Parallel algorithms for approximation of distance maps on parametric surfaces

Alexander M. Bronstein* Michael M. Bronstein†
Yohai S. Devir‡ Ron Kimmel§
Ofir Weber¶

Department of Computer Science,
Technion–Israel Institute of Technology,
Haifa 32000, Israel.

April 14, 2007

Abstract

We present an efficient $\mathcal{O}(n)$ numerical algorithm for first-order approximation of geodesic distances on parametric surfaces, where n is the number of points on the surface. The structure of our algorithm allows efficient implementation on parallel architectures. Two implementations on a SIMD processor and on a GPU are discussed. Numerical results demonstrate a two order of magnitude improvement in execution time compared to the state-of-the-art algorithms.

1 Introduction

Approximation of geodesic distances on curved surfaces is an important computational geometric problem, appearing in many computer graphics applications. For example, several surface segmentation and editing methods are based on cutting the surface along geodesic paths [1, 2]. Function interpolation on meshes requires the knowledge of geodesic distances, and has numerous uses such as skinning [3] and mesh watermarking [4]. Isometry-invariant shape classification [5, 6, 7, 8], minimum-distortion parametrization [9, 10, 11], and non-rigid correspondence techniques [12] require the matrix

*e-mail: bron@cs.technion.ac.il
†e-mail: mbron@cs.technion.ac.il
‡e-mail: yd@cs.technion.ac.il
§e-mail: ron@cs.technion.ac.il
¶e-mail: weber@cs.technion.ac.il

of all pair-wise geodesic distances on the surface. Other fields where the need to compute geodesic distance maps arises are medical imaging, geophysics[13], and robot motion planning [14] and navigation to mention a few.

The problem of distance map computation can be formulated as the viscosity solution of the *eikonal equation*,

$$\|\nabla t\| = 1, \quad t(S) = 0, \tag{1}$$

where S is a set of source points on the surface. In optics and acoustics, the eikonal equation governs the propagation of waves through a medium. The solution of the eikonal equation demonstrates that light or acoustic waves traverse the path between two points, which takes the least time, a physics law known as *Fermat's principle*.

In [15], Sethian proposed an $\mathcal{O}(n \log n)$ algorithm for first-order approximation of weighted distance maps on domains with weighted Euclidean metric, known as *fast marching*. A similar algorithm based on a different discretization of the eikonal equation was developed independently by Tsitsiklis [16]. The main idea of fast marching is to simulate a wave front advancing from a set of source points S. The propagating front can be thought of as a "prairie fire" evolution towards directions where the grid has not yet been "burnt out". At time $t = 0$, the fire starts at the source points, and the algorithm computes the time values t for each vertex at which the advancing fire front reaches it.

Algorithm 1 outlines the fast marching method. Solution of the eikonal equation starts by setting initial (usually zero) distance to the set of source points S and updating the neighboring points by simulating an advancing wavefront. The algorithm is constructed similar to Dijkstra's algorithn for finding shortest paths in graphs. It maintains a set of fixed vertices S, for which the time of arrival has already been computed, and a priority queue Q of all other vertices sorted by their times of arrival. The basic operation of the fast marching algorithm is the *update* step, which computes the time of arrival of the wavefront to a grid point based on the times of arrival to its neighbor points.

By construction, the updated value cannot be smaller than the values of the supporting vertices. This monotonicity property ensures that the solution always propagates outwards by fixing the vertex with the smallest t. The latter implies that the values of grid points in S vertices are never recomputed. Since the *update step* has constant complexity, the overall complexity of the fast marching algorithm is determined by the procedure that finds the smallest t in the priority queue Q. Heap sorting-based priority queue allows to implement this task in $\mathcal{O}(\log n)$, where n is the number of grid vertices. Since each vertex is removed from Q and inserted to S only once, the overall complexity is $\mathcal{O}(n \log n)$.

Over the last decade, the fast marching algorithm was generalized to arbitrary triangulated surfaces [17], unstructured meshes [18], implicit unorganized surfaces [19], and parametric surfaces [20]. Higher-order versions of fast marching were also proposed [18]. Besides fast marching, there exist other families of numerical algorithms for approximate and exact computation of geodesic distances on surfaces, among which the

most notable one is the Mount-Mitchel-Papadimitriou (MMP) algorithm [21], whose most recent approximate implementation by Surazhsky *et al.* [22] appears to be the fastest distance computation code available in public domain.

In this paper, we explore the problem of geodesic distance map approximation on parametric surfaces, a representation becoming growingly popular as an alternative to unordered triangular meshes [23]. We distinguish between two problems: computation of geodesic distances from an arbitrary source to all vertices on the surface, and computation of the matrix of all pair-wise geodesic distances. The paper is organized as follows. In Section 2, we formulate the eikonal equation on parametric surfaces. Section 3 is dedicated to the update step. We show a compact expression in matrix-vector form for a first-order update step on parametric surfaces based on the planar wavefront model. We show that the scheme is numerically stable, which allows its use with low-precision arithmetics. Section 4 presents a raster scan algorithm for approximate distance map computation on parametric surfaces. The proposed algorithm can be thought of as a generalization of Danielsson's raster scan method [24] to parametric surfaces, or as a raster-scan version of the parametric fast marching algorithm [20]. We show that the raster scan algorithm converges with a bounded number of iterations, which enables its use for geodesic distance map computation. In Section 5, we discuss two parallel implementations of the raster scan algorithm on a SIMD processor and a GPU. Graphics hardware has been previously used for computation of distance maps and Voronoi diagrams on the plane or in the three-dimensional Euclidean space [25, 26, 27, 28, **?**]. However, the use of vector processors for computation of geodesic distance maps is a different and significantly more complex problem, which to the best of our knowledge, has not been yet addressed in the literature. In Section 6, we present numerical tests and performance benchmarks for our algorithms. Parallel raster scan algorithms outperform the state-of-the-art distance computation algorithms by up to two orders of magnitude on commodity hardware, making feasible real-time implementation of many applications, where the complexity of geodesic distance computation has been so far prohibitively high. Section 7 concludes the paper.

2 Eikonal equation on parametric surfaces

In this paper, we focus our attention on parametric two-dimensional manifolds, i.e. surfaces that can be represented by a smooth mapping $\mathbf{x} : \mathbf{U} \to \mathbb{R}^3$, where $\mathbf{U} \subset \mathbb{R}^2$ is a parametrization domain. The topology of $\mathbf{U}$ depends on the topology of the surface. The derivatives

$$\xi_i \quad = \quad \frac{\partial \mathbf{x}}{\partial u^i} \tag{2}$$

with respect to the parametrization coordinates constitute a local system of coordinates on the surface (Figure 1). Distances on the surface are measured according to the differential arclength element,

$$ds^2 \quad = \quad d\mathbf{u}^{\mathrm{T}} \mathbf{G} d\mathbf{u}, \tag{3}$$

Algorithm 1: Fast marching method.

Input: Numerical grid $\mathbf{U}$, set of source points $S \subset \mathbf{U}$ with the corresponding initial values $t(s)$

Output: The distance map $t : \mathbf{U} \mapsto \mathbb{R}^+$.

Initialization

$Q \longleftarrow \emptyset$
foreach *point* $\boldsymbol{u} \in \mathbf{U} \setminus S$ **do** $t(\mathbf{u}) \longleftarrow \infty$
foreach *point* $\boldsymbol{u} \in S$ **do**
    $Q \longleftarrow Q \cup \mathcal{N}(\mathbf{u})$
end

Iteration

while $Q \neq \emptyset$ **do**
    $\mathbf{u} \longleftarrow$ `ExtractMin`(Q)
    $S \longleftarrow S \cup \{\mathbf{u}\}$
    foreach *point* $\boldsymbol{v} \in \mathcal{N}(\boldsymbol{u})$ **do** `Update` $(\mathbf{v})$
end

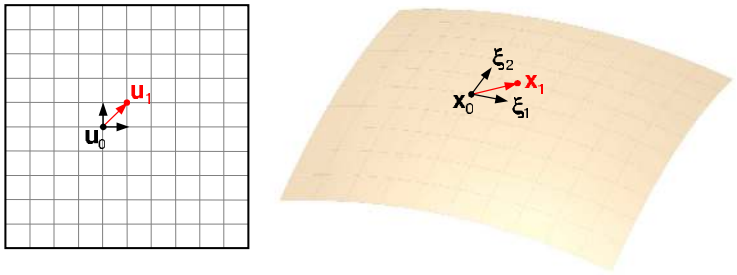

Figure 1: A system of coordinates in the parametrization domain (left) and the corresponding local system of coordinates on the surface (right).

where $d\mathbf{u} = (du^1, du^2)$ and $\mathbf{G}$ is a 2×2 *metric* matrix, whose elements are given by $g_{ij} = \xi_i^{\mathrm{T}} \xi_j$. The local system of coordinates is *orthogonal* if and only if $\mathbf{G}$ is diagonal (note that orthogonality of the coordinate system in the parametrization domain does not imply orthogonality of the coordinate system on the surface).

A distance map on the surface is computed by solving the eikonal equation, expressed in our notation as

$$\|\nabla_{\mathbf{G}} t\|^2 \quad = \quad \nabla_{\mathbf{u}}^{\mathrm{T}} t \; \mathbf{G}(\mathbf{u})^{-1} \nabla_{\mathbf{u}} t \; = \; 1 \tag{4}$$

on a discrete grid obtained by sampling the parametrization domain $\mathbf{U}$. For convenience, we discretize the parametrization domain on a regular Cartesian grid with unit steps. A grid point $\mathbf{u}_0$ is connected to its neighbors $\mathbf{u}_0 + \mathbf{m}$ according some grid connectivity. The simplest grid connectivity is based on four neigbors: $\mathbf{m} =$

$(\pm 1,0)^{\mathrm{T}},(0,\pm 1)^{\mathrm{T}}$. Another possible grid connectivity is the eight-neigbor connectivity, where $\mathbf{m} = (\pm 1,0)^{\mathrm{T}},(0,\pm 1)^{\mathrm{T}},(\pm 1,\pm 1)^{\mathrm{T}}$.

The former two grid connectivity patterns create four and eight triangles, respectively, supporting the grid point $\mathbf{u}_0$. Let us examine a triangle created by $\mathbf{x}_0 = \mathbf{x}(\mathbf{u}_0)$, $\mathbf{x}_1 = \mathbf{x}(\mathbf{u}_0 + \mathbf{m}_1)$, and $\mathbf{x}_2 = \mathbf{x}(\mathbf{u}_0 + \mathbf{m}_2)$; without loss of generality we will henceforth assume that $\mathbf{x}_0 = 0$. In local coordinates, we can write

$$\mathbf{x}_i \quad = \quad \mathbf{x}_0 + m_i^1 \xi_1 + m_i^2 \xi_2, \tag{5}$$

or $\mathbf{X} = \mathbf{TM}$, where $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2)$, $\mathbf{T} = (\xi_1, \xi_2)$, and $\mathbf{M} = (\mathbf{m}_1, \mathbf{m}_2)$. The matrix $\mathbf{E} = \mathbf{M}^{\mathrm{T}}\mathbf{G}\mathbf{M}$ describes the geometry of the triangle. If $e_{12} > 0$ is positive, the angle $\sphericalangle \mathbf{x}_1\mathbf{x}_0\mathbf{x}_2$ on the surface is acute.

3 Update step

The fast marching algorithm can be formulated for parametric surfaces as shown in [20]. All computations are performed on the grid in the parametrization domain, though the distances are computed with respect to the surface metric $\mathbf{G}$. In the numerical core of this algorithm lies the update step, which given a grid point $\mathbf{u}_0$ and the times of arrival of its neighbors, computes the time of arrival $t(\mathbf{u}_0)$. Since $\mathbf{u}_0$ is shared by several triangles (the exact number of triangles depends on the grid connectivity), $t(\mathbf{u}_0)$ is computed in each triangle and the smallest value is selected to update the time of arrival at $\mathbf{u}_0$.

Let $\mathbf{u}_0$ be updated from its two neighbors $\mathbf{u}_1 = \mathbf{u} + \mathbf{m}_1$ and $\mathbf{u}_2 = \mathbf{u}_0 + \mathbf{m}_2$, whose times of arrival are $t_1 = t(\mathbf{u}_0 + \mathbf{m}_1)$ and $t_2 = t(\mathbf{u}_0 + \mathbf{m}_2)$. We denote $\mathbf{x}_i = \mathbf{x}(\mathbf{u}_i)$ and assume without loss of generality that $\mathbf{x}_0 = 0$. Our goal is to compute $t_0 = t(\mathbf{u}_0)$ based on t_1, t_2 and the geometry of the triangle $\mathbf{x}_1\mathbf{x}_0\mathbf{x}_2$. The update of $\mathbf{x}_0$ has to obey the following properties:

1. *Consistency*: $t_0 > \max\{t_1, t_2\}$.
2. *Monotonicity*: an increase of t_1 or t_2 increases t_0.
3. *Upwinding*: the update has to be accepted only from a triangle containing the characteristic direction (characteristics of the eikonal equation coincide with minimum geodesics on the surface).
4. *Numerical stability*: a small perturbation in t_1 or t_2 results in a bounded perturbation in t_0.

In the original fast marching algorithm, a vertex is updated by simulating a planar wavefront propagating inside the triangle [17]; the values of the two supporting vertices allow to compute the front direction. The same update scheme was used in [20]. Here, we develop a similar scheme, expressing it more compactly and without the use of trigonometric functions, which allow more efficient computation. We model the wavefront as a planar wave propagating from a virtual planar source described by the

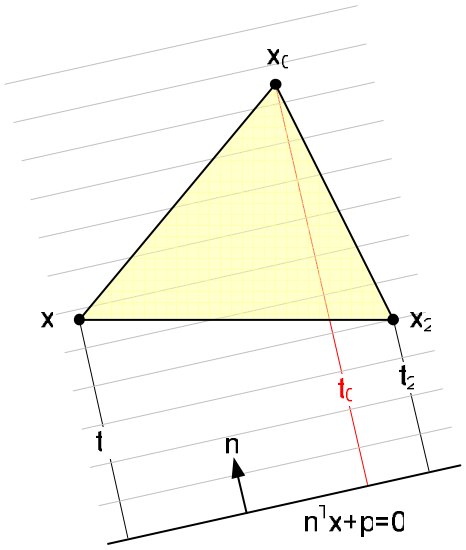

Figure 2: Update scheme based on the planar wavefront propagation model.

equation $\mathbf{n}^{\mathrm{T}}\mathbf{x}+p=0$, where $\mathbf{n}$ is the propagation direction (Figure 2). Demanding that the supporting vertices $\mathbf{x}_1$, $\mathbf{x}_2$ of the triangle lie at distances t_1 and t_2, respectively, from the source, we obtain

$$\mathbf{X}^{\mathrm{T}}\mathbf{n}+p\cdot\mathbf{1} \quad = \quad \mathbf{t}, \tag{6}$$

where $\mathbf{X}$ is a matrix whose columns are $\mathbf{x}_1$ and $\mathbf{x}_2$, $\mathbf{1}=(1,1)^{\mathrm{T}}$, and $\mathbf{t}=(t_1,t_2)^{\mathrm{T}}$. The wavefront time of arrival to the updated vertex $\mathbf{x}_0$ is given by its distance from the planar source,

$$t_0 \quad = \quad \mathbf{n}^{\mathrm{T}}\mathbf{x}_0+p \;=\; p. \tag{7}$$

Assuming that the mesh is non-degenerate, $\mathbf{x}_1$ and $\mathbf{x}_2$ are linearly independent, and we can solve (6) for $\mathbf{n}$, obtaining

$$\mathbf{n} \quad = \quad \mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}(\mathbf{t}-p\cdot\mathbf{1}). \tag{8}$$

Invoking the condition $\|\mathbf{n}\|=1$ yields

$$\begin{aligned} 1 &= \mathbf{n}^{\mathrm{T}}\mathbf{n} \\ &= (\mathbf{t}-p\cdot\mathbf{1})^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-\mathrm{T}}\mathbf{X}^{\mathrm{T}}\mathbf{X}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}(\mathbf{t}-p\cdot\mathbf{1}) \\ &= (\mathbf{t}-p\cdot\mathbf{1})^{\mathrm{T}}(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}(\mathbf{t}-p\cdot\mathbf{1}) \\ &= p^2\cdot\mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{1}-2p\cdot\mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{t}+\mathbf{t}^{\mathrm{T}}\mathbf{Q}\mathbf{t}, \end{aligned} \tag{9}$$

where $\mathbf{Q} = (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1} = \mathbf{E}^{-1}$. Hence, t_0 can be found as the largest solution of the quadratic equation

$$t_0^2 \cdot \mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{1} - 2t_0 \cdot \mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{t} + \mathbf{t}^{\mathrm{T}}\mathbf{Q}\mathbf{t} - 1 \quad = \quad 0 \tag{10}$$

(the smallest solution corresponds to the opposite propagation direction, where the wavefront arrives to $\mathbf{x}_0$ *before* it arrives to $\mathbf{x}_1$ and $\mathbf{x}_2$ and therefore has to be discarded). To speed the solution up, the terms $\mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{1}$ and $\mathbf{1}^{\mathrm{T}}\mathbf{Q}$ depending on the grid geometry only are pre-computed.

The consistency condition can be written as $p \cdot \mathbf{1} > \mathbf{X}^{\mathrm{T}}\mathbf{n} + p \cdot \mathbf{1}$ or simply $\mathbf{X}^{\mathrm{T}}\mathbf{n} < 0$, which can be interpreted geometrically as a demand that the direction $-\mathbf{n}$ must form an acute angle with the triangle edges. In order to impose monotonicity, we demand that

$$\nabla_{\mathbf{t}}\, t_0 \quad = \quad \left(\frac{\partial t_0}{\partial t_1}, \frac{\partial t_0}{\partial t_2}\right)^{\mathrm{T}} \; > \; 0. \tag{11}$$

Differentiating (10) with respect to $\mathbf{t}$, we obtain

$$t_0 \cdot \nabla_{\mathbf{t}}\, t_0 \cdot \mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{1} - \nabla_{\mathbf{t}}\, t_0 \cdot \mathbf{1}^{\mathrm{T}}\mathbf{Q}\mathbf{t} - t_0 \cdot \mathbf{Q}\mathbf{1} + \mathbf{Q}\mathbf{t} \quad = \quad 0, \tag{12}$$

from where

$$\nabla_{\mathbf{t}}\, t_0 \quad = \quad \frac{\mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1})}{\mathbf{1}^{\mathrm{T}}\mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1})}. \tag{13}$$

Substituting (8), we can write

$$\mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1}) \quad = \quad (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{n} \; = \; \mathbf{Q}\mathbf{X}^{\mathrm{T}}\mathbf{n}. \tag{14}$$

The monotonicity condition $\mathbf{X}^{\mathrm{T}}\mathbf{n} < 0$ and the fact that $\mathbf{Q}$ is positive definite imply that at least one of the coordinates of $\mathbf{Q}\mathbf{X}^{\mathrm{T}}\mathbf{n}$ must be negative. Hence, demanding $\nabla_{\mathbf{t}}\, t_0 > 0$ yields $\mathbf{1}^{\mathrm{T}}\mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1}) < 0$. The latter condition can be rewritten as

$$0 \quad > \quad \mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1}) \; = \; (\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}\mathbf{n}. \tag{15}$$

Observe that the rows of the matrix $(\mathbf{X}^{\mathrm{T}}\mathbf{X})^{-1}\mathbf{X}^{\mathrm{T}}$ are orthogonal to $\mathbf{x}_1$, $\mathbf{x}_2$, or in other words, are normal to the triangle edges. This gives the following geometric interpretation of the monotonicity condition: the direction $-\mathbf{n}$ must come from within the triangle. Since the update direction also obeys the consistency condition, any direction coming from within the triangle must form acute angles with the triangle edges, leading to the demand that the angle $\sphericalangle \mathbf{x}_1\mathbf{x}_0\mathbf{x}_2$ is acute (or, equivalently, $e_{12} > 0$).

Consistency and monotonicity conditions should guarantee that the update is performed only from a triangle that contains the characteristic direction, which makes the update scheme upwind [18]. However, since $\mathbf{n}$ is only an approximation of the characteristic direction, it may happen that the conditions are not satisfied although the true characteristic lies inside the triangle. For a sufficiently small triangle, this can happen only

Algorithm 2: Planar update scheme for acute triangulation.

```
Set t_0^new ⟵ t_0.
foreach triangle X = (x_1, x_2)^T do
    Solve the quadratic equation (10) for t_0.
    if Q(t − t_0 · 1) > 0 or t_0 < max{t(x_1), t(x_2)} then compute t_0 according to (16).
    Set t_0^new ⟵ min{t_0^new, t_0}.
end
```

if any of the two inner products $\mathbf{n}^{\mathrm{T}}\mathbf{x}_1$, $\mathbf{n}^{\mathrm{T}}\mathbf{x}_2$ is sufficiently close to zero. This corresponds to the situation in which t_0 can be updated from one of the triangle edges (one-dimensional simplices) $\mathbf{x}_0\mathbf{x}_1$, $\mathbf{x}_0\mathbf{x}_2$. In this case, the simple Dijkstra-type update,

$$t_0 \quad = \quad \min\{t_1 + \|\mathbf{x}_1\|, t_2 + \|\mathbf{x}_2\|\}, \tag{16}$$

is performed.

In order to ensure that the update formula is numerically stable, we assume that t_i is affected by a small error ε, which, in turn, influences the computed time of arrival t_0. Using first-order Taylor expansion, we have

$$\tilde{t}_0 \quad \approx \quad t_0 + \varepsilon \cdot \frac{\partial t_0}{\partial t_i} \;\leq\; t_0 + \varepsilon \cdot \left(\left| \frac{\partial t_0}{\partial t_1} \right| + \left| \frac{\partial t_0}{\partial t_2} \right| \right). \tag{17}$$

Under the monotonicity condition $\nabla_{\mathbf{t}}\, t_0 > 0$, we can write

$$\tilde{t}_0 \;\approx\; t_0 + \varepsilon \cdot \mathbf{1}^{\mathrm{T}} \nabla_{\mathbf{t}}\, t_0 \;=\; t_0 + \varepsilon \cdot \frac{\mathbf{1}^{\mathrm{T}}\mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1})}{\mathbf{1}^{\mathrm{T}}\mathbf{Q}(\mathbf{t} - p \cdot \mathbf{1})} \;=\; t_0 + \varepsilon. \tag{18}$$

The error in t_0 is also bounded in the one-dimensional Dijkstra-type update, which makes the update formula stable.

The planar wavefront update scheme is summarized in Algorithm 2. Note that it is valid only for acute triangulations; when some triangles have obtuse angles ($e_{12} < 0$), they have to be split by adding connections to additional neighbor grid points, as proposed by Spira and Kimmel in [20].

4 Raster scan algorithm

One of the disadvantages of the fast marching algorithm is that it is inherently sequential, thus allowing no parallelization. In addition, the order of visiting the grid points depend on the shape of the propagating wavefront and is therefore data-dependent. This results in irregular memory access that is unlikely to utilize the caching system efficiently. These drawbacks call for searching for alternative grid traversal orders.

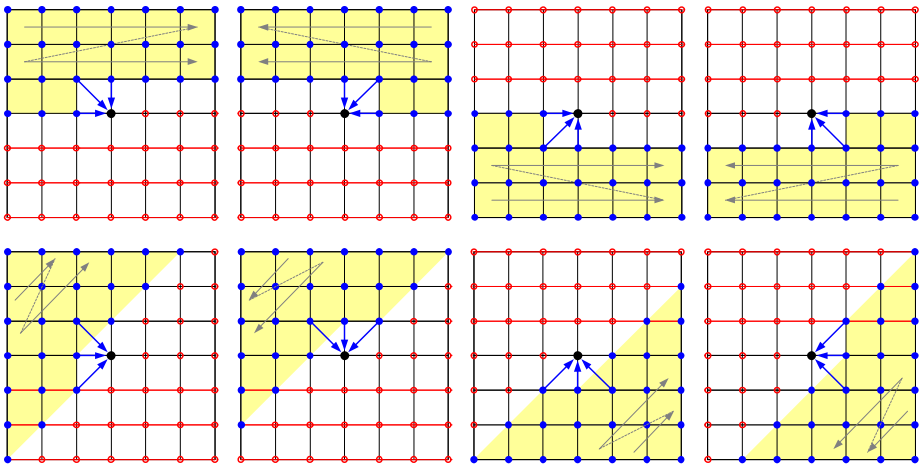

Figure 3: Update of a point on a grid with eight-neighbor connectivity using the raster scan algorithm. First row: four directed raster scans; second row: the same raster scans rotated by 45°.

In his classical paper, Danielsson [24] observed that since the geodesics on the Euclidean plane are straight lines, all possible characteristic directions of the eikonal equation fall into one of the four quadrants of a Cartesian grid and can be therefore covered by traversing the grid in four directed raster scans. Danielsson's raster scan spirit was adopted by Zhao [29] for solving the eikonal equation on weighted Euclidean domains; similar ideas date back to Dupuis and Oliensis' studies on shape from shading [30].

Raster scan traversal has linear complexity in the grid size, and is characterized by regular access to memory, which increases the efficiency of caching. Since the order of visiting of the grid points is independent of the data and is known in advance, one can use the pre-caching mechanism, supported in many modern processors. In addition, unlike its priority queue-based counterpart, raster scan can be efficiently parallelized as will be shown in Section 5.

Here, we use the raster scan order to traverse the Cartesian grid in the surface parametrization domain, as summarized in Algorithm 3. As in the priority queue-based traversal order, all computations are done in the parametrization domain, taking into account the metric on the surface. Since each directed raster scan covers only 90° of possible characteristic directions, the update of a point on the grid can be done only from the triangles containing that direction. For example, if the eight-neighbor grid connectivity is used, only two triangles formed by three neighbors are absolutely required in the update (Figure 3, first row).

Observe that unlike the Euclidean case where the characteristics are straight lines, on a general parametric surface, the characteristics in the parametrization domain are usually curved. This implies that the four raster scans may cover only a part of a characteristic, and have to be repeated more times in order to produce a consistent distance

Algorithm 3: Raster scan algorithm on a parametric surface.

Input: Numerical $M \times N$ grid $\mathbf{U}$, set of source points $S \subset \mathbf{U}$ with the corresponding initial values $t(s)$

Output: The distance map $t : \mathbf{U} \mapsto \mathbb{R}^+$.

Initialization

Pre-compute the update equation coefficients for each triangle.
foreach *point* $\boldsymbol{u} \in \mathbf{U} \setminus S$ **do** $t(\mathbf{u}) \longleftarrow \infty$

Iteration

for iter = *1,2,...* **do**
  for $i = 1,2,...,M$ **do**
    Right-up scan
    for $j = 1,2,...,N$ **do** `Update` $(\mathbf{u}_{ij})$
    Right-down scan
    for $j = N,N-1,...,1$ **do** `Update` $(\mathbf{u}_{ij})$
  end
  for $i = M,M-1,...,1$ **do**
    Left-up scan
    for $j = 1,2,...,N$ **do** `Update` $(\mathbf{u}_{ij})$
    Left-down scan
    for $j = N,N-1,...,1$ **do** `Update` $(\mathbf{u}_{ij})$
  end
  if $\|t^{(n)} - t^{(n-1)}\| \leq \varepsilon$ **then** stop
end

map. As a consequence, the complexity of the raster algorithm for parametric surfaces is $\mathcal{O}(N_{\text{iter}} \cdot n)$, where n is the grid size, and N_{iter} is the data-dependent number of iterations. In what follows, we present a bound on the maximum number of iterations.

Theorem 1 *The maximum number of raster scan iterations required to produce a consistent approximation of the distance map on a parametric surface* $\boldsymbol{x}(\mathbf{U})$ *is bounded by*

$$N_{\text{iter}} \quad \leq \quad \left\lceil \frac{2D\,\lambda^{\boldsymbol{G}}_{\max}}{\pi\lambda^{\boldsymbol{G}}_{\min}} \sqrt{(\lambda^{\boldsymbol{H}^1}_{\min})^2 + (\lambda^{\boldsymbol{H}^2}_{\min})^2 + (\lambda^{\boldsymbol{H}^3}_{\min})^2} \right\rceil + 1.$$

where D is the surface diameter, $\lambda^{\boldsymbol{H}^i}_{\min}$ *is the smallest eigenvalue of the Hessian matrix* $\boldsymbol{H}^i = \nabla^2_{\boldsymbol{uu}} x^i$ *of* x^i *with respect to the parametrization coordinates* $\boldsymbol{u}$*, and* $\lambda^{\boldsymbol{G}}_{\max}/\lambda^{\boldsymbol{G}}_{\min}$ *is the condition number of the metric* $\boldsymbol{G}$*.*

For proof, see Appendix A. When the surface is given as a graph of a function $z(x,y)$, the bound can be simplified as

$$N_{\text{iter}} \quad \leq \quad \left\lceil \frac{2D\,\lambda^{\mathbf{G}}_{\max}}{\pi\lambda^{\mathbf{G}}_{\min}} \lambda^{\mathbf{H}}_{\min} \right\rceil + 1, \tag{19}$$

where $\mathbf{H} = \nabla^2 z$.

The main significance of this bound is that the maximum number of iterations does not depend on the discretization of $\mathbf{U}$ and is a constant regardless of the grid size. Note, however, that the bound depends both on the properties of the surface expressed in terms of the metric $\mathbf{G}$ and the diameter D, and those of the parametrization expressed in terms of the Hessian $\mathbf{H}^i$. This means that some parametrizations of the same surface may be less favorable for the raster scan algorithm. For example, in the parametrization $\mathbf{x} = (u^1 \cos u^2, u^1 \sin u^2, 0)^{\mathrm{T}}$ of a flat disc, the characteristics in the parametrization domain are curved and require multiple iterations to be covered.

Note that the bound is a worst case bound; in practice the number of iterations required for convergence may be smaller. Adding another triangle to the grid update such that every grid point is updated from four "causal" (in the raster scan order) neighbors rather than from three causal neighbors as shown in Figure 3 may reduce the number of iterations. It is important to emphasize that in the worst case N_{iter} will remain unchanged.

5 Parallelization

The structure of the raster scan algorithm gives much opportunity for exploiting data independence to compute some of the grid updates concurrently on a set of parallel computation units. Here we explore parallelization of two different problems: computation of all pair-wise geodesic distances on a surface and computation of a single distance map from an arbitrary source. In addition, we show how to parallelize the latter computation on a graphics processing unit using its architecture-specific features.

5.1 Computation of all pair-wise distances

Given a grid of n vertices in the parametrization domain, our goal is to compute the $n \times n$ matrix, whose elements are the geodesic distances $d(\mathbf{u}_i, \mathbf{u}_j)$ between all pairs of points $\mathbf{u}_i$ and $\mathbf{u}_j$ on the surface. The computation can be split into n separate problems of solving the eikonal equation on the same grid for n sets of different boundary conditions, or, said differently, computing n distance maps $T^i = \{t_j^i : j = 1, ..., n\}$ from the point sources $t_i = 0$. Since there is no dependence between the problems, they can be solved concurrently on a set of P processors. The fact that the same computation is performed for each updated grid point allows to use a single stream of instructions for all the CPUs operating on multiple streams of data. This makes the algorithm especially attractive for implementation on *single instruction multiple data* (SIMD) processors.

The parallel algorithm works as outlined in Algorithm 4. First, the grid is initialized by pre-computing the coefficients of the update equation (10), which are shared by all processors, and the $n \times n$ distance matrix is allocated. To exploit better memory alignment, the memory is organized by collocating the data belonging to each of the P processors, i.e., $t_1^1, t_1^2, ..., t_1^P, t_2^1, t_2^2, ..., t_2^P$, *etc.*

Algorithm 4: Parallel computation of pair-wise geodesic distances.

Input: Numerical grid $\mathbf{U}$ containing n vertices.
Output: The pair-wise distances $T : \mathbf{U} \times \mathbf{U} \mapsto \mathbb{R}^+$.

Pre-compute the update equation coefficients for each triangle.
for $k = 0, P, 2P, ..., n-P$ **do**
    Initialization
    for $i = k, k+1, ..., k+P-1$ **do**
        $t_i^i \longleftarrow 0$.
        for $j = 1, 2, ..., k$ **do** $t_j^i \longleftarrow t_i^j$
        for $j = k+1, k+2, ..., n$ **do** $t_j^i \longleftarrow \infty$
    end
    Iteration
    for iter = *1,2,... until convergence* **do**
        Perform Steps 4-11 of Algorithm 3 for t^k, t^{k+1}, ..., t^{k+P-1} concurrently on P processors.
    end
end

Second, the first P distance maps are initialized by setting $t_i^i = 0$ and $t_j^i = \infty$ for all $i = 1, ..., P$ and $j = 1, ..., n$, $j \neq i$. The raster scan algorithm is then executed, where an update of a grid point i is performed by all the processors simultaneously, each of which operates on its own distance map T^i. After the completion of the raster scan algorithm, the elements t_j^i, $i = 1, ..., P$; $j = 1, ..., n$ of the distance matrix become available.

The step is repeated for the next set of distance maps $T^{P+1}, ..., T^{2P}$. Note that since the distance matrix is symmetric, we can use the previously computed values of t_j^i in the initialization,

$$t_j^i \quad = \quad \begin{cases} t_i^j & : \quad 1 \leq j \leq P \\ 0 & : \quad i = j \\ \infty & : \quad \text{otherwise,} \end{cases} \tag{20}$$

and skip the updates of the first P points on the grid. The algorithm proceeds, computing each time n distance maps until the entire matrix is computed. Note that the last distance maps require less computation time due to the fact that many grid points are initialized with the values from the previously computed distance maps, and are therefore not updated. The complexity of the algorithm is $\mathcal{O}(n^2)$, and the speedup due to parallelization on P processors is exactly P, which is usually labeled with the term "embarrassingly parallel" in the parallel computing jargon.

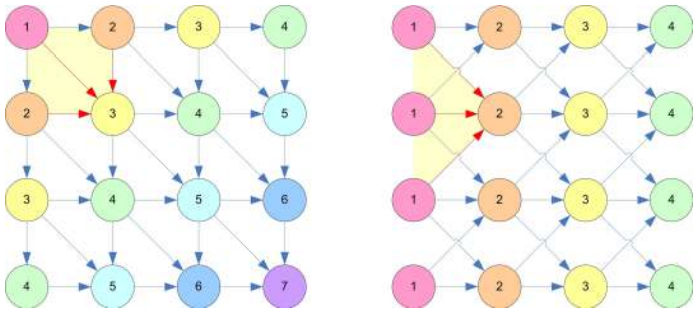

Figure 4: Dependency graph in the right-down (left) and the rotated up-left-down (right) raster scan updates. Grid point updates that can be computed concurrently are numbered and shaded with different colors.

5.2 Computation of a single distance map

The structure of the raster scan algorithm allows to exploit data independency of many grid updates when computing a distance map from a single source. To demonstrate the parallelism, let us consider for example the right-down raster scan, starting from the top leftmost grid point t_{11}. After t_{11} has been updated, the points t_{12} and t_{21} can be updated concurrently, since their updates do not depend on each other. Next, the points t_{31}, t_{22} and t_{13} are updated concurrently, and so on (Figure 4, left). Assuming the number of available computation units is $P \geq \min\{M,N\}$, the right-down raster scan can be performed in $M+N-1$ steps, where at each step k the points along the line $i+j=k+1$ are updated. If the number of processors is smaller, every step is serialized into $\lceil (k+1)/P \rceil$ sub-steps. The other three directed raster scans are parallelized in the same manner.

An obvious disadvantage of such a parallelization is the lack of data coherence in the memory, which may deteriorate performance on many architectures such as GPUs. Another disadvantage is the fact that the number of operations in each step is not constant and the benefit from the parallelization is obtained only on sufficiently long diagonals. A way to overcome these two difficulties is to rotate the direction of all raster scans by 45° (Figure 3, second row). Using the rotated raster scans, rows or columns of the grid can be updated concurrently (Figure 4, right). This allows coherent access to memory and provides better parallelization with a speedup factor of P. Since the same operations are performed to update all the grid points, the algorithm is suitable for implementation on a SIMD processor.

5.3 Distance map computation on a GPU

Recent advances in GPU architectures make these processors increasingly attractive for general purpose computing [31]. Usually, GPU's highly-parallel programmable fragment processors are exploited for performing arithmetic operations, and texture

memory is used for storing the data. Here, we discuss the implementation of the raster scan algorithm on a GPU.

It is important to keep in mind that GPUs are not general purpose processors and, therefore, implementation of general algorithms requires several issues to be considered. First, architectural limitations do not allow the same memory location to be accessed for read and write simultaneously. A common technique to overcome this difficulty is the *ping-pong* methodology [32], according to which two texture buffers are held. At each iteration, the fragment shader reads the data from the first buffer containing the current distance map, and simultaneously writes the updated distance map to the second buffer. At the end of the iteration, the buffers are swapped.

Second, the texture buffers have to be accessed coherently in order to exploit the GPU memory efficiently. The structure of the raster scan algorithm with rotated scan directions fits well this requirement, since each fragment is updated according to adjacent fragments.

Third, the *arithmetic intensity* (i.e., the ratio between arithmetic calculations and memory accesses) should be high. In order increase the ratio, we pack every 2×2 grid points into one fragment (in our implementation, 4×32 IEEE floating point textures were used). In addition, another four times bigger texture is held, in which each four texels contain the pre-computed coefficients of the update equation for all the triangles participating in the update of the corresponding vertex. Although such a representation is redundant, it saves multiple texture fetches.

Another architectural consideration requires many fragments (thousands) to be processed at each iteration to achieve good pipeline utilization. On medium-sized meshes containing about $1,000$ vertices in each row or column, this can be achieved by combining all the four scans. On smaller meshes (below 128×128 vertices), performing n updates of all n grid points appears to be faster. Another way to achieve better pipeline utilization is to update several rows or columns simultaneously, repeating the computation several times.

Additional speedup is made by skipping vertices, all of whose neighbors have infinite distances. For example, at the first iterations during the left to right scans, points located left to the leftmost source point will not be updated and can be skipped. Moreover, points that are distant from the source point can also be skipped at the first iterations. The selection of fragments to skip is implemented using the *scissor test*.

It is worthwhile noting that there usually exists some overhead in transferring data to and from the GPU, which varies from hardware to hardware. Therefore, a standalone implementation of a distance computation algorithm may result less efficient. However, in many applications there is no need to transfer all the distance map back to the CPU; for example, in order to compute a geodesic path, only a small number of distances has to be fetched. In other applications, operations performed on the distance map can be executed on the GPU itself, thus requiring no data transfer back to the CPU.

6 Numerical results

In order to demonstrate the efficiency of the proposed algorithms, we consider its two particular implementations. The pairwise distance computation algorithm was implemented in C on an AMD Opteron platform, with the update step written in assembly and taking advantage of the SSE2 extensions (Intel-compatible SIMD architecture). The single distance map computation algorithm was implemented on an nVidia 7950 GT GPU, using the CG shading language. Single precision floating point representation was used in both programs. Grid construction and pre-computation of coefficients was done on the CPU and was excluded from time measurements. Since this step is fully parallelizable, its implementation on a GPU would make this time complexity negligible compared to the complexity of the iteration itself.

6.1 Performance benchmarks

Table 1 presents the execution times of the SSE2 implementation with $N_{\text{iter}} = 1$ on grids of sizes varying from $1,024$ to $4,096$ vertices (1 to 16.8 million distances). All timing measurements were averaged over ten runs and do not include grid memory allocation. For comparison, execution times of optimized C code of the sequential fast marching method are presented (for fairness of comparison, grid construction time was not measurement). The SSE2 implementation computes approximately 6.8 million distances per second, which is about ten times faster than fast marching. Table 2 presents the execution times of the GPU implementation with $N_{\text{iter}} = 1$ on grids of sizes varying from $22,000$ to 1.44 million vertices. All timing measurements were averaged over ten runs and do not include data transfer to and from the GPU. For the 1.44 million vertex grid, the transfer overhead was about 60 msec. About 1.95 million distances per second are computed[1], which is almost 90 time faster compared to sequential fast marching and exceeds by over 50 times the best results with comparable numerical error reported so far [22].

6.2 Numerical accuracy

Figure 5 presents the convergence plots of a single iteration of the raster scan algorithm on three simple surfaces (plane, tilted plane with obtuse angles, and a sphere). The choice of the surfaces was governed by the availability of analytic expressions for the geodesic distances. Note the linear convergence of the algorithm, which is a manifest of its first-order accuracy. The dependence of the distance map accuracy on the number of iterations N_{iter} is visualized in Figure 6, which shows the distance map computed from a single point source on the "maze" surface with complicated spiral characteristics. As it appears from the figure, three iterations are sufficient to produce a consistent

[1] This number is lower compared to the SSE2 implementation, since in the tests performed with GPU code, the mesh sizes, and as consequence the length of the geodesics, were significantly larger.

Vertices	Distances	Raster scan Exec. time (sec)	Fast Marching Exec. time (sec)
1,024	1,048,576	0.1541 ± 0.00	1.2824 ± 0.01
1,936	3,748,096	0.5156 ± 0.01	5.5156 ± 0.02
2,500	6,250,000	0.8348 ± 0.01	9.4335 ± 0.08
2,916	8,503,056	1.1336 ± 0.01	12.9128 ± 0.08
3,600	12,960,000	1.8539 ± 0.01	19.5472 ± 0.09
4,096	16,772,166	2.4762 ± 0.01	25.8347 ± 0.14

Table 1: Execution times of a single iteration of the raster scan algorithm on AMD Opteron with SSE2 extensions for pairwise geodesic distances computation on grids of different sizes. For a reference, execution times of the sequential fast marching is given.

Vertices	Exec. time (sec)	Mem. usage (bytes)
22,500	0.1110 ± 0.0031	1,620,000
40,000	0.1208 ± 0.0046	2,880,000
160,000	0.1629 ± 0.0045	11,520,000
360,000	0.2766 ± 0.0034	25,920,000
640,000	0.4192 ± 0.0260	46,080,000
1,000,000	0.6125 ± 0.0249	72,000,000
1,440,000	0.7362 ± 0.0044	103,680,000

Table 2: Execution times and peak memory consumption of a single iteration of the raster scan algorithm on an GPU for distance map computation on grids of different sizes.

approximation of the distance map. In general, our practice shows that few iterations are sufficient to obtain accurate distance map even on complicated surfaces.

6.3 Geodesic paths, offset curves, and Voronoi diagrams

Figure 7 shows several computational geometric operations requiring the knowledge of a distance map on a surface. For this visualization, a face surface from the Notre Dame University database was used [33]. The surface contained 21,775 vertices and 42,957 faces. In the first two examples, a distance map was computed from a point source located at the tip of the nose. Equi-distant contours were computed using the marching triangle technique in the parametrization domain and then projected back onto the surface. Minimum geodesic paths were computed by backtracking the curve from some starting point along the gradient of the distance map t in the parametrization domain. Formally, geodesic computation can be thought of as solution of the ordinary differential equation

$$\dot{\gamma} = -\mathbf{G}^{-1}\nabla_{\mathbf{u}} t, \tag{21}$$

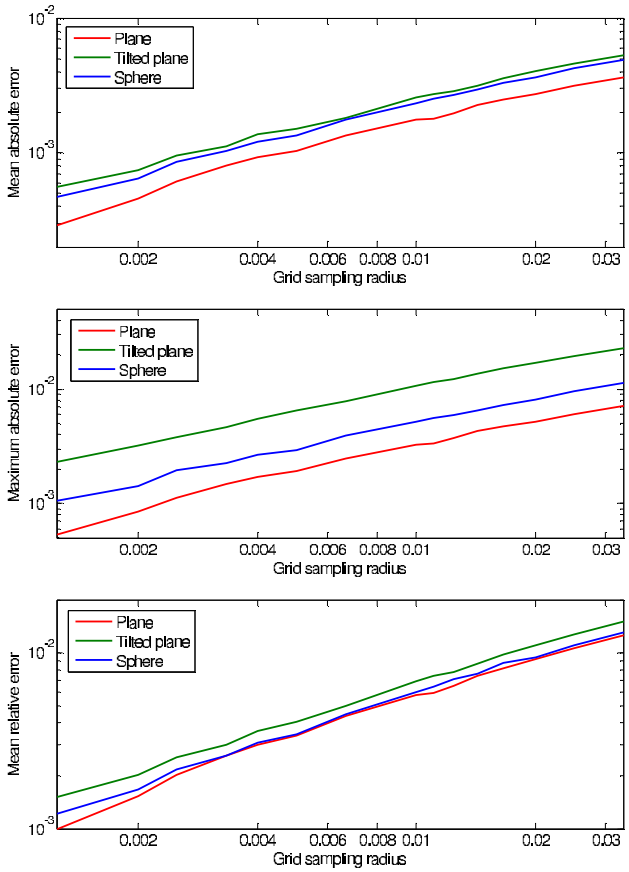

Figure 5: Convergence plots of the raster scan algorithm with $N_{\text{iter}} = 1$ for different mesh sampling radii.

where $\gamma(s)$ is the geodesic path in the parametrization domain and $\Gamma(s) = \mathbf{x}(\gamma(s))$ is the geodesic on the surface. A first-order integration technique to compute $\gamma(s)$.

In the third example, a distance map from 20 random points on the surface was computed and a geodesic Voronoi diagram was found using marching triangles. In the fourth example, the distance map was computed from two disconnected curves and marching triangles were used to trace the geodesic offset curves.

6.4 Isometry-invariant canonical forms

As an example of application requiring the knowledge of the matrix of pair-wise geodesic distances on the surface, we show computation of *canonical forms* introduced in [5] as isometry-invariant signatures for classification of non-rigid objects. Figure 8

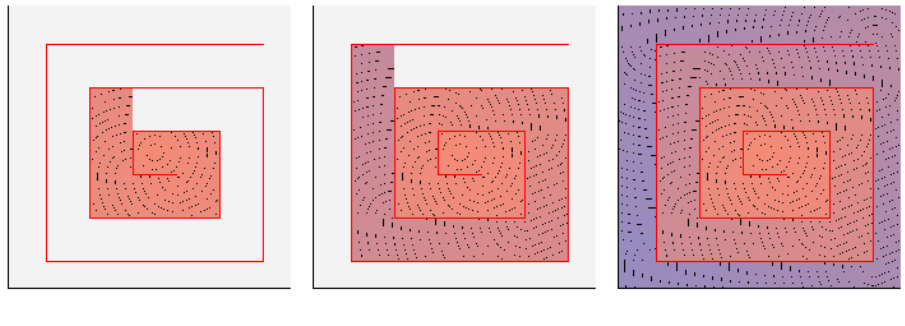

Figure 6: Distance map computation on the "maze" surface using the raster scan algorithm after one iteration (left), two iterations (middle) and three iterations (right). Equidistant contours in the parametrization domain are show. Grayed regions stand for infinite distance.

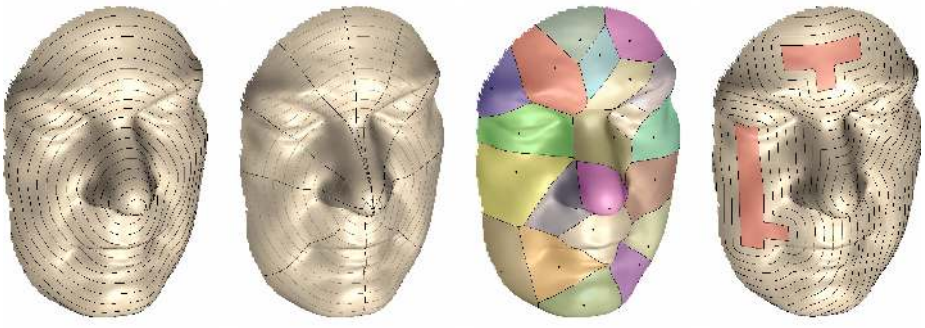

Figure 7: Computation of distance maps on a parametric surface ($21,775$ vertices, $42,957$ faces). Left-to-right: equi-distant contours; minimum geodesic paths; geodesic Voronoi diagram; offset curves.

shows a surface containing $2,695$ vertices and its canonical form computed from the distance matrix using least squared multidimensional scaling. All computations were performed on AMD Opteron with SSE2 extensions and took about one second. The algorithm was employed in our three-dimensional face recognition system [34], achieving near real-time performance.

7 Conclusion

We presented a raster scan-based version of the fast marching algorithm for computation of geodesic distances on parametric surfaces. The structure of the algorithm allowed its efficient parallelization on SIMD processors and GPUs, which have been

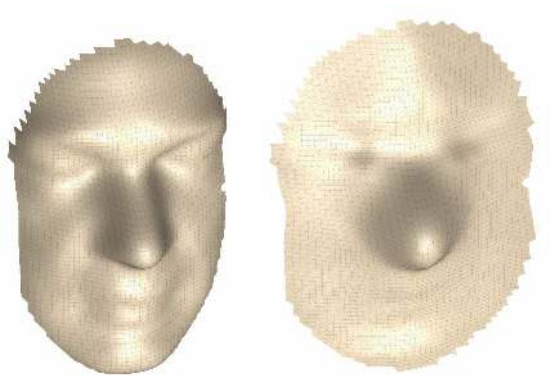

Figure 8: A surface (left) and its isometry-invariant canonical form (right) obtained by embedding into the Euclidean space using LS-MDS. Fast computation of canonical forms enables non-rigid shape matching applications such as expression invariant face recognition.

considered in this paper. Numerical experiments showed that the proposed method outperforms state-of-the-art methods for first-order distance map approximation by one or two orders of magnitude, thus allowing real-time implementation of applications involving intensive geodesic distance computations. In our sequel works, we are going to demonstrate some of such applications.

The main limitation of the presented approach is that it works with parametric surfaces represented as a single chart, though the latter can be of arbitrarily complex topology. Such a topology usually introduces "holes" in the parametrization domain, which are handled efficiently by our code by "masking" the update in those regions. Generalizing the algorithm to multiple charts or triangular meshes as well as adaptive samplings is an interesting research direction.

A Proof of Theorem 1

Let $\Gamma(s)$ be the characteristic curve with on the surface, s its arclength, and $\gamma(s) = (u^1(s), u^2(s))^{\mathrm{T}}$ its parametrization in $\mathbf{U}$. Since $\Gamma(s) = \mathbf{x}(\gamma(s))$, using the chain rule we obtain

$$\begin{aligned} \dot{\Gamma} &= \mathbf{T}\dot{\gamma} \\ \ddot{\Gamma} &= \mathbf{T}\ddot{\gamma} + \mathbf{r}, \end{aligned} \tag{22}$$

where $\mathbf{r} = (\dot{\gamma}^{\mathrm{T}}\mathbf{H}^1\dot{\gamma}, \dot{\gamma}^{\mathrm{T}}\mathbf{H}^2\dot{\gamma}, \dot{\gamma}^{\mathrm{T}}\mathbf{H}^3\dot{\gamma})^{\mathrm{T}}$ and $\mathbf{H}^i = \nabla^2_{\mathbf{uu}} x^i$ are the Hessian matrices of x^i with respect to the parametrization coordinates $\mathbf{u}$. Since Γ is a geodesic, $\ddot{\Gamma}$ is normal to the surface and hence

$$0 = \mathbf{P}_{\mathbf{T}}\ddot{\Gamma} = \mathbf{T}\ddot{\gamma} + \mathbf{P}_{\mathbf{T}}\mathbf{r}, \tag{23}$$

where $\mathbf{P_T}$ denotes the projection on the tangent space.

Hence,

$$\begin{aligned} \|\mathbf{T}\ddot{\gamma}\| &= \|\mathbf{P_T}\mathbf{r}\| \leq \|\mathbf{r}\| \\ &\leq \sqrt{(\lambda_{\min}^{\mathbf{H}^1})^2 + (\lambda_{\min}^{\mathbf{H}^2})^2 + (\lambda_{\min}^{\mathbf{H}^3})^2} \cdot \|\dot{\gamma}\|, \end{aligned} \tag{24}$$

where $\lambda_{\min}^{\mathbf{H}^i}$ is the smallest eigenvalue of the Hessian $\mathbf{H}^i$.

Since Γ is a geodesic, $\|\dot{\Gamma}\| = 1$. From (22) we have

$$1 = \|\dot{\Gamma}\|^2 = \dot{\gamma}^{\mathrm{T}}\mathbf{T}^{\mathrm{T}}\mathbf{T}\dot{\gamma} = \dot{\gamma}^{\mathrm{T}}\mathbf{G}\dot{\gamma} \geq \lambda_{\min}^{\mathbf{G}} \cdot \|\dot{\gamma}\|^2. \tag{25}$$

Hence, $1/\lambda_{\max}^{\mathbf{G}} \leq \|\dot{\gamma}\|^2 \leq 1/\lambda_{\min}^{\mathbf{G}}$. In a similar manner,

$$\|\mathbf{T}\ddot{\gamma}\|^2 = \ddot{\gamma}^{\mathrm{T}}\mathbf{T}^{\mathrm{T}}\mathbf{T}\ddot{\gamma} = \ddot{\gamma}^{\mathrm{T}}\mathbf{G}\ddot{\gamma} \geq \lambda_{\min}^{\mathbf{G}} \cdot \|\ddot{\gamma}\|^2 \tag{26}$$

Combining the above results, yields a bound on the curvature of γ

$$\begin{aligned} \kappa &= \frac{\|\ddot{\gamma} \times \dot{\gamma}\|}{\|\dot{\gamma}\|^3} \leq \frac{\|\ddot{\gamma}\|}{\|\dot{\gamma}\|^2} \\ &\leq \frac{\lambda_{\max}^{\mathbf{G}}}{\lambda_{\min}^{\mathbf{G}}} \sqrt{(\lambda_{\min}^{\mathbf{H}^1})^2 + (\lambda_{\min}^{\mathbf{H}^2})^2 + (\lambda_{\min}^{\mathbf{H}^3})^2}. \end{aligned} \tag{27}$$

Therefore, the total variation of the tangential angle of γ is bounded by

$$\mathrm{TV}(\phi) = \int_{\gamma} \kappa ds \leq \max \kappa \cdot \int_{\Gamma} ds \leq \max \kappa \cdot D. \tag{28}$$

In the worst case, an iteration is required for every $\pi/2$ in $\mathrm{TV}(\phi)$ to consistently cover the characteristic γ, which completes the proof.

References

[1] S. Katz and A. Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. on Graphics*, 22(3):954–961, July 2004.

[2] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkewicz, and D. Dobkin. Modeling by example. In *Proc. SIGGRAPH*, pages 652–663, 2004.

[3] P.-P. J. Sloan, C. F. Rose, and M. F. Cohen. Shape by example. In *ACM Symposium on Interactive 3D Graphics*, pages 133–144, 2001.

[4] E. Praun, H. Hoppe, and A. Finkelstein. Robust mesh watermarking. In *Proc. SIGGRAPH*, pages 49–56, 1999.

[5] A. Elad and R. Kimmel. Bending invariant representations for surfaces. In *Proc. CVPR*, pages 168–174, 2001.

[6] M. Hilaga, Y. Shinagawa, T. Komura, and T. L. Kunii. Topology matching for fully automatic similarity estimation of 3D shapes. In *Proc. SIGGRAPH*, pages 203–212, 2001.

[7] F. Mémoli and G. Sapiro. A theoretical and computational framework for isometry invariant recognition of point cloud data. *Foundations of Computational Mathematics*, 5(3):313–347, 2005.

[8] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Generalized multidimensional scaling: a framework for isometry-invariant partial surface matching. *Proc. National Academy of Sciences*, 103(5):1168–1172, January 2006.

[9] G. Zigelman, R. Kimmel, and N. Kiryati. Texture mapping using surface flattening via multi-dimensional scaling. *IEEE Trans. Visualization and computer graphics*, 9(2):198–207, 2002.

[10] K. Zhou, J. Snyder, B. Guo, and H.-Y. Shum. Iso-charts: Stretch-driven mesh parameterization using spectral analysis. In *Symposium on Geometry Processing*, 2004.

[11] G. Peyré and L. Cohen. Geodesic re-meshing and parameterization using front propagation. In *Proc. VLSM'03*, 2003.

[12] A. M. Bronstein, A. M. Bronstein, and R. Kimmel. Calculus of non-rigid surfaces for geometry and texture manipulation. *IEEE Trans. Visualization and Comp. Graphics*, 2006. to appear.

[13] J.A. Sethian and A.M. Popovici. 3-d traveltime computation using the fast marching method. *Geophysics*, 64(2):516–523, 2006.

[14] J. Hershberger and S. Suri. An optimal algorithm for Euclidean shortest paths in the plane. *SIAM J. Computing*, 28(6), 1999.

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

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

[17] R. Kimmel and J. A. Sethian. Computing geodesic paths on manifolds. *Proc. of National Academy of Sciences*, 95(15):8431–8435, 1998.

[18] J. A. Sethian and A. Vladimirsky. Fast methods for the Eikonal and related Hamilton-Jacobi equations on unstructured meshes. *PNAS*, 97(11):5699–5703, 2000.

[19] F. Mémoli and G. Sapiro. Fast computation of weighted distance functions and geodesics on implicit hyper-surfaces. *Journal of Computational Physics*, 173(1):764–795, 2001.

[20] A. Spira and R. Kimmel. An efficient solution to the eikonal equation on parametric manifolds. *Interfaces and Free Boundaries*, 6(4):315–327, 2004.

[21] J. S. B. Mitchell, D. M. Mount, and C. H. Papadimitriou. The discrete geodesic problem. *SIAM Journal of Computing*, 16(4):647–668, 1987.

[22] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. In *Proc. SIGGRAPH*, pages 553–560, 2005.

[23] X. Gu, S.J. Gortler, and H. Hoppe. Geometry images. *ACM Transactions on Graphics*, 21(3):355–361, 2002.

[24] P.-E. Danielsson. Euclidean distance mapping. *Computer Graphics and Image Processing*, 14:227–248, 1980.

[25] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *Proc. IEEE Visualization*, pages 83–90, 2003.

[26] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Fast computation of generalized Voronoi diagrams using graphics hardware. In *Proc. ACM SIGGRAPH*, pages 277–286, 1999.

[27] I. Fischer and C. Gotsman. Fast approximation of high order Voronoi diagrams and distance transforms on the GPU. Technical report CS TR-07-05, Harvard University, 2005.

[28] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3D distance field computation using linear factorization. In *Proc. ACM Symposium on Interactive 3D Graphics and Games*, pages 117–124, 2006.

[29] Hongkai Zhao. A fast sweeping method for eikonal equations. *Mathematics of computation*, 74(250):603–627, 2004.

[30] P. Dupuis and J. Oliensis. Shape from shading: Provably convergent algorithms and uniqueness results. In *Proc. ECCV*, pages 259–268, 1994.

[31] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kruger, A.E. Lefohn, and T.J. Purcell. A survey of general-purpose computation on graphics hardware. *Eurographics 2005, State of the Art Reports*, pages 21–51, 2005.

[32] GPGPU : General-purpose computation using graphics hardware. Website: `www.gpgpu.org`.

[33] K. Chang, K. Bowyer, and P. Flynn. Face recognition using 2D and 3D facial data. *ACM Workshop on Multimodal User Authentication*, pages 25–32, 2003.

[34] A. M. Bronstein, M. M. Bronstein, and R. Kimmel. Three-dimensional face recognition. *International Journal of Computer Vision (IJCV)*, 64(1):5–30, August 2005.