Applied Mathematics and Computation 215 (2009) 235–250

Contents lists available at ScienceDirect

Applied Mathematics and Computation

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

Computing generalized higher-order Voronoi diagrams on triangulated surfaces

Marta Fort *, J. Antoni Sellarès

Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain

ARTICLE INFO

Keywords:
Computational geometry
Triangulated surface
Shortest path
Voronoi diagram
Hardware graphics

ABSTRACT

We present an algorithm for computing exact shortest paths, and consequently distance functions, from a generalized source (point, segment, polygonal chain or polygonal region) on a possibly non-convex triangulated polyhedral surface. The algorithm is generalized to the case when a set of generalized sites is considered, providing their distance field that implicitly represents the Voronoi diagram of the sites. Next, we present an algorithm to compute a discrete representation of the distance function and the distance field. Then, by using the discrete distance field, we obtain the Voronoi diagram of a set of generalized sites (points, segments, polygonal chains or polygons) and visualize it on the triangulated surface. We also provide algorithms that, by using the discrete distance functions, provide the closest, furthest and k-order Voronoi diagrams and an approximate 1-Center and 1-Median.

1. Introduction

Computing shortest paths, and consequently distances, on polyhedral surfaces is a fundamental problem in Computational Geometry with important applications in Geographical Information Systems, Robotics and Computer Graphics [23,35]. The computation of shortest path distances often arises as a subroutine in the solution of proximity and facility location problems. The k-order Voronoi diagram of a set of sites on a triangulated polyhedral surface associates to each subset of size k of sites the region of the triangulated surface that consists of points closer to those k sites than any other k sites. Voronoi diagrams allow to answer proximity queries like finding the nearest neighbor, the farthest neighbor, all the neighbors and the kth nearest neighbor [28]. Facility location problem tries to determine the 'best' location of a given set of facilities in various settings and under various constraints. The 1-Center is the point minimizing the maximal distance of a point on the triangulated surface to a set of sites, and the 1-Median is the point minimizing the sum of distances of a point on the polyhedral surface to the sites.

1.1. Preliminaries

Let $\mathscr{P}$ be a possibly non-convex polyhedral surface represented as a mesh consisting of n triangular faces $f_1,\ldots,f_n$. From now on, a *generalized element* on $\mathscr{P}$ refers to a point, segment, polygonal chain or polygon.

We only consider paths from *generalized sources* to points on $\mathscr{P}$ that stay on $\mathscr{P}$. The length of a path Π on $\mathscr{P}$ is defined by $\|\Pi\| = \sum_{i=1}^{n}|\Pi_i|$, where $|\Pi_i|$ denotes the Euclidean length of the path lying inside the face f_i. For a generalized source s and a point q on $\mathscr{P}$, the path of least length between them is called shortest path. The length of the shortest path is called the

* Corresponding author.
E-mail addresses: mfort@ima.udg.edu (M. Fort), sellares@ima.udg.edu (J.A. Sellarès).

0096-3003/$ - see front matter © 2009 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2009.04.075

(shortest path) distance between s and q. The *distance function* defined by a generalized source s on $\mathscr{P}$ is a function D_s such that for any point $q \in \mathscr{P}$, $D_s(q)$ is the distance between s and q.

When instead of a single source we have a set S of generalized sources on the polyhedral surface $\mathscr{P}$, the function that gives the distance from each point of $\mathscr{P}$ to its closest source in S is called the *distance field* defined by S.

Let S be a set of r *generalized sites* on the polyhedral surface $\mathscr{P}$ and let S' be a subset of k sites of S, $k \in \{1, \ldots, r-1\}$. The set of points of $\mathscr{P}$ closer to each site of S' than to any other site of S, is a possibly empty region called the k-order Voronoi region of S'. The set of k-order Voronoi regions of all subsets of k sites of S is called the k-order Voronoi diagram of S. When $k = 1$ and $k = r - 1$ the k-order Voronoi diagram is called the closest Voronoi and the furthest Voronoi diagram, respectively.

Finally, the 1-Center of a set S of generalized sites is the point on the polyhedral surface $\mathscr{P}$ that minimizes the maximum distance to the sites. The 1-Median of the set S is the point on $\mathscr{P}$ that minimizes the sum of distances to the sites.

1.2. Previous work

In this section we give an overview of previous work on shortest paths on polyhedral surfaces, generalized Voronoi diagrams computation and 1-Center and 1-Median determination.

1.2.1. Shortest paths

The single point source *shortest path* problem consists on finding a shortest path in the Euclidean metric from a source point to any target point such that the path stays on a triangulated polyhedral surface $\mathscr{P}$. Mitchell et al. [24] present an algorithm for solving the single point source shortest path problem by using a 'continuous Dijkstra' method which propagates distances from the source to the rest of $\mathscr{P}$. The algorithm constructs a data structure that implicitly encodes the shortest paths from a given source point to all other points of $\mathscr{P}$ in $O(n^2 \log n)$ time. The structure allows single-source shortest path queries, where the length of the path and the actual path can be reported in $O(\log n)$ and $O(\log n + \overline{n})$ time respectively, $\overline{n}$ is the number of mesh edges crossed by the path. Mitchell et al. also provide three shortest path properties that are used to locally characterize a shortest path on a polyhedral surface. These properties are restated by Surazhsky et al. [35]: (i) in the interior of a triangle a geodesic is a straight line. (ii) When crossing an edge a geodesic corresponds to a straight line if the two adjacent triangles are *unfolded* or placed into a common plane. (iii) a geodesic can go through a vertex if and only if it is a boundary vertex or if the total angle of the vertex is at least 2π.

Different improvements of this algorithm have been proposed [22,35]. Surazhsky et al. [35] described a simple way to implement the algorithm and showed to run much faster on most polyhedral surfaces than the $O(n^2 \log n)$ theoretical worst case time. Chen and Han [11], using a rather different approach, improved this to an $O(n^2)$ time algorithm. Their algorithm constructs a search tree and works by unfolding the facets of the polyhedral surface. The algorithm also answers single-source shortest path queries in $O(\log n + \overline{n})$ time. Kaneva and O'Rourke [17] implemented Chen and Han's algorithm and reported that the implementation is difficult for non-convex polyhedral surfaces and that memory size is a limiting factor of the algorithm. Kapoor [18] presented an algorithm following the 'continuous Dijkstra' paradigm that computes a shortest path from a source point to a target point in $O(n \log^2 n)$ time. Recently, Schreiber and Sharir [31] provided a $O(n \log n)$ worst case time algorithm for convex polyhedral surfaces.

1.2.2. Voronoi diagrams

The diverse generalizations of Voronoi diagrams (considering sites of different shape or nature, associating weights to the sites or changing the underlying metrics) have important applications in many fields and application areas, such as computer graphics, geometric modelling, geographic information systems, visualization of medical data-sets, pattern recognition, robotics and shape analysis [4,5,9,27].

Practical and robust optimal $O(r \log r)$ and $O(r^2)$ algorithms for computing the exact Voronoi diagram of a set of points in 2D and 3D have been extensively developed in computational geometry and related areas. On the other hand, the computation of exact generalized Voronoi diagrams used to be complicated, because it involved typically he manipulation of high-degree algebraic curves or surfaces and their intersections. For this reason, many authors have proposed algorithms to approximate the real diagram within a predetermined precision. Different algorithms based on distance functions have been proposed to compute 2D and 3D discretized closest Voronoi diagrams along a grid using graphics hardware [13,16,33,34]. These algorithms rasterize the distance functions of the generalized sites and use the depth buffer hardware to compute an approximation of the lower envelope of the distance functions.

Mount [25] shows that the closest Voronoi diagram of r sites on a polyhedral surface with n faces, assuming $r \leqslant n$, has complexity $O(n^2)$ in the worst case and also gives an algorithm that computes the diagram in $O(n^2 \log n)$ time. Aronov et al. [3] show that the furthest Voronoi diagram of the sites on the polyhedral surface has maximum combinatorial complexity $\Theta(rn^2)$, and present an algorithm that computes the diagram in $O(rn^2 \log^2 r \log n)$ expected time. Cabello et al. [7] study the complexity of higher-order Voronoi diagrams on triangulated surfaces under the geodesic distance.

1.2.3. 1-Center and 1-Median

Agarwal et al. [2] present an approach to efficiently solve the 1-Center problem for a set of r points in the plane using graphics hardware. Aronov et al. [3] give an algorithm for computing the 1-Center of a set of point sites for the special case

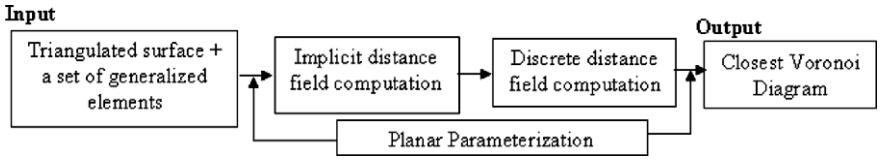

Fig. 1. Closest Voronoi diagram computation from the distance field defined by the sites.

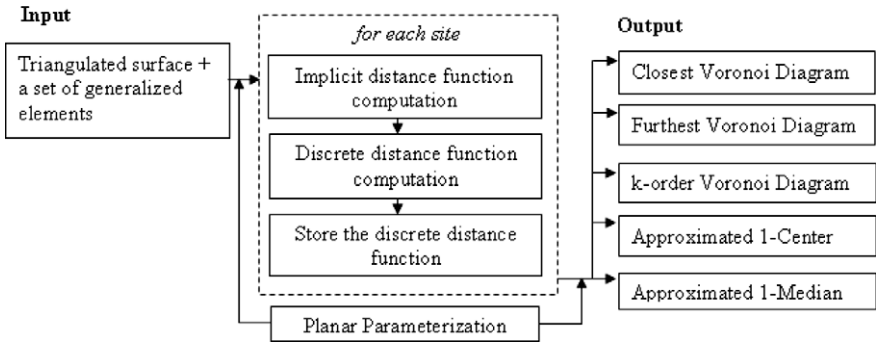

Fig. 2. Voronoi diagrams, 1-Center and 1-Median computation from the set of distance functions of the sites.

of a polyhedral terrain. The algorithm is based on the fact that the 1-Center lies at a vertex or on an edge of the furthest Voronoi diagram, consequently, given the diagram the location of the 1-Center can be determined in $O(rn^2)$ time.

No polynomial-time algorithm for computing the 1-Median, also known as the Fermat–Weber point, of a set of r point sites in the Euclidean plane is known, nor has the problem been shown to be NP-hard. The most common approximation algorithm is Weiszfeld's algorithm [36], an iterative procedure that converges to the 1-Median. Chandrasekaran and Tamir [10] present a polynomial time approximation scheme based on the standard ellipsoid method. Bose et al. [6] give a linear-time randomized algorithm and an $O(r \log r)$ time deterministic one for $1+\epsilon$ approximations of the Euclidean median. To the best of our knowledge, Lanthier et al. [21] presented the only algorithm available to provide an approximation to the 1-Median of r point sites on the surface of a polyhedral terrain with n triangular faces. The theoretical worst case running time of the algorithm is $O(rn^2 \log rn + rn^3)$. However, for typical terrain data, the expected running time is $O(rn \log rn)$ in both cases.

1.3. Our contribution

We present an exact algorithm to compute implicit distance functions on polyhedral surfaces from generalized sources. The algorithm uses the continuous Dijkstra paradigm (Section 3). We extend the algorithm to the case when several generalized sources are considered, obtaining their distance field (Section 4) that implicitly represents the Voronoi diagram of the set of sources (sites). We describe how the distance from the source to any point of the polyhedral surface and the actual shortest path can be obtained (Section 5).

Next, we describe an algorithm to obtain a discrete representation of the implicit distance function or the implicit distance field. The algorithm exploits graphics hardware capabilities (Section 6). As application, we provide several algorithms to compute discrete k-order Voronoi diagrams of a set of generalized sites on a triangulated polyhedral surface (Section 7). In Section 7.1 we present an algorithm to compute the closest Voronoi diagram from the implicit distance field defined by a set of generalized sites on the polyhedral surface (see Fig. 1).

Finally, given a set of generalized sites and by using their distance functions on the polyhedral surface we compute: (i) the closest and furthest Voronoi diagram (Section 7.3); (ii) a k-order Voronoi diagram (Section 7.3); (iii) an approximate 1-Center and 1-Median (Section 7.4) (see Fig. 2).

2. Graphics hardware

The increasing programmability and high computational rates of graphics processing units (GPUs) make them attractive as an alternative to CPUs for general-purpose computing. Recently, different algorithms and applications that exploit the inherent parallelism, easy programmability and vector processing capabilities of GPUs have been proposed [29,30]. In computational geometry and GIS fields there exist several algorithms that have a fast hardware-based implementation [2,12,15,16,19,20,26].

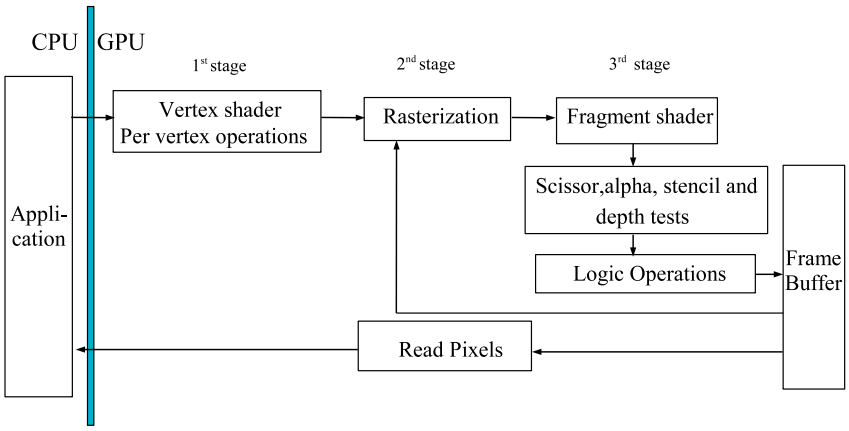

Fig. 3. The graphics pipeline.

The graphics pipeline [32] is divided into several stages, which are implemented as separate pieces of hardware on the GPU. The input to the pipeline is a list of 3D geometric primitives expressed as vertices defining points, lines, or polygon corners with associated attributes such as 3D coordinates, color and texture coordinates. The output is an image in the frame buffer, a collection of several hardware buffers corresponding to two dimensional grids whose cells are called pixels.

In the first stage of the pipeline, per-vertex operations take place, each input vertex is transformed from 3D coordinates to window coordinates. Next stage is rasterization, when it finishes we obtain a fragment, with its associated attributes, for each pixel location covered by a geometric primitive. Fragment attributes such as depth, color, and texture coordinates are obtained from the attributes associated to the vertices by linear interpolation. The third stage, the fragment stage, computes the color for each pixel according to the fragments corresponding to it. Each fragment passes a series of tests such as the depth test and per-fragment operations before being placed into its corresponding pixel of the frame buffer. Per fragment operations can use values from textures to modify its depth, color, etc. Finally, the fragments that pass the tests and the operations of the previous stage are drawn or rendered on the screen or on a specified texture with the appropriate color. Information of a user defined rectangle of the color buffer can be easily transferred to the CPU or directly to a texture. The graphics pipeline is schematically represented in Fig. 3.

The unique programmable parts of the graphics pipeline are the vertex and fragment shaders. Vertex shaders are executed on a per-vertex basis and fragment shaders, also called pixel shaders, are executed on a per-fragment basis. Fragment shaders can change the appearance of the pixels by combining fragment values, such as color and depth, with values stored in the fragment attributes or in textures, which are sent to them as parameters.

3. Implicit distance function

A quick overview of our algorithm to compute shortest path distances from a generalized source on a polyhedral surface is provided first. The shortest path problem from a generalized source s is solved by tracking together groups of shortest paths and partitioning each triangle edge into a set of intervals. All shortest paths that cross an interval are encoded locally using a parameterization of the distance function D_s. After an initialization step, where intervals encoding D_s in the edges of the triangles containing s are created, the distance function is propagated across mesh triangles in a 'continuous Dijkstra' fashion by repeatedly using an interval propagation process. A complete intrinsic representation of D_s is obtained when the propagation process ends. From this representation, the shortest path from any point q to source s is computed by using a 'backtracing' algorithm.

3.1. Point and segment sources

A point can be considered as a degenerated segment whose endpoints coincide and consequently, an algorithm for computing the distance function of a segment source can be used for the special case of a point source. Consequently, we center our study on segment sources. We start by providing some properties of shortest paths for segment sources and next explain the algorithm steps.

Lemma 1. *Shortest paths from a segment source s to any destination point q fulfill the following four properties:*

(1) *In the interior of a face a geodesic is a straight line.*
(2) *When two neighboring faces are unfolded in a common plane, a geodesic becomes a straight line.*
(3) *A geodesic can only go through a vertex v if it is a boundary vertex or if its total angle is bigger than* 2π.
(4) *A geodesic starting at an interior point of s is orthogonal to s in f, where f is a face of* $\mathscr{P}$ *containing s.*

Proof. The shortest path distance function defined by a segment source s is given by

$$D_s(q) = min_{p \in s} D_p(q).$$

Since s is a closed interval and $D_s(q)$ is a continuous function the minimum is reached at a point of s. Consequently these shortest path are shortest paths to point source and necessarily fulfill the properties of shortest paths on triangulated surfaces which correspond to the first three points (Section 1.2.1). Finally the fourth property characterizes the fact that each point is connected with the closest point of segment s, thus a path obeys the properties of shortest paths to segments in the plane. □

To compute the distance function D_s for a segment source s we track together groups of geodesic paths by partitioning the edges of $\mathscr{P}$ into intervals. Geodesic paths that cross an interval go through the same triangles and bend at the same vertices of $\mathscr{P}$. In an initialization step, intervals defining D_s in the triangle(s) containing s are created. Then the distance function is propagated across triangles in a Dijkstra-like sweep in such a way that, over each interval, D_s can be represented in a compact way by using an appropriate codification.

3.1.1. Distance function codification

The distance function and the shortest paths defining it are encoded and propagated by using intervals, subsegments of the edges of $\mathscr{P}$. We use two different types of intervals: (1) intervals associated to the interior of the segment source s; and (2) intervals associated to either an endpoint of s or a vertex of $\mathscr{P}$.

Consider a shortest path from the source segment s to some point q on an edge e, and let us assume that this path does not bend at any mesh vertex. When all the triangles intersecting the path are unfolded in a common plane, the path forms a straight line (Lemma 1). The set of neighboring points of q on e whose shortest paths back to s pass through the same sequence of triangles form: (a) a pencil of rays emanating from an endpoint of s; and (b) a pencil of orthogonal rays emanating from the interior of s (see Fig. 4). In both cases we represent the group of shortest paths over an interval on the edge e.

Assume now that the shortest path from $p \in s$ to q bends at one or more vertices on its way to the source s, and let v be the nearest such vertex to q. Consider the set of neighboring points on e whose shortest paths back to v go through the same strip of triangles. In the unfolding of the strip between e and v, these shortest paths will form a pencil of rays emanating from the *pseudosource* vertex v, as seen in Fig. 4. As before, we represent this group of shortest paths over an interval on the edge e. Consequently, we can provide the following property.

Property 1. *Shortest path emanating from a segment source s can be grouped by using intervals on the edges of* $\mathscr{P}$.

Intervals representing a group of geodesics emanating from a punctual source p (an endpoint of s, a punctual source, or a mesh pseudosource vertex) are called *p-intervals*, and intervals representing a group of geodesics emanating from interior points of s are called *s-intervals*. Intervals not only group shortest paths, but also encode them.

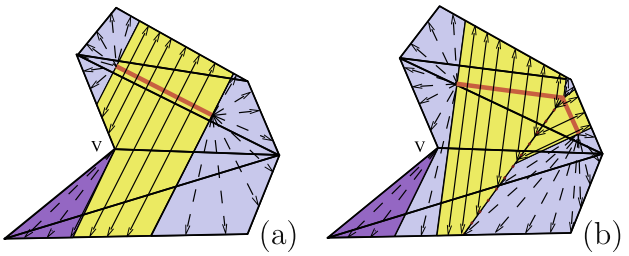

Fig. 4. An unfolded strip of triangles with: (a) a segment source and (b) a polygonal chain source.

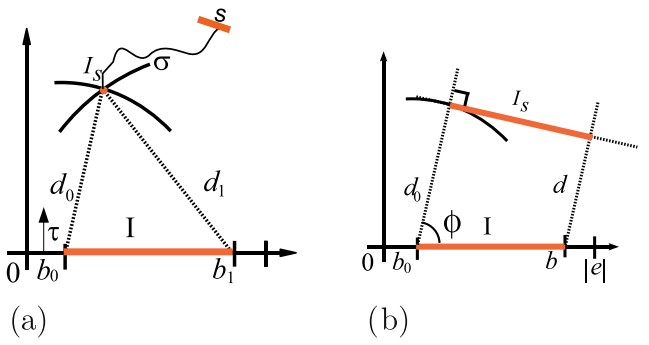

Fig. 5. The virtual source I_s is positioned using the information stored in: (a) a p-interval and (b) a s-interval.

3.1.1.1. p-intervals. The group of geodesics associated to a p-interval I originated at a point p (an endpoint of s or a pseudo-source vertex) is locally encoded by using a 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. Where b_0, $b_1 \in [0, |e|]$ measure the Euclidian distance from the endpoints of I to the origin (the lexicographically smallest endpoint) of e. Distances d_0 and d_1 measure the Euclidean distance from the endpoints of I to p, direction τ specifies the side of e on which p lies, and σ encodes the distance from p to s.

Lemma 2. *A source p defining a p-window can be positioned in $O(1)$ on the plane of a triangle t containing I by using the 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$. The distance function within I can be recovered in $O(1)$ time.*

Proof. Point p is positioned by simulating the planar unfolding adjacent to e in the rectangular coordinate system $\mathscr{S}_e$ that aligns e with the x-axis as it is shown in Fig. 5a). To obtain the position of $p = (p_x, p_y)$ we have to solve the following system:

$$\begin{cases} (p_x - b_0)^2 + p_y^2 = d_0^2, \\ (p_x - b_1)^2 + p_y^2 = d_1^2. \end{cases}$$

The solution is $p_x = \frac{b_1^2 - d_1^2 - (b_0^2 - d_0^2)}{2(b_1 - b_0)}$ and $p_y = \pm\sqrt{d_0^2 - (p_x - b_0)^2}$. The sign of s_y is chosen according to parameter τ. Once the source has been located the distance from source s to a point $q \in I$ is

$$D_s(q) = \|\overline{qp}\| + \sigma.$$

The operations involved in both processes take constant time. □

The point source obtained from $(b_0, b_1, d_0, d_1, \sigma, \tau)$ in the coordinate system $\mathscr{S}_e$ is referred as the *virtual source* of I and noted I^s.

3.1.1.2. s-Intervals. The group of geodesics associated to an s-interval I is locally encoded by using a 5-tuple $(b_0, b_1, d_0, d_1, \phi)$. Where $b_0, b_1 \in [0, |e|]$ are the distances from the endpoints of I to the origin of e, d_0 and d_1 measure the distance of the endpoints of I to s, finally the angle determined by e and the rays emanating from s is stored in $\phi \in [0, 2\pi]$.

Lemma 3. *Given an s-interval, the part s' of s from which geodesics to I emanate can be positioned from the 5-tuple $(b_0, b_1, d_0, d_1, \phi)$ in $O(1)$ time. The distance function within I can be recovered in $O(1)$ time.*

Proof. Segment s' is positioned by simulating the planar unfolding adjacent to e in the rectangular coordinate system $\mathscr{S}_e$ that aligns e with the x-axis as it is shown in Fig. 5b). According to the parameters stored in the 5-tuple and using $\mathscr{S}_e$, the virtual segment source is the segment delimited by point $(b_0 + d_0 \cos\phi, d_0 \sin\phi)$ and point $(b_1 + d_1 \cos\phi, d_1 \sin\phi)$, which are obtained by using basic trigonometry. The distance from s' to a point $q = (q_x, 0) \in I$ is

$$D_s(q) = d(q, s') = d_0 + \frac{(d_1 - d_0)(q_x - b_0)}{b_1 - b_0},$$

this distance is computed by using Tales rule. The operations involved in both processes take constant time. Notice that we do not relay on any trigonometric function for the computation of the distances. □

The obtained segment source is referred again as the *virtual source* of I and noted I^s.

According to Observation 1 shortest paths can be grouped by using intervals, Lemmas 2 and 3 prove that the distance function can be recovered from the information stored in the intervals. Therefore, we can provide the following proposition.

Proposition 1. *Distance function D_s can be encoded by using p-intervals and s-intervals.* □

3.1.2. Interval propagation

We propagate the distance function D_s encoded in an interval on an edge e to the next adjacent triangle t in the unfolding by creating new (potential) intervals on the two opposing edges of t. They are potential intervals because they may overlap previously computed intervals. Consequently, we must intersect the potential interval with previous intervals and determine the combined minimum distance function.

Lemma 4. *The distance function encoded in an interval I can be propagated in $O(1)$ time.*

Proof. Given a p-interval or s-interval I on an edge e, we propagate D_s by computing how the pencil of straight rays representing geodesics associated to I extends across one more unfolded triangle t adjacent to e. New potential intervals can be created on the opposing edges of t (see Fig. 6a and b). To encode D_s in a new potential interval on e' we first obtain the position of the source in the coordinate system $\mathscr{S}_e$. Then, we consider the rays emanating from the source through the endpoints of I to determine the new interval $[b'_0, b'_1]$ on e'. New distances d'_0 and d'_1 from the interval endpoints to the source are computed. For p-intervals σ' does not change and for s-intervals the angle ϕ' is the angle defined by the rays and edge e'. When the interval I is adjacent to a saddle vertex v, geodesics may go through it. Vertex v is a new pseudosource and generates new potential p-intervals with σ' given by d_0 or d_1 when I is an s-intervals and σ' given by $d_0 + \sigma$ or $d_1 + \sigma$ when I is a p-intervals (see vertex v of Fig. 6c).

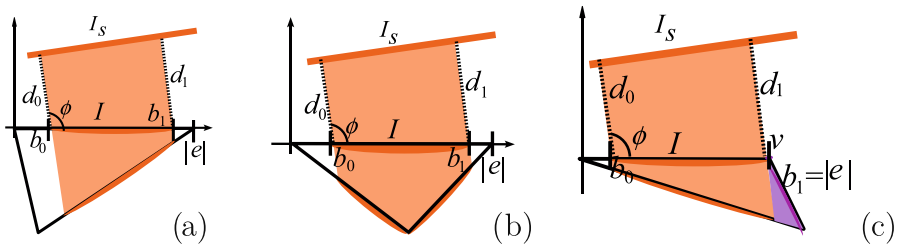

Fig. 6. Examples of s-interval propagation resulting in: (a) a new single interval; (b) two new intervals; and (c) a new single interval and a pseudosource vertex v.

Since I_s is obtained in $O(1)$ time (Lemmas 2 and 3), and computations involved in this process take constant time, the whole cost is $O(1)$. □

3.1.3. Interval overlapping

After the propagation, each new potential interval I' on edge e' may overlap with previously created intervals. Let I be a previously created interval which overlaps I', notice that both I and I' can either be s-intervals or p-intervals. We have to decide which interval defines the minimal distance on the overlapped subsegment $\delta = [b_0, b_1] \cap [b'_0, b'_1]$. In this process, interval I' or I can be deleted or trimmed. The most interesting case is when they are trimmed but not deleted. To correctly obtain the intervals we have to compute the point q in δ where the two distance functions coincide. Notice that in this stage we discard those geodesics encoded in I and I' that cannot be shortest paths.

Lemma 5. *The overlapping of two intervals is computed in $O(1)$ time.*

Proof. To obtain the point q where the distance functions defined by I and I' coincide, we define the rectangular coordinate system $\mathscr{S}_{e'}$ that aligns e' with the x-axis. In this coordinate system point q is $(q_x, 0)$, and let the virtual sources be $I^s = (i_x, i_y)$ and $I'^s = (i'_x, i'_y)$. To obtain q we have to solve the equation obtained when we make that the distance functions coincide at q. Depending on whether I and I' are p-intervals or $s-$intervals, we have to distinguish between the three following cases.

- Two p-intervals: with $I = (d_0, d_1, b_0, b_1, \sigma, \tau)$ and $I' = (d'_0, d'_1, b'_0, b'_1, \sigma'\tau')$. We solve the equation $|\overline{I^s q}| + \sigma = |\overline{I'^{sq}}| + \sigma'$, or equivalently

$$\sqrt{(i_x - q_x)^2 + i_y^2} + \sigma = \sqrt{(i'_x - q_x)^2 + i'^2_y} + \sigma'.$$

  This equation can be reduced to a quadratic equation with a single root in δ. This equation is $Aq_x^2 + Bq_x + C = 0$ with $A = \alpha^2 - \beta^2$, $B = \gamma\alpha + 2i'_x\beta^2$ and $C = 1/4\gamma^2 - \|I'^s\|^2\beta^2$, where $\alpha = i'_x - i_x$, $\beta = \sigma' - \sigma$ and $\gamma = \|I^s\|^2 - \|I'^s\|^2 - \beta^2$.
- An s-interval and a p-interval: with $I = (d_0, d_1, b_0, b_1, \phi)$ and $I' = (d'_0, d'_1, b'_0, b'_1, \sigma', \tau')$ an s-interval and a p-interval, respectively. In this case I^s is a virtual segment source and I'^s a virtual punctual source. We solve the equation $d(I^s, q) = |\overline{I'^{sq}}| + \sigma'$, where $d(I^s, q)$ is the Euclidean distance from q to segment I^s, using the correspondent formulas we obtain:

$$d_0 + \frac{d_1 - d_0}{b_0 - b_1}(q_x - b_0) = \sqrt{(i'_x - q_x)^2 + i_y^2} + \sigma'.$$

  This equation can be reduced to the quadratic equation $Dq_x^2 + Eq_x + F = 0$ with a single solution in δ. The quadratic equation coefficients are $D = \eta^2 - 1$, $E = 2(\eta\kappa + i'_x)$, $F = \kappa^2 - \|I'^s\|$, where $\eta = \frac{d_1 - d_0}{b_0 - b_1}$ and $\kappa = (d'_0 - \eta b'_0 - \sigma')$.
- Two s-intervals: with $I = (d_0, d_1, b_0, b_1, \phi)$ and $I' = (d'_0, d'_1, b'_0, b'_1, \phi')$. We have two virtual segment sources I^s and I'^s and the equation we have to solve is $d(I^s, q) = d(I'^s, q)$, consequently we obtain the equation:

$$d_0 + \frac{d_1 - d_0}{b_0 - b_1}(q_x - b_0) = d'_0 + \frac{d'_1 - d'_0}{b'_0 - b'_1}(q_x - b'_0),$$

  which is a linear equation with a single solution $q_x = \frac{\eta b_0 - d_0 - (\eta' b'_0 - d'_0)}{\eta - \eta'}$ where $\eta = \frac{d_1 - d_0}{b_0 - b_1}$ and $\eta' = \frac{d'_1 - d'_0}{b'_0 - b'_1}$.

In all three cases the virtual sources are determined in constant time (Lemmas 2 and 3), and the solution of the obtained equations is computed in $O(1)$ time. Consequently, the overlapping between two intervals takes $O(1)$ time. □

Notice that in this process some already existent intervals can be deleted but only two already existent intervals can be trimmed but not deleted.

3.1.4. Continuous Dijkstra propagation strategy

The algorithm uses a Dijkstra-like propagation strategy. In the initialization step, we create intervals encoding the distance function on the edges of the triangle(s) containing the segment source s. If the closest point of s to point $q \in \mathscr{P}$ is

an interior point of s, point q is contained in an s-interval. On the other hand, if the closest point of s to q is an endpoint of s, point q is contained in a p-interval.

When intervals are created they are stored in a priority queue which contains both s-intervals and p-intervals. Intervals are stored by increasing distance to the source. The minimum distance from an s-interval to s is $min(d_0, d_1)$. For p-intervals we use $min(d_0, d_1) + \sigma$ as their weight in the priority queue, even though it may not be the minimal distance. This can be done because the solution obtained does not depend on the order in which intervals are removed from the queue, however, by using the priority queue a wavefront is simulated and the process is accelerated.

The first interval of the priority queue is selected, deleted and propagated. Next, overlappings are checked, intersections are computed and the new intervals are added to the priority queue. Notice that when there is an overlap, intervals may be modified or deleted and the priority queue has to be updated accordingly.

Now, we present some lemmas to determine the time and space complexity needed to obtain the shortest path distance functions from a segment source. We first bound the number of intervals generated on each mesh edge. Remember that we work with a non-convex polyhedral surface $\mathscr{P}$ represented as a mesh consisting of n triangles and a segment source placed on $\mathscr{P}$.

Lemma 6. *The algorithm creates, at most, $O(n)$ intervals per mesh edge.*

Proof. Assume that on the edge e there are $\hat{n}$ intervals. Consider $\hat{n}$ shortest paths starting at an interior point of each interval of e and arriving at s. There may be shortest paths arriving at interior points of s and others at the endpoints of s. We sort the $\hat{n}$ shortest paths clock-wise around e and consider pairs of consecutive shortest paths. There may exist at most four pairs of consecutive shortest paths that traverse exactly the same triangles. This may only happen when: (1) one path arrives at an interior point of s and the other to an endpoint of s; and (2) the first path arrives at an endpoint of s and the second path to the other endpoint of s. The other pairs of consecutive shortest paths are associated to a triangle-vertex pair (t, v), where t is the last triangle traversed by both shortest paths, and v is the vertex separating the pair of paths. Observe that the vertex v may be the first pseudosource of one of the shortest paths. It is not difficult to see that at most two different pairs of shortest paths can be associated to the same (t, v) pair (when v is a pseudosource). Since there exists a bijection between triangle-vertex pairs and edges, $\hat{n} \in O(n)$. □

Lemma 7. *At most $O(n^2)$ intervals can be deleted in the overlapping step.*

Proof. Once an interval is deleted it cannot be propagated and an interval that has been propagated cannot be deleted in the future because it defines the minimum distance at some point. Any interval creates at most two new intervals on the opposite edges. These two new intervals may end up being deleted before being propagated. Thus, according to Lemma 6 there are at most $O(n^2)$ created intervals that can be deleted. □

Theorem 1. *The distance function defined by a segment source can be propagated on a non-convex triangulated surface with n faces in $O(n^2 \log n)$ time and $O(n^2)$ space.*

Proof. At most $O(n^2)$ intervals are created and propagated, thus the time needed to propagate and created them is $O(n^2)$ (Lemmas 4 and 7). The time needed in the overlapping process is $O(n^2)$ because at most $O(n^2)$ intervals are deleted in $O(1)$ time each (Lemma 7), and at most two intervals are trimmed at each step in $O(1)$ time (Lemma 5). This yields a $O(n^2)$ complexity. Consequently, the time complexity for computing the shortest path distance function is bounded by the maximum between the number of created intervals and the time needed to create and maintain the priority queue. In the worst case the total complexity is $O(n^2 \log n)$ time and the $O(n^2)$ space. □

3.2. Polygonal sources

The distance function defined by a polygonal chain is obtained by simultaneously considering all the segments of the polygonal chain in the initialization step. For each segment s of the polygonal chain we create potential s-intervals in the triangle(s) containing s and for each vertex we create potential p-intervals. We handle one segment/point conforming the polygonal line after the other, new potential intervals are intersected with the already created ones to ensure that they define the actual distance function. The other parts of the algorithm do not need changes.

The distance function defined by a polygonal region r, a connected region of $\mathscr{P}$ whose boundary is a closed polygonal chain, is the distance function defined by its boundary in the complementary of r, and is 0 in the interior of r. We compute the distance function defined by its boundary polygonal chain creating, in the initialization step, intervals only in the complementary of r. From now on $\tilde{n}$ denotes the number of segments conforming the generalized source s, it will be different from 1 when s is a polygon or polygonal line.

Lemma 8. *At most $O(n + \tilde{n})$ intervals per mesh edge are created.*

Proof. We proceed as in the proof of Lemma 6. Now we have at most four shortest paths traversing the same edges and faces for each segment conforming the polygon or polygonal source. Consequently the number of intervals is now $O(n + \tilde{n})$. □

Putting together this lemma and the previous observations on how we have to modify the algorithm to compute shortest paths from segment sources allow us to provide the following theorem.

Theorem 2. *The distance function defined by a polygonal or polygon source can be propagated on a non-convex triangulated surface with n faces in* $O(n(n+\tilde{n})\log(n+\tilde{n}))$ *time and* $O(n(n+\tilde{n}))$ *space.*

Proof. The algorithm provided to compute shortest paths from a segment source works with minor changes. These changes do not affect the time or space complexity of the algorithm which is again determined by the number of created intervals. Lemma 8 bounds the number of created intervals by $O(n(n+\tilde{n}))$, and consequently the time complexity of the algorithm is $O(n(n+\tilde{n})\log(n(n+\tilde{n})))$ which equals $O(n(n+\tilde{n})\log(n+\tilde{n}))$. The space complexity is $O(n(n+\tilde{n}))$. □

4. Implicit distance field computation

The algorithm for computing the distance function from a generalized source extends naturally to the case of several sites. In this case we obtain a generalized *distance field*, which for any point of $\mathscr{P}$ gives the shortest path distance to its nearest site. Notice that the implicit distance field provides the implicit generalized Voronoi diagram.

In the initialization step we generate intervals for each single site and store them in a unique priority queue. Thus, we propagate the distance functions defined by all the sites simultaneously. This way we obtain a codification of the distance field that yields an implicit representation of the Voronoi diagram of the set of generalized sites.

From now on we denote by $\tilde{r}$ the total number of segments conforming the set S of generalized sites.

Theorem 3. *The implicit distance field of a set of r generalized sites can be obtained in* $O(n(n+\tilde{r})\log(n+\tilde{r}))$ *time and* $O(n(n+\tilde{r}))$ *space.*

Proof. When the distance field of a set of generalized sites is computed, the maximum number of created intervals per edge is $O(n+\tilde{r})$, this can be proved similarly to Lemma 8. It gives at most $O(n(n+\tilde{r}))$ created and deleted intervals providing a $O(n(n+\tilde{r})\log(n+\tilde{r}))$ time and $O(n(n+\tilde{r}))$ space complexity. □

5. Distance and shortest path computation

When the propagation of the distance function has concluded, the distance from any point of a triangle of $\mathscr{P}$ to the source can be obtained from the implicit distance function. If we are given a set of sites S, we can compute the shortest path distance to the closest site and the actual path but using the same technique by using the implicit distance field of S instead of the distance function of a source s.

5.1. Influence regions

To make the computation of the distance and shortest paths easier, we first determine which points of $\mathscr{P}$ can be reached by the geodesics encoded in an interval.

Let I be an interval on a mesh edge e and t be a triangle adjacent to e. We define the *influence region of interval I*, denoted R_I, as the set of point of t that can be reached by geodesics encoded in I. According to the geodesic properties, a point $q \in t$ is reached by a geodesic associated to I in the planar unfolding adjacent to e when: (1) the triangle t and the virtual source of I, I^s, are placed in opposite sides of e; and (2) the point q belongs to a line emanating from I^s or orthogonal to I^s depending on whether I is a p-interval or s-interval, respectively. Therefore, each interval I defines a unique influence region R_I which is a polygon of at most five vertices contained in one of the two adjacent triangles to e.

When I has an endpoint in a saddle vertex v of t, the interval I can also define a pseudosource. The points of t that are not contained in R_I and that can be reached by a line segment emanating from the pseudosource v without intersecting R_I determine the *influence region of pseudosource v*, P_v, which is a convex polygon of at most three vertices.

Lemma 9. *The influence region* R_I *of an interval can be computed in* $O(1)$ *time.*

Proof. To compute the influence region we have to propagate interval I and determine the convex hull defined by the endpoints of the obtained intervals. Since the propagation is done in $O(1)$ time (Lemma 4) and since R_I has at most five vertices, region R_I can be obtained in $O(1)$. □

5.2. Distance computation

The shortest path distance from any point q on a triangle t to the source s can be obtained by finding the interval I_m on the edges of t or the pseudosource v defining the minimum distance value.

If D_I denotes the distance function defined by the interval I in R_I, we have $D_I(q) = D_s(q') + |\overline{qq'}|$, where q' is a point on I such that the line segment $\overline{qq'}$ from q' to q is contained in a geodesic emanating from I^s. Notice that to determine I_m, those intervals whose influence area do not contain q' can be directly discarded. Therefore, we only take into account an interval I on an edge e if its virtual source I^s and triangle t are located in different sides of e, and $q \in R_I$. If q belongs to the influence region of a pseudosource v, the distance function defined by v, $D_v(q) = D_s(v) + |\overline{qv}|$, needs also to be considered. If we denote Ω the set of not discarded intervals and the possible pseudosources, we have that $D_s(q) = \min_{\iota \in \Omega} D_\iota(q)$. Thus we claim the following proposition.

Proposition 2. *The length of the shortest path from a point of $\mathscr{P}$ to the source s or to its nearest source in S can be obtained by standard methods in $O(n + \tilde{n})$ and $O(n + \tilde{r})$ time, respectively.* □

5.3. Shortest path computation

The shortest path from any point q on a triangle t to the source s can be obtained by using a backtracing technique.

Proposition 3. *The shortest path from a generalized source s to point p can be obtained in $O(n + \tilde{n} + \overline{n})$ time, where $\overline{n}$ is the number of crossed triangles. The shortest path to the closest site of S is obtained in $O(n + \tilde{r} + \overline{n})$ time.*

Proof. We first determine the element, interval or vertex, ι_m defining the minimum distance to q. There exist two possibilities: (1) If ι_m is an interval I_m, we jump to the adjacent triangle $t\prime$ by using the direction τ when I_m is a p-interval or the angle ϕ when I_m is an s-interval; and (2) If ι_m is a pseudosource, it is an endpoint of an interval I_m which is used to jump to the adjacent triangle. Then, we keep on jumping to the adjacent triangle until we get s.

According to Proposition 2 the distance can be obtained in $O(n + \tilde{n})$ or $O(n + \tilde{r})$ time. Once ι_m is obtained we jump to a previous interval in constant time by locating the virtual source ι_m^s and finding the new interval when ι_m is an interval, or checking the adjacent edges to the virtual source ι_m otherwise. Both searchings can be done in constant time. □

6. Explicit distance function computation

In this section we provide a way to obtain a discrete representation of the distance function by using the methods given to obtain implicit distance functions.

6.1. Distance vectors

The distance vector $\overrightarrow{d_q}$ of a point q with respect to a virtual source I^s (a point or a segment) is the vector joining the closest point to q of I^s with q.

Distance vectors are useful to compute distances due to the fact that $d(I^s, q) = |\overrightarrow{d_q}|$. Consequently, if we consider an interval I and a point $q \in R_I$, distance $D_I(q)$ is given by $D_s(I^s) + |\overrightarrow{d_q}|$.

Next, we provide some properties that allow us to compute distances vectors from I^s to points on R_I by using distance vectors from the vertices of R_I to I^s when R_I has been triangulated.

6.1.1. Punctual source

Consider a triangulation of R_I and assume that the triangle containing q has vertices p_1, p_2, p_3. Denote $\overrightarrow{d_1}$, $\overrightarrow{d_2}$, $\overrightarrow{d_3}$ the distance vectors of p_1, p_2, p_3 associated to a punctual virtual source I^s. Point q can be univocally expressed as $q = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$, with $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_1, \alpha_2, \alpha_3 \geqslant 0$. Consequently $\overrightarrow{d_q} = \overrightarrow{I^s q} = q - I^s$. Since $q - I^s = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 - (\alpha_1 + \alpha_2 + \alpha_3) I^s$ the following equality is fulfilled $\overrightarrow{d_q} = \alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2} + \alpha_3 \overrightarrow{d_3}$ (see Fig. 7a). Thus the distance vector $\overrightarrow{d_q}$ can be obtained by using linear interpolation from $\overrightarrow{d_1}$, $\overrightarrow{d_2}$, and $\overrightarrow{d_3}$. This is summarized in the following lemma.

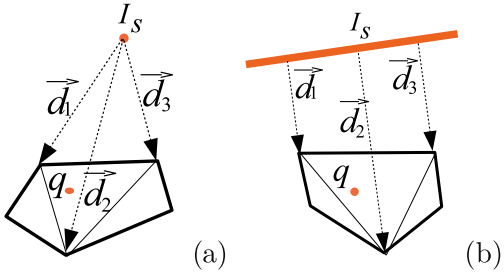

Fig. 7. The distance vectors associated to the vertices of a convex polygon when dealing with: (a) a point source and (b) a segment source. In both figures the distance vector to point q can be obtained by the distance vectors shown.

Lemma 10. *The distance vector from a punctual source to a point q in the influence region R_l can be obtained by using linear interpolation from the distance vectors associated to the vertices of R_l.* □

6.1.2. Segment source

Assume that R_l has been triangulated and that p_1, p_2, p_3 are the vertices of the triangle containing the point q of R_l. Denote $\overrightarrow{d_1}, \overrightarrow{d_1}, \overrightarrow{d_3}$ the distance vectors of p_1, p_2, p_3 relative to a segment virtual source I^s, which are parallel and orthogonal to I^s. As before point q can be univocally expressed as $q = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3$, with $\alpha_1 + \alpha_2 + \alpha_3 = 1$ and $\alpha_1, \alpha_2, \alpha_3 \geqslant 0$. We want to prove that $\overrightarrow{d_q} = \alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2} + \alpha_3 \overrightarrow{d_3}$ (see Fig. 7b).

Let us first consider the case where one α_i is equal to 0, for example $\alpha_3 = 0$. Then, the vector $\alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2}$ is the vector perpendicular to I^s obtained by joining the point q to $\alpha_1 p'_1 + \alpha_2 p'_2$ where p'_1 and p'_2 are the points of I^s defining $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$, respectively. When $\alpha_i \neq 0$, $i = 1, 2, 3$, we can consider a line joining q and p_3, and determine the point q' on the segment for which the parameter $\alpha_3 = 0$. The distance vector $\overrightarrow{d_{q'}}$ can be obtained from $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ as explained for the first case and, finally, $\overrightarrow{d_q}$ can be similarly obtained from $\overrightarrow{d_{q'}}$ and $\overrightarrow{d_3}$. This is summarized in the following lemma.

Lemma 11. *The distance vector from a segment source to a point q in the influence region R_l of an interval can be obtained by using linear interpolation from the distance vectors associated to the vertices of R_l.* □

6.2. Process overview

A *planar parametrization* of $\mathscr{P}$ is a bijective function that maps $\mathscr{P}$ into a bounded region of the plane $\mathscr{R}$. When the parameterization maintains the triangles shape, the discretization error in the *xy*-plane matches with the discretization error on $\mathscr{P}$ [1,8,14]. We use a planar parametrization that maps $\mathscr{P}$ to a bounded planar region $\mathscr{R}$ on the *xy*-plane. For the special case of a polyhedral terrain, which is a polyhedral surface that its intersection with any vertical line is either empty or a point, we can alternatively use the terrain projection on the *xy*-plane as a planar parametrization. We represent the axis-parallel rectangular region bounding the terrain projection on the *xy*-plane by $\mathscr{R}$. In this case the shape of the projected triangles do not correspond with its shape on the polyhedral terrain.

We discretize the rectangular region $\mathscr{R}$ of the *xy*-plane as a rectangular grid of size $W \times H$ that induces a discretization on the triangles of $\mathscr{P}$. This discretization and the planar parametrization is used to obtain a discrete representation on $\mathscr{R}$ of the distance function defined by shortest paths on $\mathscr{P}$. This is achieved by keeping track of the explicit representation of the distance function while it is propagated along the surface $\mathscr{P}$.

For each interval I we compute the distance it defines to the grid points contained in its influence region(s) R_l. We also consider influence regions R_v defined by pseudosources and compute the distance defined by v in R_v. In Section 6.3 we explain how these distances are obtained by using a planar parametrization of the polyhedral surface, distance vectors and graphics hardware. Notice that grid points may be contained in the influence region of different intervals, thus, during the process we may obtain several distances for the same grid point. Since we are interested in the shortest path distance, only the minimal one is stored.

The planar parameterization maps the points on $\mathscr{P}$ to points on the rectangular region $\mathscr{R}$ of the *xy*-plane. The OpenGL pipeline triangulates input polygons, processes the triangle vertices and rasterizes the triangles into fragments by using interpolation. The vertex shader uses the planar parameterization to map the points on $\mathscr{P}$ to points on the rectangular region $\mathscr{R}$ of the *xy*-plane. Within the rasterization step all the parameters associated to vertices such as texture coordinates, color, normal vectors, etc. are also interpolated from those associated to the triangle vertices. Consequently, the value obtained in these channels in a fragment is the interpolation of the values associated to the triangulated polygon vertices. Then the fragment shader computes the distance defined by the current interval or pseudosource at each point. It also sets this distance normalized into [0,1] as the depth value of the rasterized fragments. Finally, the depth test keeps the minimum distance obtained at each position.

6.3. Discrete distance function computation

To compute the discrete distance function we represent the rectangular region $\mathscr{R}$ by a grid of $W \times H$ pixels, using the depth buffer.

The discrete representation of the distance function is obtained during the distance function propagation process. In the initialization process of the propagation method used to compute the implicit distance function, we initialize the depth buffer to the maximal depth value (1). When an interval I is propagated or a pseudo-source is found we compute its distance function in it influence region R_l. This is done by mapping R_l into the plane by using the planar parametrization obtaining PR_l. Next, the planar region PR_l representing R_l is rendered and the distances to the pixels it covers are computed by using a fragment shader. The distance given by I in a point $q \in R_l$ is obtained by adding the distance from q to the virtual source I^s and the distance from s to I^s. The distances to these points are computed and stored in the depth value of the fragments. The depth buffer is updated whenever we still have not obtained a smaller distance value. At the end of the process the value stored in the depth buffer is the minimum depth (distance) defined by all the processed fragments.

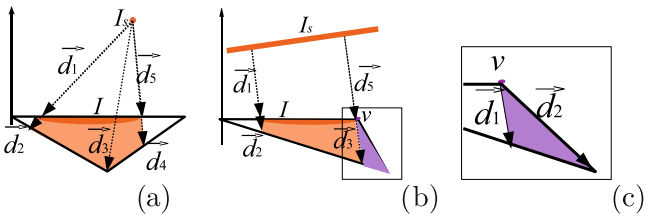

Fig. 8. Examples of influence regions and distance vectors of: (a) a *p*-interval; (b) a *s*-interval; and (c) the vertex pseudosource obtained in (b).

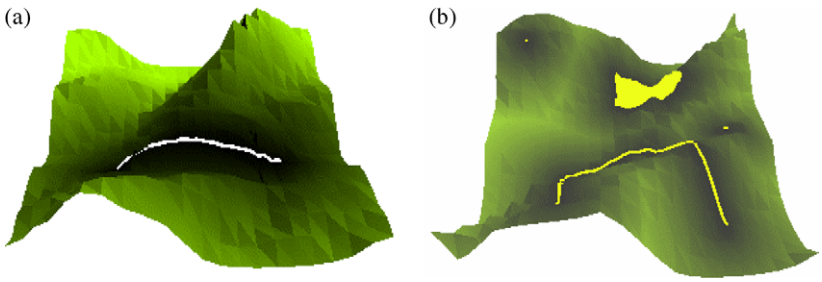

Fig. 9. (a) The distance function defined by a segment source *s*. (b) The distance field defined by a set of four generalized sites.

The previous process correctly computes the distance values: (a) observations related to the influence regions given in Section 5.1 show that we will consider the actual shortest path. (b) Its length or cost is correctly computed according to Lemmas 10 and 11. (c) The minimum distance value is stored in the depth buffer. Consequently, the shortest path distance function is correctly computed and the algorithm is correct.

Consequently, the discrete representation of the distance function can be exactly obtained by using graphics hardware and the algorithms to compute the distance function from a generalized source.

The propagation of interval I defines the influence region R_I and sometimes a new pseudosource v is created (see Fig. 8). The influence region P_v of the pseudosource v is handled in the same way that interval influence regions.

Lemma 12. *The distance field defined by an interval I in its influence region R_I can be computed in O (number of pixels in R_I).*

Proof. To compute the distance in R_I, we compute the distance vectors from the virtual source I^s to the vertices of R_I. Posteriorly the polygon PR_I is rendered by using OpenGL. We associate to the polygon, in texture coordinate channels, the distance from the virtual source to the initial source. We also associate to each vertex of PR_I its distance vector using another texture coordinate channel. The influence regions P_v of pseudosources are also considered and handled in the same way. At the end of the process, we have in the depth buffer the distance function defined by the source. □

Therefore, a discretization of the distance function is obtained by rendering all the influence regions, one after the other and storing the minimal distance value at each pixel.

Theorem 4. *A discrete representation of the distance function of a generalized source s is obtained in $O(n^2 \log n + nHW)$ time, which represents an extra time of $O(nHW)$.*

Proof. The previously explained process to obtain the discrete representation of the distance field does not increase the time of the Continuous Dijkstra. In fact, for each interval I we have to compute the influence regions and the distance vectors from the virtual source I^s to the influence region vertices, which can be done in constant time.

Since each influence region R_I is contained in a face f, the extra time needed to render the influence regions can be bounded as follows. We have at most n intervals per edge and each face is rendered, at most, $O(n)$ times. The extra time needed to render all the influence regions in the non-weighted case is $O(nHW)$ time. □

7. Voronoi diagrams and facility location

In this section we present algorithms, based on discretized distance functions, for approximately computing discrete k-order Voronoi diagrams and approximately solving some facility location problems (1-Center and 1-Median) involving a set of generalized sites on polyhedral surfaces.

In the results, we can distinguish among two different types of error. The discretization error, which depends on the discretization size. The biggest the grid size $W \times H$ we use, the smallest the error. The floating errors, which are specially related to the depth buffer and depth texture precision. The 32-bit precision is sufficient to store the normalized distances which take values in the interval $[0, 1]$.

7.1. Discrete closest Voronoi diagrams

In Section 7.3 a general procedure to compute k-order Voronoi diagrams is given, however, for the special case of the closest Voronoi diagram ($k = 1$) here we present a process which uses the implicit distance field.

To obtain the closest Voronoi diagram, we discretize the generalized Voronoi function and properly determine the Voronoi regions which are given by the distance field (Section 4). The discretization of the generalized Voronoi function is obtained by slightly modifying the algorithm for computing discrete distance functions described in Section 6. We use the planar parametrization of the triangulated surface $\mathscr{P}$ and a discretized xy-plane. We propagate the Voronoi function defined by all single sources. The Voronoi function is discretized by painting the corresponding influence regions. To this purpose the process is modified as follows. We associate a different color to each source in S and the influence regions are colored with the color of the generalized source from where they come from. To identify this source, each interval stores an extra parameter that gives the index of the initial source in S. At the end of the process, the values stored in the depth buffer are the values of the Voronoi function and the obtained image in the color buffer is the discrete closest Voronoi diagram.

We want to mention that there can exist one or two dimensional regions equidistant to two different sites. One dimensional regions are easily detected for being the boundary separating points of different color. The two dimensional regions are, somehow, lost and will be colored in one or the other color depending on the behavior of the continuous Dijkstra, and the order in which regions are rendered.

7.1.1. Complexity analysis

The extra time needed to obtain the closest Voronoi diagram while using the algorithm for implicitly obtain the distance field is the time spent by rendering the influence regions. Each influence region is rendered in constant time. We have at most $O(\tilde{n})$ intervals per face and the whole domain is rendered $O(\tilde{n})$ times, where $\tilde{n}$ is the maximum of n and the total number of segments conforming the generalized sources. Therefore, the time and space complexity are $O(\tilde{n}^2 \log \tilde{n} + \tilde{n}HW)$ and $O(\tilde{n}^2)$.

7.2. Visualization on the polyhedral surface

Once we have obtained a discrete representation of the Voronoi diagram in the color buffer, we can transfer the values of this buffer to a texture. The texture is an explicit representation of the Voronoi diagram and by using texturing methods the Voronoi diagram can be visualized on the polyhedral surface. Distance functions can also be visualized on the polyhedral surface by transferring the values of the depth buffer to a depth texture. Since distances vary from 0 to 1 we use them to weight the color of the pixels. If the white color is used pixels are colored in a grey gradation from black (distance 0) to white (distance 1) according to the distance function OR DISTANCE FIELD value (see Fig. 9).

7.3. Discrete high order Voronoi diagrams

In this section we describe a way to obtain the discrete k-order Voronoi diagram, $k = 1, \ldots, r-1$, for a set of r generalized sources S on the polyhedral surface $\mathscr{P}$. Since obtaining the distance function of a source is expensive, we assume that we have computed and stored the discrete distance function of each source. After obtaining the discrete representation of a distance function, it can be stored by rendering the depth buffer in a depth texture or transferring it to the CPU.

High order Voronoi diagrams can also be visualized on the polyhedral surface using texturing methods as it is explained in the case of the discrete closest Voronoi diagram.

7.3.1. Closest Voronoi diagram

The closest Voronoi diagram can be obtained by rendering, one after the other, the r distance functions and determining their lower envelope. To render a distance function, we store it in a depth texture and set its value as the depth value of the fragment. The depth test is used to store the smallest depth value, and consequently each pixel is colored in the color of the closest site. Finally, the Voronoi diagram is obtained in the color buffer, and the Voronoi function in the depth buffer.

The time needed to obtain the closest Voronoi diagram of r already computed and stored distance functions is $O(rHW)$ time.

7.3.2. Furthest Voronoi diagram

The furthest Voronoi diagram is obtained as the upper envelope of the distance functions. It is computed by rendering one after the other each distance function in their corresponding colors and storing the distance values as the depth of the fragments. The depth test is used to store, in each pixel, the maximal depth value. Each point is accordingly rendered with the color of the fragment with maximal depth value, which is the color associated to the furthest site to each point. The furthest

Voronoi diagram is obtained in the color buffer and the furthest distances in the depth buffer. The complexity analysis is the same as the one given for the closest Voronoi diagram.

7.3.3. k-order Voronoi diagram

Our algorithm to compute discrete k-order Voronoi diagrams uses a depth peeling technique similar to the one described in [13] to compute k-order Voronoi diagrams of a set of points in the plane. It is a multi-pass algorithm that at every pass 'peels' off one level of the arrangement of distance functions. At each pass all the distance functions are rendered in their corresponding colors and the minimal depth value is stored in the depth buffer. In the first pass the closest Voronoi diagram is obtained. The depth buffer is transferred to a texture and sent to the fragment shader. In the second pass all the distance functions are again rendered. The distance function that is being rendered is compared in the fragment shader with the distance obtained in the previous pass at the current fragment. Only the fragments with distance bigger than the distance obtained in the previous pass are rendered. The others are discarded. Therefore, the values stored in the depth buffer in the second step are the second minimal distance. When this process is repeated k times, the kth-nearest diagram is obtained. The k-order Voronoi diagram can be obtained by overlaying the ith-nearest diagrams, $i = 1 \dots k$, with transparency $1/k$.

To obtain the k-order Voronoi diagram of r discrete distance function are already computed and stored. For each of the k peeling passes, the depth buffer is copied to a texture and the r distance functions are transferred to a texture and rendered. Therefore, the algorithm has $O(krHW)$ time and $O(rHW)$ space complexities.

7.4. Facility location problems

7.4.1. 1-Center

The 1-Center of a set of sites S is the point C that minimizes the maximum distance to the sites. Consequently, it is the center of the minimum enclosing disc defined by the shortest path distance. Thus we have to find the point where $min_{q\in T} max_{s\in S} D_s(q)$ is achieved. Notice that $max_{s\in S} D_s(q)$ is the value stored in the depth buffer when the furthest Voronoi diagram is computed with the algorithm explained in Section 7.3. We take as an approximate 1-Center of S the point of $\mathscr{P}$ corresponding to the position of the depth buffer with minimal value. This value is the radius of the minimum enclosing disc. Once the Furthest Voronoi diagram, the process to obtain the 1-Center takes $O(HW)$ time and space complexity.

7.4.2. 1-Median

The 1-Median of a set of r sites S, is the point that minimizes $\sum_{s\in S} D_s(q)$. We will use the depth buffer to store this sum of distances, since it only stores values within 0 and 1 we will multiply each distance field by $1/r$ to ensure that $\sum_{s\in S} D_s(q)$ does not exceed 1. To obtain $\sum_{s\in S} D_s(q)$ we use r rendering steps. After initializing the depth buffer to 0, at the first step we render D_{s_0} and store its values in the depth buffer. In the second step we transfer the depth buffer to a texture t_0, render D_{s_1} and store in the depth buffer the sum of D_{s_1} with the value stored in t_0. In the ith step we transfer the depth buffer to a texture t_{i-1}, render D_{s_i} and store in the depth buffer the sum of D_{s_i} with the value stored in t_{i-1}. We repeat the process until D_{s_r} is considered. At the end of the process, we have in the depth buffer the desired $\sum_{s\in S} D_s(q)$. We obtain an approximate 1-Median for the set S by determining the pixel where the minimum value is achieved. The process to obtain the 1-Median takes $O(rHW)$ time once we have precomputed all the distance functions.

8. Experimental results

We have implemented the proposed methods using C++ and OpenGL for the special case of polyhedral terrains. All the images have been carried out on a Intel(R) Pentium(R) D at 3 GHz with 1GB of RAM and a GeForce 7800 GTX/PCI-e/SSE2 graphics board. All our GPU algorithms are implemented with Cg language since they map naturally to the parallel computation using shader programs and takes advantage of the OpenGl capabilities.

In Table 1 we present some experimental results, obtained by considering triangulated terrains, a set S of six sites (one point, two segments, two polygonal lines and one polygon), $\epsilon = 0.5$, $\epsilon' = 0.1$ and a grid to discretize the domain of size 500×500. In the table we specify the number of terrain faces, n, and the number of faces intersected by the sites n'. We have

Table 1
Implicit distance computation.

n	D. Field (s)	D. Functions (s)
800	0.3	2.3)
5000	2.3	14
10,000	5.8	39
20,000	13	96
45,000	29	184

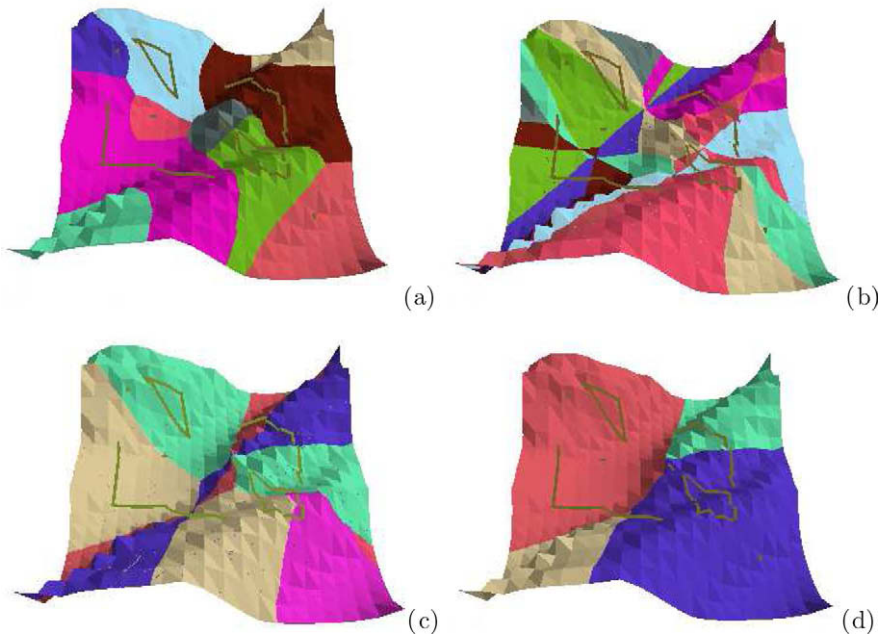

Fig. 10. A terrain, ten generalize sites and their: (a) closest Voronoi diagram; (b) 7th-nearest diagram; (c) 9th-nearest diagram; and (d) furthest Voronoi diagram.

used the Continuous Dijkstra strategy and provided the time needed to compute the distance field and to obtain the explicit Voronoi diagram of S, and next the time needed to compute and store the six distance functions. Notice that the time needed to obtain the six distance functions is about six times that needed to obtain the distance field. The extra time needed to obtain, from the already computed distance fields, the closest or furthest Voronoi diagram is 0.08 s, the 4th nearest site to each point is 0.3 s and the 5th nearest site is 0.35 s. Finally, in 0.1 s we obtain an approximate 1-Center when $r = 6$ and in 0.12 s an approximation of the 1-Median.

Fig. 10 shows Voronoi diagrams on polyhedral terrains obtained by using the Continuous Dijkstra strategy. We have considered a set S of 10 sites: four points, two segments, two polygonal chains and two polygon sources, and a terrain with $n = 800$ faces. In Fig. 10a we show the closest Voronoi diagram of S; in Fig. 10b each point is colored in the color of the 7th nearest site, we do not show the 7th order Voronoi because the image is not easy to understand due to the merged colors. In Fig. 10c the furthest Voronoi diagram is shown.

The error produced by the 32-bit precision of the depth buffer can be seen in Fig. 10b), isolated pixels are colored in the color of the regions adjacent to the region they belong to.

9. Conclusions

We have presented an algorithm for computing exact shortest paths from generalized sources (point, segment, polygonal line and polygon sources) on triangulated polyhedral surfaces. The algorithm takes $O(n(n+\tilde{n})\log(n+\tilde{n}))$ time and $O(n(n+\tilde{n}))$ space, where $\tilde{n}$ is the number of segments conforming the generalized source. The algorithm is extended to the case of several generalized sites when their implicit distance field is obtained in $O(n(n+\tilde{r})\log(n+\tilde{r}))$ time and $O(n(n+\tilde{r}))$ space where $\tilde{r}$ is the total number of segments conforming the generalized sites. The output of algorithm is a codification of the distance function or distance field. From this codification the distance from a point on $\mathscr{P}$ to the site and the actual shortest path can be obtained in $O(n+\tilde{n}+\overline{n})$ time, where $\overline{n}$ is the number of faces the path goes through. We can also obtain the shortest path distance and the actual shortest path to the closest site of a set S in $O(n+\tilde{r}+\overline{n})$ time.

We want to mention that the algorithm of Chen and Han, a typical algorithm for computing exact shortest paths on polyhedral surfaces, can also be adapted to support generalized sites without increasing the time and storage complexity of the original algorithm. This algorithm works when the one angle one split observation holds, and it holds when generalized sites are considered.

Next we present methods to obtain an explicit discrete representation of the implicit distance functions and distance fields previously computed. From the explicit discrete representation of the distance functions we show how the closest, furthest or any k-order Voronoi diagram can be obtained. Notice that we also present a way to directly obtain the closest Voronoi diagram of a set of sites using their distance field. Finally, we also approximately solve some facility location problems such as the 1-Center and 1-Median. Finally we present some experimental results for the particular case of terrains.

Acknowledgement

Support of the Spanish Ministerio de Educación y Ciencia under Grant TIN2007-67982-C02-02 is gratefully acknowledged.

References

[1] Computational Geometry Algorithms Library User and Reference Manual. <http://www.cgal.org/Manual/3.3/dochtml/cgalmanual/contents.html>, 2008.
[2] P.K. Agarwal, S. Krishnan, N. Mustafa, S. Venkatasubramanian, Streaming geometric optimization using graphics hardware, in: 11th European Symposium on Algorithms, 2003, pp. 544–555.
[3] B. Aronov, M.J. van Kreveld, R. van Oostrum, K.R. Varadarajan, Facility location on a polyhedral surface, Discrete Comput. Geom. 30 (3) (2003) 357–372.
[4] F. Aurenhammer, Voronoi diagrams: a survey of a fundamental geometric data structure, ACM Comput. Surv. 23 (3) (1991) 345–405.
[5] F. Aurenhammer, R. Klein, Handbook of Computational Geometry, Elsevier, 2000. pp. 201–290 (Chapter Voronoi diagrams).
[6] P. Bose, A. Maheshwari, P. Morin, Fast approximations for sums of distances, clustering and the Fermat–Weber problem, Comput. Geom. Theory Appl. 24 (3) (2003) 135–146.
[7] S. Cabello, M. Fort, J.A. Sellarès, Higher-order Voronoi diagrams on triangulated surfaces, Information Processing Letters 109 (2009) 440–445.
[8] N. Carr, J. Hart, J. Maillot, The solid map: methods for generating a 2-d texture map for solid texturing, in: Proceedings of the Western Computer Graphics Symposium, 2000, pp. 179–190.
[9] H. Chakroun, G. Goze, B. Benie, O. O'Neill, J. Desilets, Spatial analysis weighting algorithm using voronoi diagrams, Int. J. Geogr. Inform. Sci. 14 (4) (2000) 319–336.
[10] R. Chandrasekaran, A. Tamir, Algebraic optimization: the Fermat–Weber location problem, Math. Program. 46 (2) (1990) 219–224.
[11] J. Chen, Y. Han, Shortest paths on a polyhedron; part i: computing shortest paths, Int. J. Comput. Geom. & Appl. 6 (2) (1996) 127–144.
[12] Q. Fan, A. Efrat, V. Koltun, S. Krishnan, S. Venkatasubramanian, Hardware assisted natural neighbour interpolation, in: Proceedings of the 7th Workshop on Algorithm Engineering and Experiments (ALENEX), 2005.
[13] I. Fisher, C. Gotsman, Fast approximation of high order voronoi diagrams and distance transforms on the GPU, J. Graph. Tools 11 (4) (2006) 39–60.
[14] M.S. Floater, K. Hormann, Surface parameterization: a tutorial and survey, in: N.A. Dodgson, M.S. Floater, M.A. Sabin (Eds.), Advances in Multiresolution for Geometric Modelling, Springer Verlag, 2005, pp. 157–186.
[15] H.W. Guesgen, J. Hertzberg, R. Lobb, A. Mantler, First steps towards buffering fuzzy maps with graphics hardware, in: FOIS Workshop on Spatial Vagueness, Uncertainty and Granularity, FOIS Workshop on Spatial Vagueness, Uncertainty and Granularity, 2001.
[16] K.E. Hoff III, J. Keyser, M. Lin, D. Manocha, T. Culver, Fast computation of generalized Voronoi diagrams using graphics hardware, Comp. Graph. 33 (Annual Conference Series) (1999) 277–286.
[17] B. Kaneva, J. O'Rourke, An implementation of Chen and Han's shortest paths algorithm, in: Proceedings of the 12th Canadian Conference on Computational Geometry, 2000, pp. 139–146.
[18] S. Kapoor, Efficient computation of geodesic shortest paths, in: STOC'99: Proceedings of the 31st Annual ACM Symposium on Theory of Computing, 1999, pp. 770–779.
[19] V. Koltun, Y. Chrysanthou, D. Cohen-Or, Hardware-accelerated from-region visibility using a dual ray space, in: Proceedings of the 12th Eurographics Workshop on Rendering Techniques, Springer-Verlag, London, UK, 2001, pp. 205–216.
[20] S. Krishnan, N.H. Mustafa, S. Venkatasubramanian, Hardware-assisted computation of depth contours, in: SODA'02: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2002, pp. 558–567.
[21] M.A. Lanthier, D. Nussbaum, T.-J. Wang, Calculating the meeting point of scattered robots on weighted terrain surfaces, in: M. Atkinson, F. Dehne (Eds.), Eleventh Computing: The Australasian Theory Symposium (CATS2005), Newcastle Australia, CRPIT, vol. 14, ACS, 2005, pp. 107–118.
[22] Y.-J. Liu, Q.-Y. Zhou, S.-M. Hu, Handling degenerate cases in exact geodesic computation on triangle meshes, Vis. Comp. 23 (9–11) (2007) 661–668.
[23] J. Mitchell, Handbook of Computational Geometry, Elsevier Science Publishers B.V., 2000. pp. 633–701 (Chapter Geometric shortest paths and network optimization).
[24] J.S.B. Mitchell, D.M. Mount, C.H. Papadimitriou, The discrete geodesic problem, SIAM J. Comput. 16 (4) (1987) 647–668.
[25] D.M. Mount, Voronoi diagrams on the surface of a polyhedron, Technical Report, University of Maryland, 1985.
[26] N.H. Mustafa, S. Krishnan, G. Varadhan, S. Venkatasubramanian, Dynamic simplification and visualization of large maps, Int. J. Geogr. Inform. Sci. 20 (3) (2006) 273–302.
[27] A. Okabe, B. Boots, K. Sugihara, S.N. Chiu, Spatial Tessellation: Concepts and Application of Voronoi Diagrams, John Wiley and Sons, 2000.
[28] A. Okabe, B. Boots, K. Suihara, Nearest neighbourhood operations with generalized voronoi diagrams: a review, Int. J. Geogr. Inform. Sci. 8 (1) (1994) 43–71.
[29] J.D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Krger, A.E. Lefohn, T.J. Purcell, A survey of general-purpose computation on graphics hardware, Comp. Graph. Forum 26 (1) (2007) 80–113.
[30] T.J. Purcell, I. Buck, W.R. Mark, P. Hanrahan, Ray tracing on programmable graphics hardware, ACM Trans. Graph. 21 (3) (2002) 703–712. ISSN 0730-0301 (Proceedings of ACM SIGGRAPH 2002).
[31] Y. Schreiber, M. Sharir, An optimal-time algorithm for shortest paths on a convex polytope in three dimensions, in: SCG'06: Proceedings of the 22nnd Annual Symposium on Computational Geometry, ACM Press, New York, NY, USA, 2006, pp. 30–39.
[32] M. Segal, K. Akeley, The OpenGL Graphics System: A Specification. <http://www.opengl.org/documentation/specs/version2.0/glspec20.pdf>, 2004.
[33] C. Sigg, R. Peikert, M. Gross, Signed distance transform using graphics hardware, in: VIS'03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03), 2003, pp. 83–90.
[34] A. Sud, M. Otaduy, D. Manocha, DiFi: fast 3D distance field computation using graphics hardware, in: Eurographics, vol. 23, 2004, pp. 557–566.
[35] V. Surazhsky, T. Surazhsky, D. Kirsanov, S.J. Gortler, H. Hoppe, Fast exact and approximate geodesics on meshes, ACM Trans. Graph. 24 (3) (2005) 553–560.
[36] E. Weiszfeld, Sur le point pour lequel la somme des distances de n points donnes est minimum, Tohoku Math. J. 43 (1936) 355–386.