Landmark-based Geodesic Computation for Heuristically Driven Path Planning

Gabriel Peyré
CMAP, UMR CNRS 7641,
École Polytechnique, 91128 Palaiseau, France
gabriel.peyre@polytechnique.fr

Laurent D. Cohen
CEREMADE, UMR CNRS 7534,
Université Paris Dauphine, 75775 Paris, France
cohen@ceremade.dauphine.fr

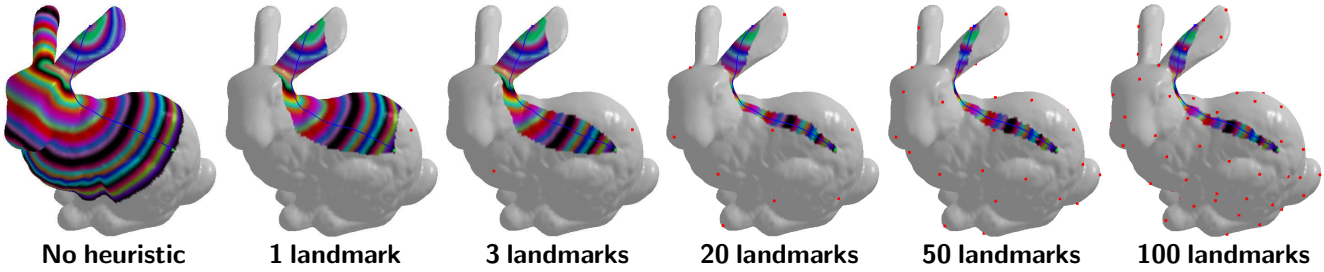


Fig. 1. *In order to compute the geodesic path between two points on the surface, compution is needed on a large part of the surface when using classical fast marching (left). When the distance map to a set of landmarks is pre-computed, and the propagation is heuristically driven by these maps, only the colored region is explored, and it becomes smaller and smaller as the number of landmarks increases.*

Abstract

This paper presents a new method to quickly extract geodesic paths on images and 3D meshes. We use a heuristic to drive the front propagation procedure of the classical Fast Marching. This results in a modification of the Fast Marching algorithm that is similar to the A^* algorithm used in artificial intelligence. In order to find very quickly geodesic paths between any given couples of points, we advocate for the initial computation of distance maps to a set of landmark points and make use of these distance maps through a relevant heuristic. We show that our method brings a large speed up for large scale applications that require the extraction of geodesics on images and 3D meshes. We introduce two distortion metrics in order to find an optimal seeding of landmark points for the targeted applications. We also propose a compression scheme to reduce the memory requirement without impacting the quality of the extracted paths.

1. Previous works

The extraction of shortest paths is a building block for a large class of applications ranging from graph theory to computer vision. The computation can be carried over a totally discrete setting (such as a graph) or over a discretized domain (such as an image or a 3D mesh).

Graphs and discrete computation. The canonical method to compute shortest paths on graphs is the Dijkstra algorithm, see for instance [4]. Fast exploration strategies have been used to speed up the computation. The A^* algorithm [12] makes use of an heuristic to reduce the search space. Other exploration strategies have been proposed in the field of artificial intelligence, such as such as IDA* [10].

Images and continuous setting. In order to extract geodesics for a continuous metric, the Fast Marching algorithm [15] uses both a discrete scheme for derivation and a front propagation. A similar algorithm was also proposed in [18].

The minimal length properties of geodesics has been applied in computer vision, for example to solve global minimization problems for deformable models [3]. The continuous nature of this method is particularly attractive for image-based processings, for instance to extract tubular structures and centerlines in 3D medical data [5].

Path planning: from discrete to continuous. Discrete computation on graphs give rise to numerous schemes to perform motion planification and the A* algorithm is intensively used for path-finding in video games [17]. For the case of an Euclidean metric, faster algorithms have been devised that exploit specific data structures such as visibility graphs [14].

The Fast Marching algorithm can be used to extract paths with a non Euclidean metric [15]. The authors of [11] compare the Fast Marching and the A* algorithms to perform motion planification, but they do not propose a heuristic modification of the Fast Marching.

2. Shortest Path: Continuous and Discrete Algorithms

In this section we expose an unifying framework which includes the Fast Marching [15], Dijkstra [4] and A^* [12] algorithms together with our new heuristically driven propagation.

2.1. Front Propagation Methods for Shortest Path

We work on a discrete set of points and for each point x we have access to its neighbors y, defining the relation $y \sim x$. These points can be embedded on a discretization grid (for the Fast Marching and for our method) or can be the vertices of a graph (for Dijkstra and A*). Our goal is to compute the distance function $U(x) = d(x_1, x)$ to some starting point x_1.

Initialization:
- Alive set: the starting point x_0;
- Trial set: the neighbors of x_0;
- Far: the set of all other grid points.

Loop:
- Let x be the Trial point with the smallest priority $\mathcal{P}(x)$;
- Move it from the Trial to the Alive set;
- For each neighbor y of the current point x:
  - if y is Far, then add it to Alive and compute a new value for $U(y)$,
  - if y is Alive, recompute the value $U(y)$, and update it if the new value is smaller,
  - recompute the priority $\mathcal{P}(y)$.
- If the end point $x = x_1$ is reached, stop the algorithm.

Table 1: Pseudo-code for the common framework for front propagation.

The propagation methods label the points during the computation according to:
- *Alive* is the set of points at which the distance value U has been computed and will not change;
- *Trial* is the set of next grid points to be examined and for which an estimate of U has been computed;
- *Far* is the set of all other grid points, for which there is not yet an estimate for U.

Table 1 shows the main steps of the algorithms. Each algorithm must implement the following sub-functions
- A way to update the value $U(y)$ at a given Trial point y. This computation uses the values of U at the adjacent locations. This computation depends on the metric used, which can be defined on a graph (for discrete methods) or on the whole space (for continuous methods). We explain in sections 2.2 and 2.3 specific instantiations for the Dijkstra and Fast Marching algorithms.
- A priority map $\mathcal{P}$ orders the set of Alive points according to some computational criterion. In the Fast Marching and Dijkstra algorithm, $\mathcal{P}(x) = U(x)$ is the current distance to the starting point. In our heuristical front propagation as in the A* algorithm, $\mathcal{P}(x)$ is chosen to minimize the number of visited points. We explain in section 4 how to actually construct a priority function $\mathcal{P}$ that makes use of a heuristic.

2.2. Discrete Case: Dijkstra

In the discrete setting, a symmetric weight $g(x, y)$ is used to define the distance between two adjacent points $x \sim y$ of the graph. The length of a path $v = [v_1, \ldots, v_m]$, of $\ell(v) = m$ adjacent points v_i is $L(v) = \sum g(v_i, v_{i+1})$, and we define the distance between two vertices

$$d(x_0, x_1) = \min_{m,v} \{L(v) \setminus \ell(v) = m, v_1 = x_0, v_m = x_1\}.$$

The distance $U(x)$ at a vertex x in the alive set is updated during the propagation according to

$$U(x) = \min_{y \sim x} (U(y) + g(x, y)).$$

The shortest path v from x_0 to x_1 is tracked backward using

$$v_0 = x_1 \quad \text{and} \quad v_{i+1} = \operatorname*{argmin}_{y \sim v_i} U(y).$$

2.3. Continuous Case: Fast Marching

In $\mathbb{R}^d$, we are given a potential function $g(x) > 0$, and the weighted geodesic distance between two points $x_0,\ x_1 \in \mathbb{R}^d$, is defined as

$$d(x_0,\ x_1) = \min_{\gamma} \left(\int_0^1 \|\gamma'(t)\|\, g(\gamma(t)) \mathrm{d}t \right), \qquad (1)$$

where γ is a piecewise regular curve with $\gamma(0) = x_0$ and $\gamma(1) = x_1$. When $g = 1$, the integral in (1) corresponds to the length of the curve γ and d is the classical Euclidean distance.

The Fast Marching uses the fact that the function U satisfies the nonlinear *Eikonal* equation:

$$\|\nabla U(x)\| = g(x). \qquad (2)$$

The distance $U(x) = u$ at a point $x = x_{i,j}$ in the trial set is updated during the propagation according to the solution of

$$\max(u - U(x_{i-1,j}),\ u - U(x_{i+1,j}), 0)^2 + \max(u - U(x_{i,j-1}),\ u - U(x_{i,j+1}), 0)^2 = h^2 g(x_{i,j})^2.$$

This is a second order equation (the equation is written in $\mathbb{R}^2$ for simplicity) and it can be solved as detailed for example in [2].

The geodesic curve γ from x_0 to x_1 can be computed by extracting the parametric curve $C(t)$ that solves the back propagation equation:

$$\frac{dC}{dt} = -\overrightarrow{\nabla U} \qquad \text{with} \qquad C(0) = x_1.$$

This gradient descent is a local computation, and it only uses the value of U for a small fraction of the visited grid points. Note that these grid points are those located in the Alive set at the end of the front propagation procedure.

3. Heuristically Driven Front Propagation

In this section we explain our algorithm in the 2D setting, and show some numerical results that illustrate the main features of this method.

3.1. Propagation with a Heuristic

In order to minimize the number of Alive points at the end of the front propagation procedure, one should use a priority function $\mathcal{P}$ that tries to advance the front toward the goal point x_1. In order to do so, we assume that, together with the the current weighted distance to the start point $U(x) = d(x_0, x)$, we have an estimate of the remaining weighted distance $V(x) \approx d(x_1, x)$. Our heuristical front propagation algorithm follows the implementation of listing 1 with a priority map

$$\mathcal{P}(x) = U(x) + V(x). \tag{3}$$

The rationale behind the definition of $\mathcal{P}$ is that $d(x_0, x) + d(x_1, x)$ is minimal and constant along the geodesic path joining x_0 and x_1, see [9].

Algorithm A*. For discrete graphs, it has been shown that if the heuristic satisfies $V(x) \leqslant d(x_1, x)$, then the extracted geodesic is a path of minimum length. This leads to the A* algorithm of [12]. Various strategies have been proposed to devise admissible heuristics, see [8] and the references therein.

Heuristically driven Fast Marching On figure 2, one can see a front propagation, where we have used the oracle heuristic $V(x) = \lambda\, d(x_1, x)$, with a parameter $\lambda \in [0, 1]$. The value $\lambda = 0$ corresponds to the classical Fast Marching propagation, which results in a large region of Alive points (colored in red). However, as we increase the value of λ toward 1, the explored region shrinks around the geodesic path that links x_0 to x_1.

There are however two important issues with this ordering of the Trial set:

- This ordering can break the monotone condition that is required by the Fast Marching algorithm to produce a valid approximation of the continuous underlying distance function. We show in the numerical results presented in sub-section 4.2 that the geometric error on the extracted geodesic remains low.
- We do not have an immediate access to the remaining distance $d(x, x_1)$, since it would involve performing another front propagation from x_1. We explain in section 4 how to overcome this problem.

3.2. Evaluation of an Heuristic

We cast the problem of finding a good heuristic into the problem of approximating the distance function $d(x, y)$ between two points (x, y) by some function $\widetilde{d}$. We define the heuristic using

$$V(x) = \widetilde{d}(x_1, x) \approx d(x_1, x).$$

This approximated distance $\widetilde{d}$ should satisfies $\widetilde{d} \leqslant d$ in order not to deviate the propagation from the true shortest path (computed with $V = 0$). It must be fast to evaluate and can use a reasonable amount of pre-computed data.

To evaluate the quality of the heuristic, we propose two metrics

- The approximation metric

$$E_1(d, \widetilde{d}) = \iint |d(x, y) - \widetilde{d}(x, y)|^2 \mathrm{d}x\, \mathrm{d}y,$$

  is approximated during evaluation using a finite number of precomputed distance maps $d(p_i, x)$ to a set of points $\{p_i\}_{i=1}^m$ chosen at random

$$E_1(d, \widetilde{d}) = \frac{1}{m} \sum_{i=1}^{m} \int |d(p_i, y) - \widetilde{d}(p_i, y)|^2 \mathrm{d}y.$$

- The computation gain metric measures the usefulness of the heuristic for extracting geodesics between a set of couples of points $\{(x_0^i, x_1^i)\}_i$. For each front propagation from x_0^i to x_1^i, we measure the area $\mathcal{A}_i(\widetilde{d})$ of the Alive set at the end of the propagation. We also measure the area $\mathcal{A}_i(0)$ of the Alive set for a propagation without heuristic. We define the computation gain metric to be

$$E_2(d, \widetilde{d}) = \frac{1}{m} \sum_{i=1}^{m} \mathcal{A}_i(\widetilde{d}) / \mathcal{A}_i(0).$$

When the geodesics queries are random, these two metrics tends to gives the same evaluation. However, the computation-gain metric is able to better adapt to typical applications where queries can be be highly non-uniform (for instance in road or tubular structure extraction).

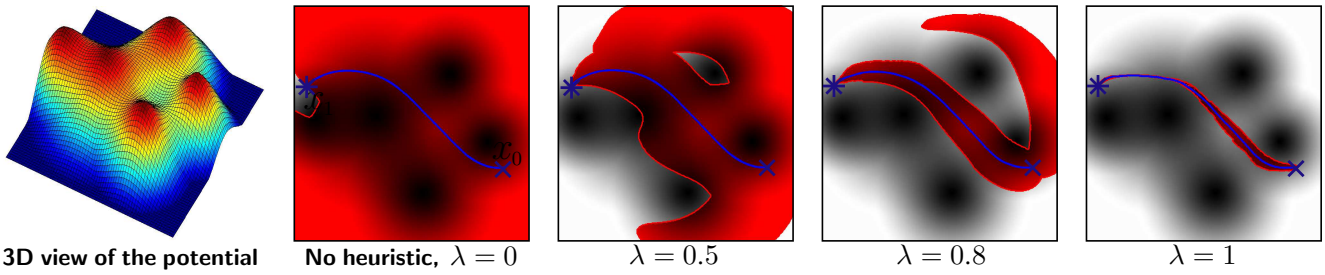


Figure 2. *An example of 2D path planning. The set of alive points according to increasing heuristic is shown in red.*

4. Landmark-based Front Propagation

In this section, we describe a particular implementation of the heuristic. This heuristic is computed using an approximated distance $\widetilde{d}$ evaluated with a set of pre-computed distances to landmark points.

4.1. Landmark-based heuristic

A new method for distance evaluation on graph has been introduced in [8] as an admissible heuristic for the A* algorithm. We explain why this method can be useful for our heuristically driven Fast Marching and give a quantitative numerical study.

This method exploits the triangular inequality, since we have, for every couple of points x et y,

$$d(x,y) = \sup_z \Big(|d(x,z) - d(z,y)| \Big).$$

This equality is still valid in the continuous framework of the Fast Marching, and in order to derive a useful heuristic, one can chose a small set $z_1, \dots, z_n$ of Landmark points. The set of distance maps $d_k(x) = d(z_k, x)$ is pre-computed once for all, and we define the approximation

$$\widetilde{d}_{z_1 \dots z_n}(x,y) = \sup_{k=1\dots n} \Big(|d_k(x) - d_k(y)| \Big). \qquad (4)$$

In the following, we drop the $z_1, \dots, z_n$ dependencies and call the approximated distance $\widetilde{d}$. We note that this approximation always satisfies the condition $\widetilde{d} \leqslant d$.

Figure 3 give an intuitive explanation of the efficiency of this approximation. In the ideal case, the geodesic γ_k joining a landmark z_k to x also pass through y. In this case, we have $d(x,y) = \widetilde{d}(x,y)$ and there is no approximation. But in most of the cases, this is not true, but a geodesic γ_k passes close to y and $\widetilde{d}$ is indeed a good approximation of the real distance.

4.2. Landmark-based Path Planing

This landmark-based propagation can be used to speed-up the computation of most path planing application, including path finding in video games or robot path planing.

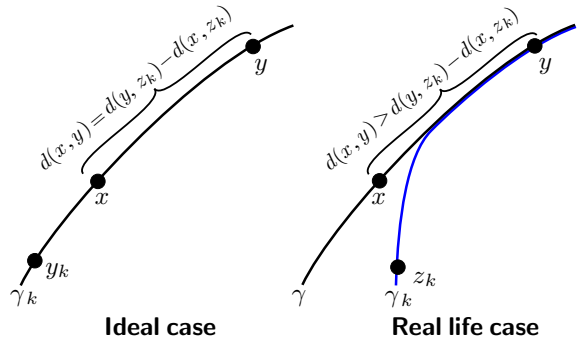


Figure 3. *Justification of the approximation properties of $\widetilde{d}$.*

On figure 4, one can see the active region explored by the propagation algorithm for an increasing number of landmark points. These points are chosen at random on the 2D image. The explored region progressively shrinks toward the central extracted curve when we add more points, since the heuristic is becoming more accurate.

On figure 5, one can see a numerical evaluation of the precision of the extracted geodesic. It reports the distortion $\|\gamma - \widetilde{\gamma}\|_{\mathcal{H}}$ between the original geodesic γ (computed without heuristic) and $\widetilde{\gamma}$ the one extracted with a heuristic. We use the symmetric mean-square Hausdorff metric

$$\|\gamma - \widetilde{\gamma}\|^2_{\mathcal{H}} = \frac{1}{2}\Big(\int_{\gamma} \min_{y \in \widetilde{\gamma}} \|x-y\|_2^2 \mathrm{d}x + \int_{\widetilde{\gamma}} \min_{x \in \gamma} \|x-y\|_2^2 \mathrm{d}x \Big),$$

since this captures well the geometric distortion caused by our partial propagation.

On 2D maps containing important curvilinear features (such as the road on the example on the left), the distortion caused by the heuristic is nearly not noticeable. On the contrary, for relatively flat maps (such as the one depicted on the right) the distortion can be relatively high (of the order of a few pixels for a map of 256×256 pixels). This is because the salient features of this map catch the geodesic and avoid large lateral moves when the front propagation is modified.

In order to improve the quality of the heuristic and thus the resulting speed-up, one has to carefully seed the landmarks across the image. In the next section we set-up a complete framework for the evaluation of this sampling, together with strategies to get a high-quality seeding of the base points.

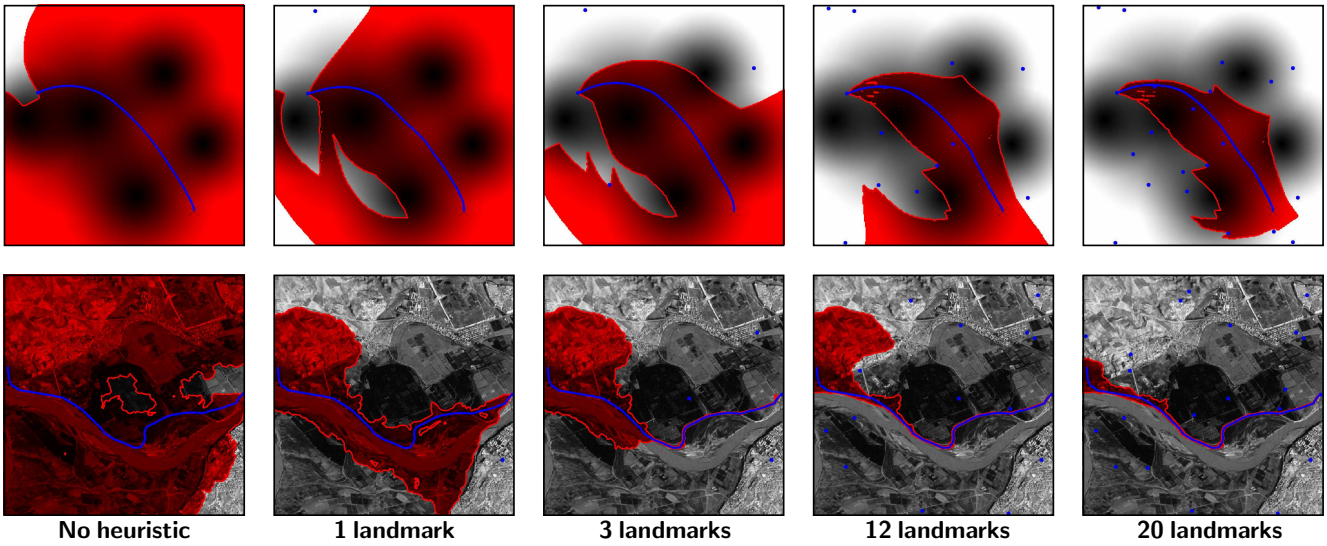


Figure 4. *Example of propagation using a landmark-based heuristic with random seeding of base points.*

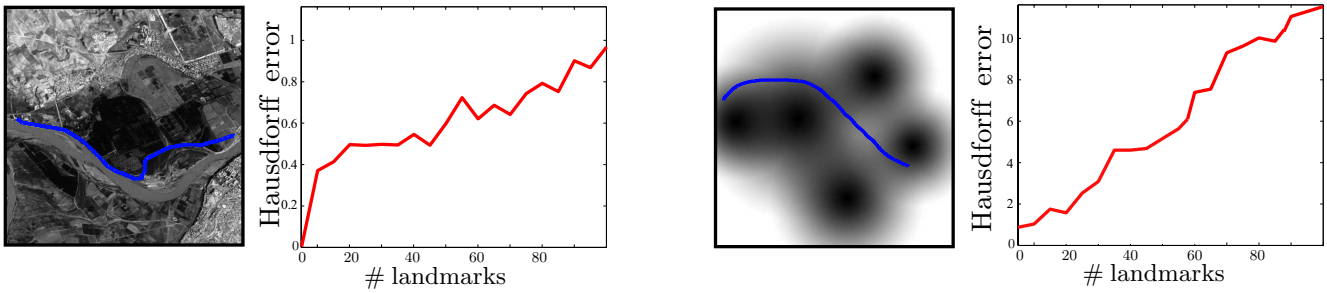


Figure 5. *Distortion caused by our heuristic propagation on the extracted geodesics. The error is reported using the Hausdorff distance expressed in pixels (the size of the image is* 256×256*).*

4.3. Seeding strategies for the landmarks

The quality of a sampling can be measured using the previously introduced metrics

$$E_i(z_1, \ldots, z_n) = E_i(d, \widetilde{d}_{z_1 \ldots z_n}) \quad \text{for} \quad i = 1, 2.$$

In order to find sampling locations $\{z_1, \ldots, z_n\}$ for the landmarks, we use various strategies, among which :

(a) A manual sampling that exploits specific knowledge of the 2D map g. This semi-automatic method is not studied in this paper.

(b) A random sampling in the image.

(c) A uniform sampling according to the distance d. This can be accomplished using the farthest point sampling procedure proposed in [13], where one chooses

$$z_{n+1} = \operatorname*{argmax}_{z} (\operatorname*{argmin}_{1 \leqslant k \leqslant n} d(z_k, z)).$$

(d) A uniform seeding on the boundary of the domain.

(e) A greedy strategy minimizing the E_1 metric. We define recursively the locations

$$z_{n+1} = \operatorname*{argmin}_{z \in R} E_1(z_1, \ldots, z_n, z),$$

where R is some small set of random candidates.

(f) Same as (e) but using E_2 instead of E_1.

Distance approximation evaluation. A good seeding strategy should be able to put a landmark before any couple of points in the domain, near the geodesic path connecting these points. For a constant (i.e. Euclidean) metric, (d) is thus the optimal sampling scheme. However, for complex metric, this is no more the case.

On figure 6, one can see the approximated distance $\widetilde{d}(x_1, x)$ to a target point x_1, computed for $n = 32$ landmarks using various seeding strategies.

On figure 7, one can see a quantitative plot of the approximation error $E_1(x_1, \ldots, x_n)$. The error-driven seeding strategy (e) clearly performs the best as expected by definition. For a smooth potential g (left) the seeding strategy on the boundary (d) performs well, but it fails to capture the topology of complex 2D maps (right).

Computation-saving evaluation. In order to evaluate our algorithm in a real setting, we have set-up a complete framework using 500 typical queries $\{(x_0^i, x_1^i)\}_i$ for various 2D

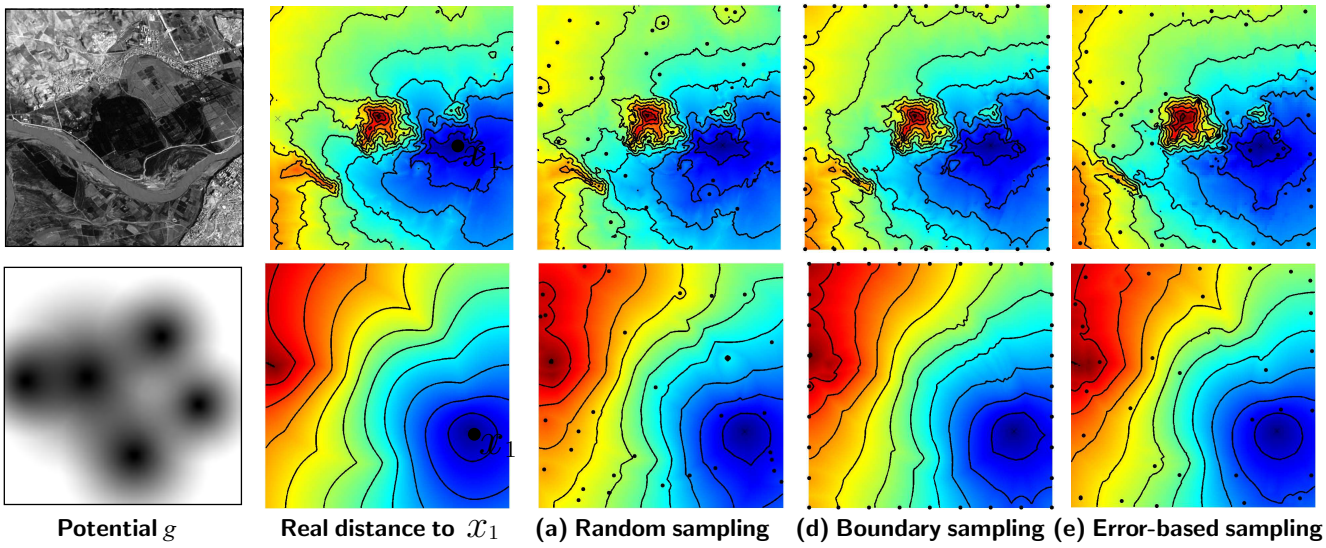


Figure 6. *Graphical display of the approximated distance $\widetilde{d}(x_1, x)$ for various seeding strategies.*

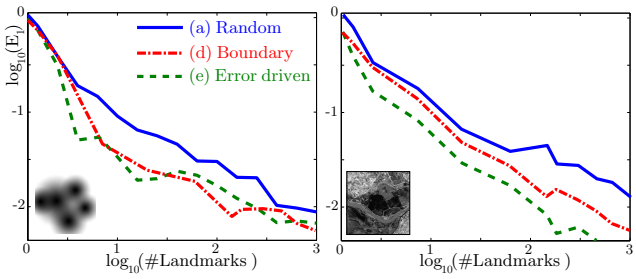


Figure 7. *Decreasing of the approximation error $E_1(z_1, \ldots, z_n)$ for various seeding strategies.*

maps. Some resulting shortest paths queries are shown on the left of figure 8, they are relevant for many standard applications such as path finding in video games or robot path planing.

On the right of figure 8, we have shown the decreasing of the computation-saving metric $E_2(x_1, \ldots, x_n)$ for several sampling strategies. For the greedy sampling strategy (f), we have used another set of 100 typical queries, in order not to bias the computation of the error E_2.

One can see that the computation-driven sampling strategy (f) clearly outperforms the other strategies. In particular, the approximation-based strategy (e) does not give good results, mainly because typical paths are not correlated to areas where the approximation error $\|d - \widetilde{d}\|$ is large.

5. Reducing Memory Usage

5.1. Cells Representation

Classical methods, such as using an octree data structure, can be used to reduce the memory usage of level set algorithms, for example in order to perform image segmentation [6].

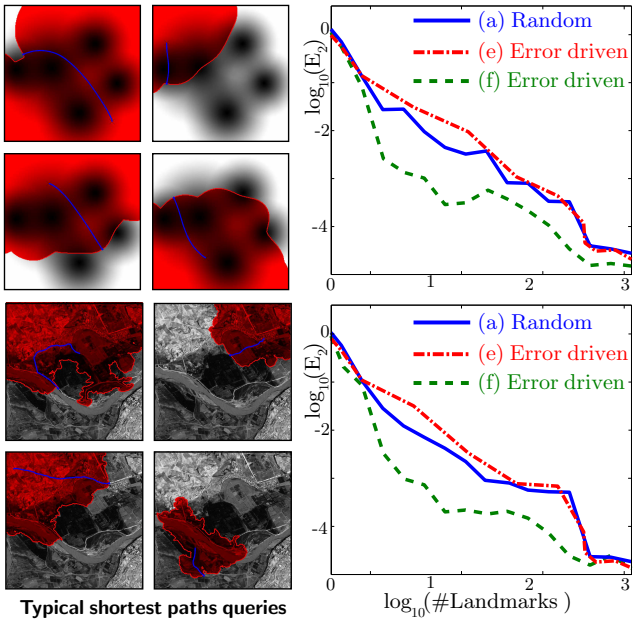


Figure 8. *Computation speed-up $E_2(z_1, \ldots, z_n)$ obtained with a various number of landmarks. The areas colored in red correspond to the explored region without heuristic.*

We choose to implement a simple data structure to reduce the memory usage by allocating the grid cell on the fly during the propagation. A typical cell data structure, for 2D computation, is:

```
struct cell {
  // current geodesic distance
  double distance;
  // either alive, trial or far
  char state;
  // pointers to the 4 neighbors
  cell* neighbors[4]; };
```

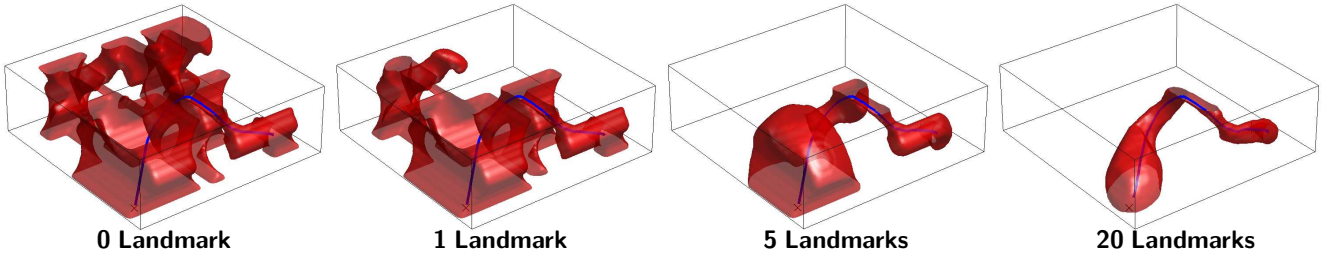
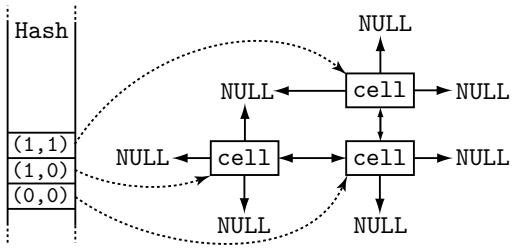


Figure 9. *Explored area for constrained path planning.*

To be able to retrieve a given cell in constant time, we also store the list of allocated cells in a hash table. This is important because when a new cell is allocated, we need to connect it to the existing cells.



This pointer-based representation of the neighboring relation is convenient to extract the geodesic with a gradient descent. There is some memory overhead due to the explicit storage of pointers to neighbors, but the fact that our scheme explores significantly less cells than the classical Fast Marching allows to save much more memory, as shown in next section. The computing time overhead due to the use of a hash table is about 40% in all our tests.

5.2. Computation from Compressed Data

To reduce the storage requirement of the distances to landmark $d_k(x)$, we implement a compression procedure. A typical implementation should

- Allow random access of the value $d_k(x)$, without decompressing the whole data.
- Be asymetric, since we do not care about the compression time, but we need fast access to distance values.
- Give low decompression error, since we need an accurate heuristic and we need to satisfy as much as possible the condition $\widetilde{d} \leqslant d$.

We use a two-steps procedure:

- Differential representation : $A_1(x) = d(x_1, x)$ and

$$A_{k+1}(x) = d(x_{k+1}, x) - d_{x_1 \dots x_k}(x_{k+1}, x),$$

  where we use the approximated distance $\widetilde{d}_{x_1 \dots x_k}$ to give an estimate of $d(x_{k+1}, x)$ using the available k first landmarks. This new representation remove the redundancy that exists between the different distance maps.

- We code each map A_i using a vector quantization scheme [7]. In our tests, we use codeblocks of size 3×3, and we quantize the resulting vectors of $\mathbb{R}^9$ using a codebook of size 256. This results in a memory gain of $36 : 1$ with respect to storing single precision floats.

The constraints we have on the computation from compressed data are similar to the process of rendering from compressed textures [1]. We use the differential coding strategy to cope with the particular redundancy of distances functions. The compression produces neither noticeable artifact nor distortion on the extracted geodesics. The computation time overhead due to decompression is about 15% in CPU time.

6. Other Applications

Our landmark-based propagation can be applied verbatim to higher dimensional propagation and to triangulated surfaces in order to speed-up shortest path queries.

6.1. Constrained Path Planning

Geodesics can be used to compute the path of a robot with various shape and motion constraints [15]. Basically, each additional degree of freedom add a new dimension to the domain in which the front propagation should be performed. Solving such high dimensional problems is time and memory consuming, so the use of a heuristic is highly desirable. In our experiment, the most important issue is the memory used by the full-grid classical Fast Marching, and the memory management strategy exposed in subsection 5 is crucial to scale to complex problems.

On figure 11, one can see two examples of path extractions in 2D with one rotational additional degree of freedom. This results in 3D front propagation, and the corresponding speed function is depicted on the left. Figure 9 shows the influence of the heuristic strength λ on the cells explored by the front propagation.

6.2. Geodesic Extraction on 3D Meshes

The Fast Marching algorithm has been extended to 3D meshes in [16]. Our heuristically driven front propagation extends to 3D surfaces in a straightforward manner. We use a constant metric $g = 1$ and as the surfaces considered does

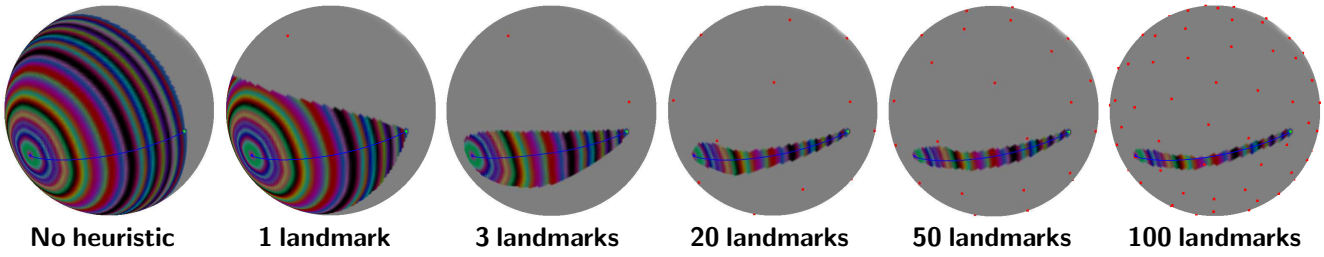


Figure 10. *Heuristically driven front propagation on 3D meshes, with an increasing number of landmark points.*

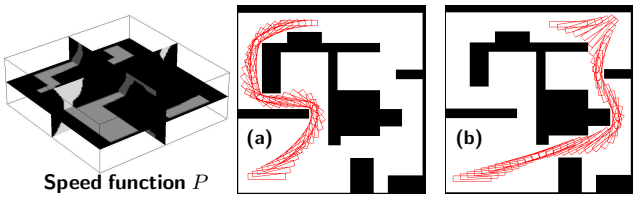


Figure 11. *Examples of constrained path planning.*

not have boundary, the uniform seeding strategy (c) gives the best results for the E_1 metric. On figure 12, one can see the approximated distance $\widetilde{d}(x_1, x)$ for various number of landmarks. On figure 1 and 10, one can see the algorithm in action on various meshes, and for various number of landmark points.

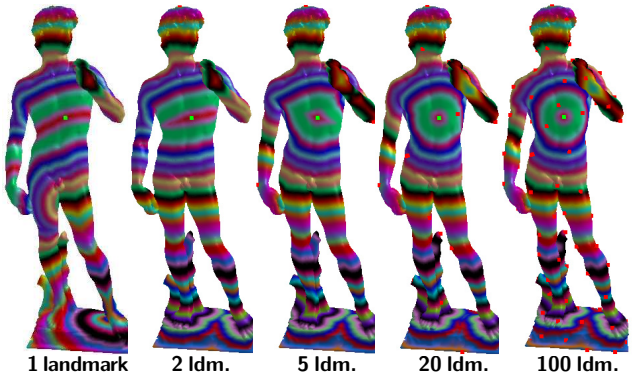


Figure 12. *Approximation of the distance function $d(x_1, x)$ using $\widetilde{d}(x_1, x)$ on a 3D mesh, with an increasing number of landmarks.*

7. Conclusion

In this paper we have presented a simple modification of the Fast Marching to speed up the extraction of geodesics on images, higher dimensional data and triangulated surfaces. This modification takes into account a heuristics computed using a set of distances to landmarks. We reduce the memory requirement of the algorithm using specific data structures and a fast compression scheme.

References

[1] A. C. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. In *Proc. of SIGGRAPH 1996*, pages 373–378, 1996.

[2] L. Cohen. Multiple Contour Finding and Perceptual Grouping Using Minimal Paths. *Journal of Mathematical Imaging and Vision*, 14(3):225–236, May 2001.

[3] L. D. Cohen and R. Kimmel. Global Minimum for Active Contour models: A Minimal Path Approach. *International Journal of Computer Vision*, 24(1):57–78, Aug. 1997.

[4] T. H. Cormen, C. E. Leiserson, and R. R. Rivest. *Introduction to Algorithms*. MIT Press, Cambridge, Massachusetts, 1990.

[5] T. Deschamps and L. Cohen. Fast Extraction of Minimal Paths in 3D Images and Applications to Virtual Endoscopy. *Medical Image Analysis*, 5(4), December 2001.

[6] M. Droske, M. Meyer, M. Rumpf, and C. Schaller. An adaptive level set method for interactive segmentation of intracranial tumors. *Neurosurgical Research*, 27, 2005.

[7] A. Gersho. *Vector Quantization and Signal Compression*. Kluwer Academic Publishers, Boston, 1992.

[8] A. V. Goldberg and C. Harrelson. Computing the shortest path: A* search meets graph theory. *Technical Report MSR-TR-2004-24*, 2004.

[9] R. Kimmel, A. Amir, and A. M. Bruckstein. Finding shortest paths on surfaces using level sets propagation. *IEEE Trans. on PAMI*, 17(6):635–640, 1995.

[10] R. E. Korf. Depth-first iterative-deepening: an optimal admissible tree search. *Artif. Intell.*, 27(1):97–109, 1985.

[11] P. Melchior, B. Orsoni, O. Lavialle, A. Poty, and A. Oustaloup. Consideration of obstacle danger level in path planning using A* and fast-marching optimisation: comparative study. *Signal Processing*, 11(11):2387–2396, Nov. 2003.

[12] N. Nilsson. *Problem-solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[13] G. Peyré and L. D. Cohen. Geodesic Remeshing Using Front Propagation. *Proc. IEEE Variational, Geometric and Level Set Methods 2003*, pages 33–40, Sept. 2003.

[14] H. Rohnert. Shortest paths in the plane with convex polygonal obstacles. *Inf. Process. Lett.*, 23(2):71–76, 1986.

[15] J. Sethian. *Level Sets Methods and Fast Marching Methods*. Cambridge University Press, 2nd edition, 1999.

[16] J. Sethian and R. Kimmel. Computing Geodesic Paths on Manifolds. *Proc. Natl. Acad. Sci.*, 95(15):8431–8435, 1998.

[17] W. B. Stout. Smart moves: Intelligent path-finding. *Game Developer*, oct. 1996.

[18] J. Tsitsiklis. Efficient Algorithms for Globally Optimal Trajectories. *IEEE Trans. on Automatic Control*, 40(9):1528–1538, Step. 1995.