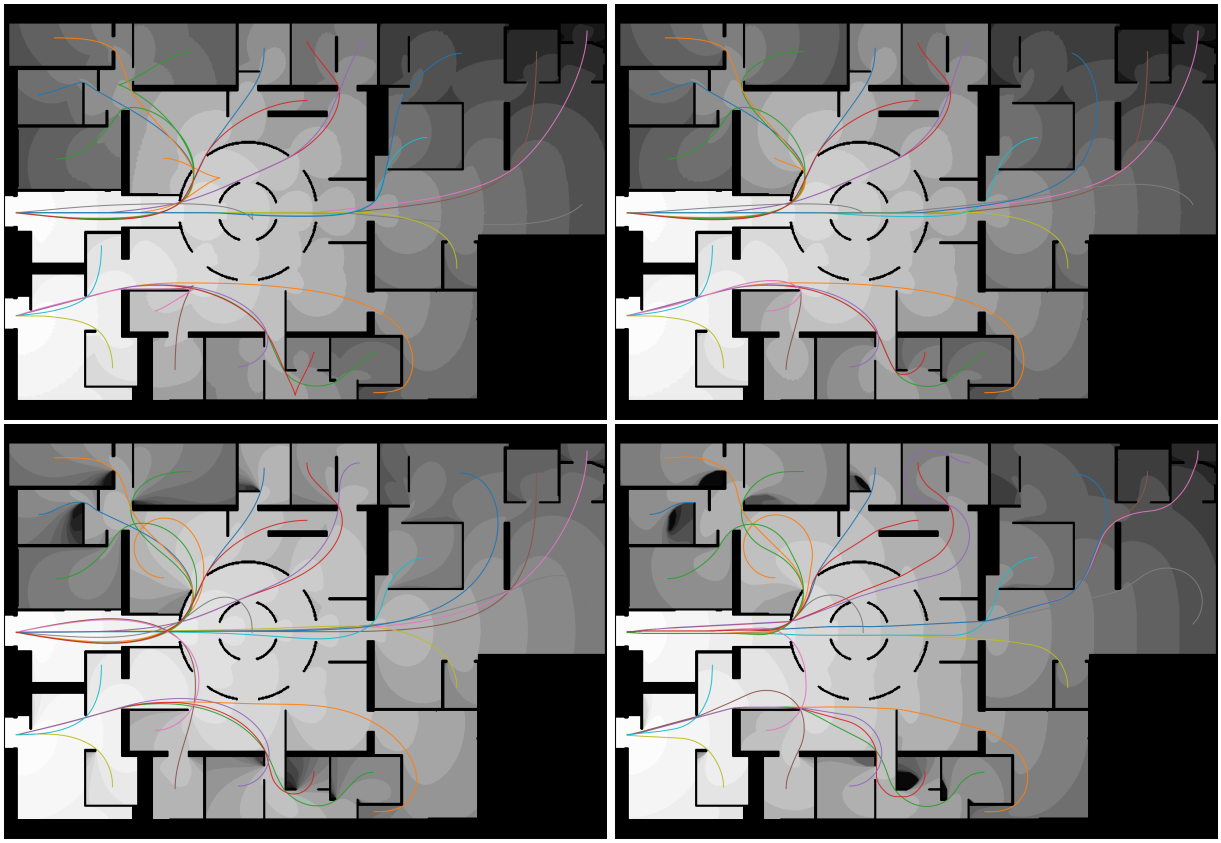

Figure 4: Planar projections of minimal geodesics for the Elastica and Dubins model (left to right). Two seed points at the exits, with horizontal tangents. Geodesics are backtracked from one tip point in each room, with a given but arbitrary tangent.

to bypass most of these intersection tests and limits their impact on computation time. In contrast with [WDB$^+$08], 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 forward and reversible 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. Note that the extracted paths minimize a functional (1) which is unrelated with safety and thus may not be suitable for motion planning, despite being smooth (Euler-Mumford) or having a bounded curvature radius (Dubins). 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 dimension, 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.

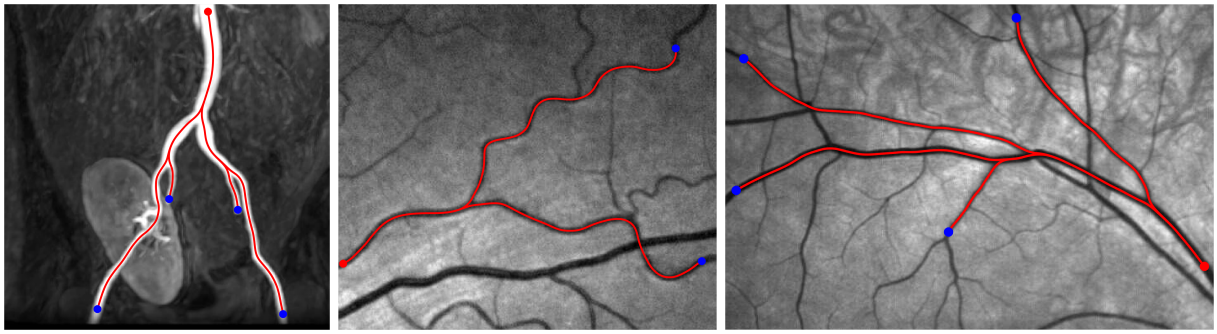

Figure 5: Segmentation of tubular structure centerlines using the Reeds-Shepp forward and Euler-Mumford elastica models, following [CMC17]. Left : Blood vessels in Magnetic Resonance Angiography (MRA) data. Center and right : Blood vessels on an image of the retina.

3.3 Tubular structure segmentation

A popular approach for segmenting tubular structures in medical images, such as blood vessels on the retinal background in this experiment, 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 CPU 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 plays against the GPU solver, which is most efficient in the presence of wide fronts with rather uniform velocity, yet the speedup 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 $V(x) = 0.6 \sin(\pi x_0) \sin(\pi x_1) x / \|x\|$ on the domain $[-1, 1]^2$. Our vehicle has unit speed, and turning radius $\xi = 0.3$.

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

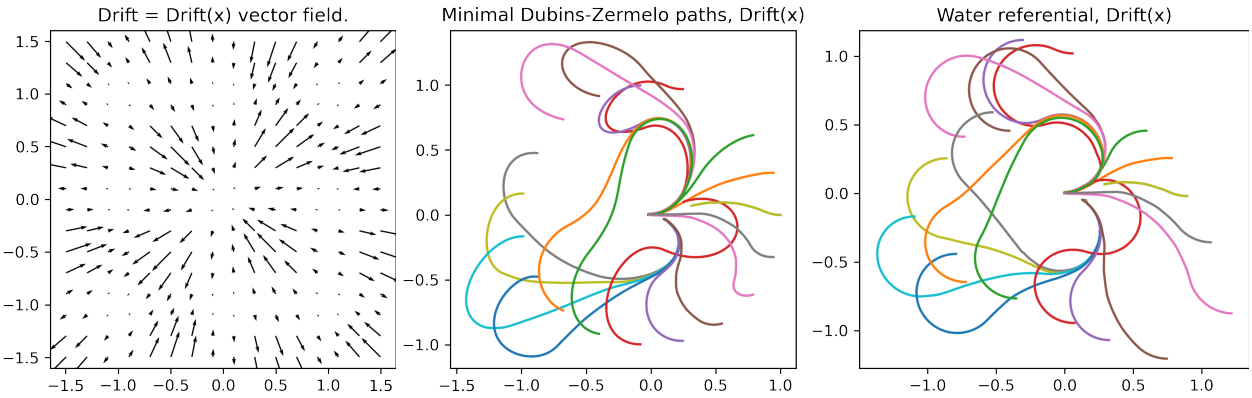

Figure 6: Illustration of the Dubins-Zermelo-Markov problem. Let drift velocity (water current). Center shortest path, between the seed point $(0, 0)$ with horizontal tangent, and other seed points, such that the radius of curvature in the water referential does not exceed the prescribed bound.

scheme (6) 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[7]. The problem takes the generic form

$$\sup_{\xi \in \Xi} \inf_{\gamma \in \Gamma} \mathcal{E}(\xi; \gamma), \tag{9}$$

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 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

[7] 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].

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 form

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

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. 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 following configuration parameters, gathered into the abstract variable $\xi \in \Xi$: the position of the first radar x_1 within a disk, the position of the second one x_2 within a line, and the blind distances r_1, r_2, r_3 subject to $v_i = 0.2/r_i$ for $1 \leq i \leq 3$.

Minimization over the parameter $\gamma \in \Gamma$ in (9) 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 configuration alone $\xi \in \Xi$. We use automatic differentiation to differentiate $\mathcal{E}(\xi)$, as described in [MD17], and optimize this quantity via gradient ascent. Using these tools, a *local* maximum of $\mathcal{E}(\xi)$ is reached in a dozen iterations approximately. 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

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 $30\times$ to $120\times$ 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 other classes of generalized eikonal equations, 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.

[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.

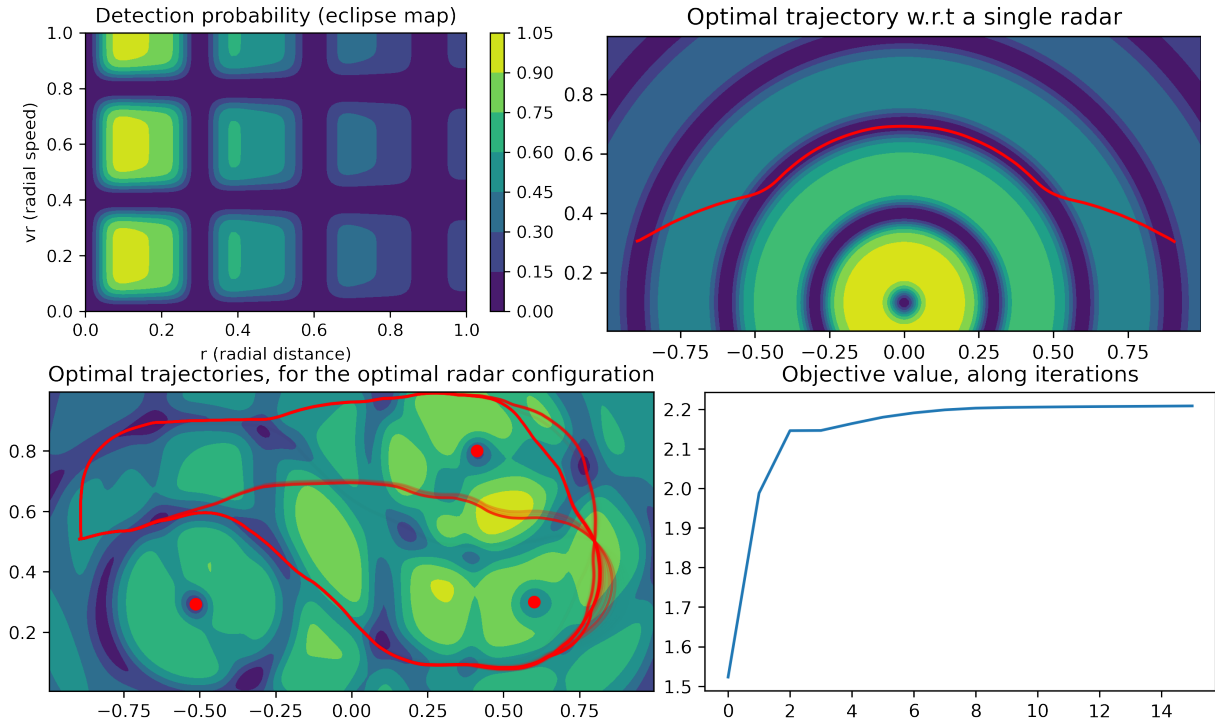

Figure 7: (Top left) Detection probability of a vehicle by a radar, depending on the radial distance and radial velocity. Note the blind distance and blind velocity periods. (Top right) Trajectory minimizing detection between two points in the presence of a single radar. It is a concatenation of circles, 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.

[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.

[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.

[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.

[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.

[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.

[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.