Generalized Higher-Order Voronoi Diagrams on Polyhedral Surfaces

Marta Fort* J. Antoni Sellarès*

Abstract

We present an algorithm for computing exact shortest paths, and consequently distances, from a generalized source (point, segment, polygonal chain or polygonal region) on a possibly non-convex polyhedral surface in which polygonal chain or polygon obstacles are allowed. We also present algorithms for computing discrete Voronoi diagrams of a set of generalized sites (points, segments, polygonal chains or polygons) on a polyhedral surface with obstacles. To obtain the discrete Voronoi diagrams our algorithms, exploiting hardware graphics capabilities, compute shortest path distances defined by the sites.

1. Introduction

In this paper we present algorithms for computing discrete Voronoi diagrams of a set of generalized sites (points, segments, polygonal chains or polygons) on a possibly non-convex polyhedral surface with obstacles. The algorithms are based on the efficient discretization, obtained by using graphics hardware, of the shortest path distance functions defined by the sites on the polyhedral surface.

1.1. Preliminaries

Let $\mathcal{P}$ be a, possibly non-convex, polyhedral surface represented as a mesh consisting of n triangular faces. From now on, a generalized element on $\mathcal{P}$ refers to a point, segment, polygonal chain or a polygon. We model obstacles in $\mathcal{P}$ by a set of non-punctual generalized elements on $\mathcal{P}$.

A shortest path between a generalized source and a point on $\mathcal{P}$ is a shortest path in the Euclidean metric between the source and the point such that the path stays on $\mathcal{P}$ and avoids the obstacles. The shortest path distance function defined by a source point p on $\mathcal{P}$ is a function D_p such that for any point $q \in \mathcal{P}$, $D_p(q)$ is the Euclidean length of the shortest path along $\mathcal{P}$ from q back to point p. The shortest path distance function defined by a generalized source s is a function D_s such that for any point $q \in \mathcal{P}$, $D_s(q)$ is the length of the shortest path from q back to source s.

Let S be a set of m generalized sites on the polyhedral surface $\mathcal{P}$ and let S' be a subset of k sites of S, $k \in \{1, \cdots, m-1\}$. The set of points of $\mathcal{P}$ closer to each site of S' than to any other site of S, where distances are shortest path distances on $\mathcal{P}$, 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 $k = m-1$ the k-order Voronoi diagram is called the closest Voronoi and the furthest Voronoi diagram, respectively.

1.2. Previous work

We give an overview of previous work on shortest path on polyhedra and generalized Voronoi diagrams computation.

Shortest path problems. Computing shortest paths on polyhedral surfaces is a fundamental problem in computational geometry with important applications in computer graphics, robotics and geographical information systems (for further details see references [10, 16].)

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 $\mathcal{P}$. Mitchell et al. [11] present an algorithm for solving the single point source shortest path problem by developing a "continuous Dijkstra" method which propagates distances from the source to the rest of $\mathcal{P}$, for the case in which obstacles are not allowed. The algorithm constructs a data structure that implicitly encodes the shortest paths from a given source point to all other points of $\mathcal{P}$ 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 $O(\log n + n')$ time respectively, n' is the number of mesh edges crossed by the path. Different improvements of this algorithm have been proposed. Surazhsky et al. [16] described a simple way to implement the algorithm and showed to run much faster on most polyhedral surfaces than the $O(n^2 \log n)$

*Institut d'Informàtica i Aplicacions, Universitat de Girona, Spain, {mfort,sellares}@ima.udg.edu. Partially supported by grant TIN2004-08065-C02-02. Marta Fort is also partially supported by grant AP2003-4305.

4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)
0-7695-2869-4/07 $25.00 © 2007 IEEE

theoretical worst case time. Chen and Han [4], 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 $O(\log n + n')$ time. Kaneva and O'Rourke [8] 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 [9] 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. However, this is a difficult algorithm to implement.

Computation of 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 [1, 2, 12].

Practical and robust 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 use to be complicated because involve the 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 distance function based algorithms have been proposed to compute 2D and 3D discretized closest Voronoi diagrams along a grid using graphics hardware [5, 7, 13, 15]. 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.

1.3. Our contribution

In this paper we present an algorithm for computing exact shortest paths, and consequently distances, from a generalized source on a possibly non-convex polyhedral surface $\mathcal{P}$ with obstacles. More specifically:

- We extend the ideas developed by Surazhsky et al. [16] for the implementation of the algorithm of Mitchell et al. [11] to the case of generalized sources and polyhedral surfaces with obstacles. The implementation is based on an implicit codification of shortest path distances on $\mathcal{P}$ (Section 2). The algorithm easily extends to the case of several sources and gives an implicit representation of the closest Voronoi diagram of a set of generalized sites on the surface (SubSection 2.5).
- We present an algorithm for discretizing, by using hardware graphics, the distance function defined by a generalized source on $\mathcal{P}$ (Section 4).

We also present two algorithms, based on distance functions and hardware graphics capabilities, for computing discrete Voronoi diagrams of a set of m generalized sites on the polyhedral surface $\mathcal{P}$ with obstacles:

- The first algorithm is designed specifically for efficiently computing discrete closest Voronoi diagrams (Section 5).
- The second algorithm permits the computation of all discrete k-order Voronoi diagrams, $k = 1, \cdots, m-1$ (Section 6).

2. Implicit distance function computation

A geodesic is a path that is locally a shortest path, thus, shortest paths are all geodesics, but the converse need not hold. Geodesics on non-convex triangulated surfaces have the following characterization [11]: 1) in the interior of a triangle the shortest path is a straight line; 2) when crossing an edge a shortest path corresponds to a straight line if the two adjacent triangles are unfolded into a common plane; 3) shortest paths can go through a vertex if and only if its total angle is at least 2π (*saddle* vertex). The basic idea of the algorithm of Mitchell et al. [11] for solving the shortest path problem from a mesh vertex v, as implemented by Surazhsky et al. [16], is to track together groups of shortest paths by partitioning each triangle edge into a set of intervals (windows) so that all shortest paths that cross a window can be encoded locally using a parameterization of the distance function D_v. After an initialization step, where windows encoding D_v in the edges of the triangles containing v are created, the distance function is propagated across mesh triangles in a "continuous Dijkstra" fashion by repeatedly using a window propagation process. A complete intrinsic representation of D_v is obtained when the propagation process ends. From this representation the shortest path from any point q to the vertex source v is computed by using a "backtracing" algorithm.

Since the shortest path distance function defined by a generalized source s can be computed as $D_s(q) = min_{p \in s} D_p(q)$, the shortest path from the generalized source s to any point destination q has the same characterization as the shortest path between two points that we described previously. Notice that if s' is a subsegment of a generalized source s contained in a triangle t of $\mathcal{P}$, the part of a shortest path interior to t starting at an interior point of s' is orthogonal to s'.

4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)
0-7695-2869-4/07 $25.00 © 2007 IEEE

2.1. Point and segment sources

Since a point can be considered as a degenerated segment whose endpoints coincide, the distance function for a point source can be reduced to the distance function for a segment source. Consequently we center our study only on such sources.

To compute the distance function D_s for a segment source s, as is done in the case of a source point, we track together groups of geodesic paths by partitioning the edges of $\mathcal{P}$ into windows. Geodesic paths that cross a window go through the same triangles and bend at the same vertices of $\mathcal{P}$. In the initialization step, windows 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 window, D_s can be represented compactly using an appropriate parameterization.

2.1.1. Distance function codification. 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. 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; b) a pencil of orthogonal rays emanating from the interior of s (see Figure 1). In both cases we represent the group of shortest paths over a window of the edge e.

Suppose now that the shortest path from $p \in s$ to q bends on one or more vertices on its way to the source s, and let v be the nearest such vertex to q. Again, 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 Figure 1. As before, we represent this group of shortest paths over a window on the edge e.

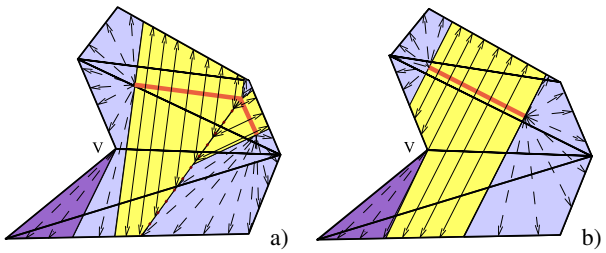

Figure 1. An unfolded strip of triangles with: a) a segment source; b) a polygonal chain source.

Windows 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-*windows*, and windows representing a group of geodesics emanating from interior points of s are called s-*windows*.

p-windows. Following the strategy of Surazhsky et al. [16], the group of geodesics associated to a p-window w originated on a point p (an endpoint of s or a pseudosource 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 w to the origin of e (the lexicographically smallest endpoint of e). Distances d_0 and d_1 measure the Euclidean distance from the endpoints of w to p, direction τ specifies the side of e on which p lies, and σ gives the distance from p to s. From the 6-tuple $(b_0, b_1, d_0, d_1, \sigma, \tau)$ it is easy to position the source p on the plane of a triangle t containing w, and to recover the distance function within w, by simulating the planar unfolding adjacent to e in the rectangular coordinate system $\mathcal{S}_e$ that aligns e with the x-axis as it is shown in Figure 2 a). The obtained point source is referred as the *virtual source* of w and noted w_s.

s-windows. The group of geodesics associated to an s-window w 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 of the endpoints of w to the origin of e, d_0 and d_1 measure the distance of the endpoints of w to s, finally the angle determined by e and the rays emanating from s is stored in $\phi \in [0, 2\pi]$. From the 5-tuple $(b_0, b_1, d_0, d_1, \phi)$ it is easy to position the part s' of s from which geodesics to w emanate (the visible part of s through the unfolding). Again it is done by simulating the planar unfolding adjacent to e in the rectangular coordinate system $\mathcal{S}_e$ that aligns e with the x-axis as it is shown in Figure 2 b). The obtained segment source is referred again as the *virtual source* of w and noted w_s. Notice that we do not need d_1 to position s' but it is useful in order to obtain the distance function within w.

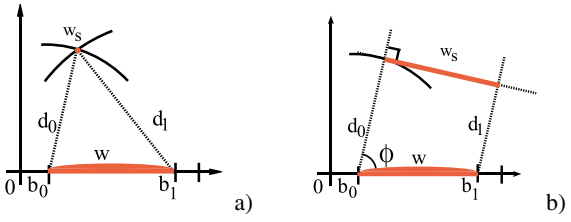

Figure 2. The virtual source s is positioned using the information stored in: a) a p-window; b) a s-window.

2.1.2. Window propagation. We propagate the distance function D_s encoded in a window on an edge e to the next adjacent triangle t by creating new (potential) windows on the two opposing edges of t. They are potential windows

4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)
0-7695-2869-4/07 $25.00 © 2007 IEEE

because they may overlap previously computed windows. Consequently, we must intersect the potential window with previous windows and determine the combined minimum distance function.

Given a p-window or s-window w on an edge e, we propagate D_s by computing how the pencil of straight rays representing geodesics associated to w extends across one more unfolded triangle t adjacent to e. New potential windows can be created on the opposing edges of t (see Figures 3 a and 3 b). To encode D_s in a new potential window e' first, we obtain the position of the source in the coordinate system $\mathcal{S}_{e'}$. Then, we consider the rays emanating from the source through the endpoints of w to determine the new window interval $[b'_0, b'_1]$ on e'. New distances d'_0 and d'_1 from the window endpoints to the source are computed. For p-windows σ' does not change, and for s-windows the angle ϕ' is the angle defined by the rays and edge e'. When the window w is adjacent to a saddle vertex v, geodesics may go through it. Vertex v is a new pseudosource and generates new potential p-windows with σ' given by $d_0(d_0 + \sigma)$ or $d_1(d_1 + \sigma)$ for s-windows(p-windows) (see vertex v of Figure 3 c).

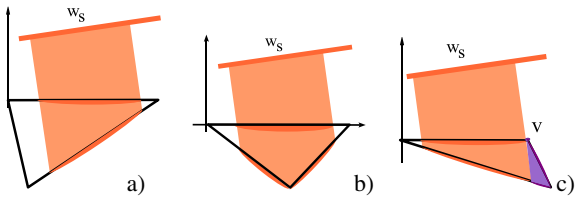

Figure 3. Examples of s-window propagation resulting in: a) a new single window; b) two new windows; c) a new single window and a pseudosource vertex v.

2.1.3. Window overlapping. After the propagation, each new potential window w' on edge e' may overlap with previously created windows. Let w be a previously created window which overlaps with w', notice that both w and w' can be either s-windows or p-windows. We have to decide which window defines the minimal distance on the overlapped subsegment $\delta = [b_0, b_1] \cap [b'_0, b'_1]$. To correctly obtain the windows we have to compute the point q in δ where the two distance functions coincide. We are discarding the geodesics encoded in w and w' that cannot be shortest paths.

In order to obtain q, we define the rectangular coordinate system $\mathcal{S}_{e'}$ that aligns e' with the x-axis. When w and w' are p-windows we obtain the position of their virtual punctual sources w_s and w'_s in $\mathcal{S}_{e'}$ and solve the equation $|w_p - q| + \sigma = |w'_p - q| + \sigma'$, as it is done in [16].When w and w' are an s-window and a p-window respectively, we obtain the virtual segment source w_s and the virtual punctual source w'_s on $\mathcal{S}_{e'}$, and solve $d(w_s, q) = |w'_s - q| + \sigma'$, where $d(w_s, q)$ is the Euclidean distance from q to segment w_s. Finally, when both are s-windows we obtain the position of both virtual segment sources w_s and $w_{s'}$ in $\mathcal{S}_{e'}$ and solve the equation $d(w_s, q) = d(w'_s, q)$. In all three cases the resulting equation has a single solution.

2.1.4. Continuous Dijkstra propagation strategy. The algorithm uses a Dijkstra-like propagation strategy. In the initialization step, we create windows encoding the distance function on the edges of the triangle(s) containing the segment source s. Those points closer to points in the interior of the segment source are contained in s-windows. On the other hand, those points whose closest point of s is an endpoint of s are contained in p-windows. When windows are created, they are stored in a priority queue which contains both s-windows and p-windows. Windows are stored by increasing distance to the source. The minimum distance from an s-window to source s is $min(d_0, d_1)$. For p-windows we use $min(d_0, d_1) + \sigma$ as weight in the priority queue, although it may not be the minimal distance. It can be done because the obtained solution does not depend on the order in which windows are removed from the queue, however, by using the priority queue a wavefront is simulated and the process is accelerated.

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

2.1.5. Complexity analysis. We analyze the time and space complexity needed to obtain the shortest path distance functions from a source segment on a non-convex polyhedral surface $\mathcal{P}$ represented as a mesh consisting of n triangles.

We first give a bound on the number of windows generated on each mesh edge when the distance function from a source segment is computed. A similar result is presented in [11] for source points.

Lemma 2.1 *There are at most $O(n)$ windows per mesh edge created by the algorithm.*

Proof. Assume that on the edge e there are n' windows. Consider n' shortest paths starting at an interior point of each window of e and arriving at s. There may be shortest paths arriving at interior points of s and other at the endpoints of s. We sort the 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. It can only happen when: 1) one path arrives at an interior point of s and the other to an endpoint of s ; 2) when 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

0-7695-2869-4/07 $25.00 © 2007 IEEE

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, we have that $n' \in O(n)$.

For a segment source s, the time needed to compute D_s is $O(n^2 \log n)$. It is not difficult to see that the time complexity for computing the shortest path distance function is bounded by the maximum between the number of created windows and the time needed to create and maintain the priority queue. Consequently, according to Lemma 2.1, at most $O(n^2)$ windows are created and in the worst case the total time complexity is, though, $O(n^2 \log n)$ and the space complexity $O(n^2)$.

2.2. 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-windows in the triangle(s) containing s and for each vertex we create potential p-windows. We handle one segment/point after the other and the new potential windows 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 $\mathcal{P}$ whose boundary is a closed polygonal chain, is the distance function of its boundary in the complementary of r, and is 0 in the interior of r. We compute the distance function produced by its boundary polygonal chain creating, in the initialization step, windows only in the complementary of r.

If we make the logical assumption that the number of segments conforming the polygonal source is smaller than n, the total time and space complexity of this algorithm are again $O(n^2 \log n)$ and $O(n^2)$, respectively.

2.3. Polygonal obstacles

Given a possibly non-convex polyhedral surface $\mathcal{P}$ (which may represent a terrain), we model obstacles as polygonal chains or polygonal regions (which may represent rivers, lakes, etc) on $\mathcal{P}$. The polyhedral surface is retriangulated so that the obstacles are represented as several mesh triangles, edges and vertices. Abusing language, we keep n as the number of triangles of the new mesh. We assume that paths cannot traverse the polygonal obstacles, but we let paths go along them. Now, geodesic paths can go through a vertex not only if it is a saddle vertex but also when it is an obstacle vertex. To compute shortest paths we only need to make two modifications in the window propagation process. On the one hand, windows on an obstacle edge are not propagated. On the other hand, obstacle vertices are new pseudosource vertices regardless of their total angle. These modifications do not increase the time or space complexities of the algorithm.

2.4. Distance and shortest path obtention

When the propagation of the distance function has concluded, the distance and also the shortest path from any point on a triangle of $\mathcal{P}$ to the source can be obtained.

Influence regions. To facilitate the computation of the distance and shortest paths we first determine which points of $\mathcal{P}$ can be reached by the geodesics encoded in a window.

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

When w has an endpoint in a saddle vertex v of t, the window w can also define a pseudosource. The points of t that are not contained in P_w and can be reached by a line segment emanating from the pseudosource v without intersecting P_w, determine the *influence region of pseudosource* v, P_v, which is a convex polygon of at most 3 vertices.

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

If $D_{s,w}$ denotes the distance function defined by the window w, we have $D_{s,w}(q) = D_{s,w}(q') + |q - q'|$, where q' is a point on w such that the line segment of endpoints q' and q is contained in a geodesic emanating from w_s. Notice that to determine w_m, those windows whose influence area does not contain q' can be directly discarded. Therefore, we only take into account a window w on an edge e if its virtual source w_s and triangle t are located on different sides of e and $q \in P_w$. If q belongs to the influence region of a pseudosource v, the distance function defined by v, $D_{s,v}(q) = D_s(v) + |q - v|$, needs also to be considered. If we denote Ω the set of such windows and the possible pseudosource, then we have $D_s(q) = \min_{\omega \in \Omega} D_{s,\omega}(q)$.

The length of the shortest path from a point of $\mathcal{P}$ to

4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)
0-7695-2869-4/07 $25.00 © 2007 IEEE

its nearest source can be obtained by standard methods in $O(\log n)$ time.

Shortest path construction. The shortest path from any point q on a triangle t to the source s can be obtained by using a backtracing technique similar to the one described in [16]. We first determine the element $\omega_m \in \Omega$ defining the minimum distance to q. There exist two possibilities: 1) If ω_m is a window w_m, we jump to the adjacent triangle t' by using the direction τ when w_m is a p-window or the angle ϕ when w_m is an s-window; 2) If ω_m is a pseudosource, it is an endpoint of a window w_m which is used to jump to the adjacent triangle. Then we keep on jumping to the adjacent triangle until we get s.

The shortest path can be obtained in $O(\log n + n')$ time, when the path crosses n' triangles.

2.5. Implicit generalized Voronoi diagram

The algorithm for computing the distance function from a generalized source extends naturally to the case of several sources. In this case we obtain a generalized *Voronoi function*, which for any point of $\mathcal{P}$ gives the shortest path distance to its nearest source. In the initialization step we generate windows for each single source and store them in a unique priority-queue. Thus, we propagate the distance functions defined by all single sources simultaneously. In this way we obtain a codification of the Voronoi function that yields an implicit representation of the Voronoi diagram of the set of generalized sources.

From now on we denote by N the maximum of n and the total number of segments conforming the generalized sources. When the Voronoi function of a set of generalized sources is computed, the maximum number of created windows is $O(N^2)$. Consequently, the implicit Voronoi function can be obtained in $O(N^2 \log N)$ time and $O(N^2)$ space.

3. Distance vectors

The distance vector $\overrightarrow{d_q}$ of a point q with respect to a virtual source w_s (a point or a segment) is the vector joining the closest point to q on w_s with q. Observe that if q belongs to the influence region P_w of a window w, we have $D_{s,w}(q) = d(s, w_s) + \|\overrightarrow{d_q}\|$, where $d(s, w_s) \neq 0$ when w_s is a pseudosource. Distance vector properties, previously used in [14] for purposes similar to ours, allow us to compute $\overrightarrow{d_q}$ interpolating the distance vectors of the vertices of P_w.

Punctual source. Assume that the convex polygon P_w has been triangulated and the point q of P_w belongs to the triangle of 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 relative to a punctual virtual source w_s. The 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 $\alpha_1, \alpha_2, \alpha_3 \geq 0$. Consequently $\overrightarrow{d_q} = w_s - q = \alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2} + \alpha_3 \overrightarrow{d_3}$ (See Figure 4a).

Segment source. Assume again that the point q of P_w belongs to the triangle of vertices p_1, p_2, p_3 and now 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 w_s. As before the 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 $\alpha_1, \alpha_2, \alpha_3 \geq 0$. We want to prove that we have $\overrightarrow{d_q} = \alpha_1 \overrightarrow{d_1} + \alpha_2 \overrightarrow{d_2} + \alpha_3 \overrightarrow{d_3}$ (See Figure 4b).

Let first consider the case when there exist one α_i 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 w_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 w_s defining $\overrightarrow{d_1}$ and $\overrightarrow{d_2}$ respectively. When $\alpha_i \neq 0$, $i = 1, 2, 3$, we can consider a segment 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 $\overrightarrow{d_3}$.

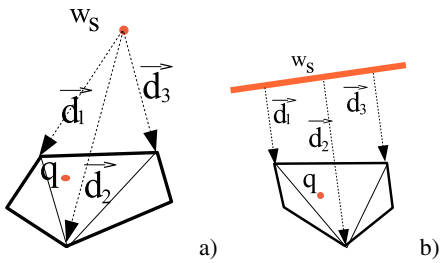

Figure 4. The distance vectors associated to the vertices of a convex polygon when dealing with: a) a point source; b) a segment source. In both figures the distance vector to point q can be obtained by the shown distance vectors.

4. Discrete distance function computation

In this section we present our algorithm to efficiently compute discrete distance functions by using graphics hardware.

4.1. Planar parameterization

To obtain a discrete representation of the distance function we use a 2D representation of the triangulated surface $\mathcal{P}$. We map each triangle t of $\mathcal{P}$ to a triangle in the $xy-$plane with the same shape (angles and area). This is done in such a way that the mapping of two different triangles of $\mathcal{P}$ do not overlap in the xy-plane. The previous mapping is known as a planar parametrization of the triangulated surface. Computing a parameterization means finding the 2D

0-7695-2869-4/07 $25.00 © 2007 IEEE

coordinates corresponding to each triangle of $\mathcal{P}$. This is a well studied problem in the literature with several strategies and different applications such as: texture mapping, geometry processing, remeshing, etc [3, 6]. The problem of obtaining a parameterization maintaining the shape of the triangles is also known as *triangle packing* problem. We use the algorithm for triangle packing given in [3]. It translates and rotates the mesh triangles so that they are placed in the $xy-$plane with the longest edge aligned to the $x-$axis. After sorting the triangles by increasing altitude, each other triangle is flipped, triangles are grouped into equal length sections and finally triangle groups are stacked vertically. At the end of the process, the triangles are packed into an axis-parallel rectangular region $\mathcal{R}$. The time cost of the algorithm is $O(n \log n)$.

For the special case of a polyhedral terrain, a polyhedral surface such that its intersection with any vertical line is either empty or a point, we can alternatively use its projection on the xy-plane as a 2D parametrization. Now $\mathcal{R}$ denotes the axis-parallel rectangular region bounding the projection. In this case the shape of the projected triangles do not correspond with its shape on the polyhedral terrain.

4.2. Process overview

After precomputing a parameterization of $\mathcal{P}$, we discretize the region $\mathcal{R}$ of the xy-plane as a rectangular grid of size $W \times H$ that induces a discretization on the triangles of $\mathcal{P}$. When the parameterization maintains the shapes of the triangles, the discretization error in the xy-plane matches with the discretization error on $\mathcal{P}$.

The parameterization and the surface discretization are used to obtain a discrete representation on $\mathcal{R}$ of the distance function defined by shortest paths on $\mathcal{P}$. This is achieved by keeping track of the explicit representation of the distance function while it is propagated along the surface $\mathcal{P}$ with the algorithm explained in Section 2. In the window propagation stage we compute the distance, defined by the current window w, to all the grid points contained in the influence region P_w. If a pseudosource v is created we also compute the distance to the points of the influence region P_v. The distance is computed by using distance vectors and graphics hardware (see Subsection 4.3). Since grid points can be contained in the influence region of different windows, during the process we store the minimum of the distances obtained for its corresponding point on $\mathcal{P}$ in each grid point of the xy-plane.

The OpenGL pipeline triangulates input polygons, processes the triangle vertices and rasterizes the triangles into fragments by interpolation. 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.

The vertex shader uses the planar parameterization to map the 3D points on the surface $\mathcal{P}$ to 2D points in the region $\mathcal{R}$ of the xy-plane. Then the fragment shader computes the distance defined by the current window or pseudosource at each point and 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 depth buffer position.

4.3. Distance function computation

To compute the explicit distance we discretize the region $\mathcal{R}$ into a grid of $W \times H$ pixels. In the initialization process of the algorithm explained in Section 2, we initialize the depth buffer to the maximal depth value (1). When a new fragment is processed, the depth buffer is updated if the depth value of the current fragment is lower than the actual value in the depth buffer. At the end of the process the value stored in the depth buffer is the minimum depth (distance) of all the processed fragments. Next we explain in more detail the computation process.

During the continuous Dijkstra propagation process, for each propagated window w with virtual source w_s, we compute its distance function in the influence region P_w (Subsection 2.4) by painting the polygon P_w. The distance given by w in a point $q \in P_w$ is obtained by adding the distance from q to w_s to the distance from s to w_s. The distances to these points are computed and stored in the depth buffer whenever the distance defined by w is smaller than the distances previously stored in the buffer. The propagation of w defines P_w, and sometimes a new pseudosource v is created (See Figure 5). The influence region P_v of the pseudosource v is handled in the same way that window influence regions.

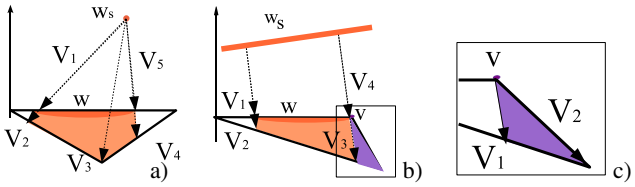

Figure 5. Examples of influence regions and distance vectors of: a) a p-window; b) a s-window; c) the vertex pseudosource obtained in b).

The distance vectors from the virtual source of w_s to the vertices of P_w are computed. Posteriorly the polygon P_w is painted by using OpenGL. We associate to the polygon, in a texture coordinate channel, the distance from the virtual source to the initial source. We also associate to each vertex

0-7695-2869-4/07 $25.00 © 2007 IEEE

of P_w its distance vector using another texture coordinate channel. The influence region P_v of a pseudosource v is handled as the influence region of a window.

At the end of the process, we have in the depth buffer the distance function defined by the source.

4.4. Algorithm correctness

We have to prove that at each point q of the polyhedral surface $\mathcal{P}$ the actual minimum distance is obtained. On the one hand, the fact that each window w only affects the points contained in its influence region and the observations related to distances and shortest paths to points of a triangle t (Subsection 2.4) prove that for a given point q in t we have considered all the possible geodesic paths arriving at q. On the other hand, the observations given in Section 3 prove that the path length is correctly computed. In fact, distance vectors of interior points of an influence region are properly obtained during the rasterization process from the distance vectors of the region vertices and, consequently, the algorithm properly computes the distance function value defined by w in the points of its influence region. Finally, since only the minimum distance value is stored in the depth buffer, the shortest path distance function is correctly computed.

5. Discrete closest Voronoi diagrams

In this Section we propose a way to compute the discrete closest Voronoi diagram for a set of generalized sources S on the polyhedral surface $\mathcal{P}$ with obstacles. Although in Section 6 a general procedure to compute k-order Voronoi diagrams is given, for the special case of the closest Voronoi diagram ($k = 1$) this process is much more efficient, in both, time and space complexity.

To obtain the closest Voronoi diagram, we discretize the generalized Voronoi function and properly determine the Voronoi regions. The discretization of the generalized Voronoi function is obtained by slightly modifying the algorithm for computing discrete distance functions described in Section 4. We use a planar parametrization of the triangulated surface $\mathcal{P}$ 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 window 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.

Notice that there can exist regions equidistant to two different sites. These regions can be one or two dimensional regions. The 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 painted in one or the other color depending on the behavior of the continuous Dijkstra propagation strategy, and the order in which the regions are painted.

The extra time needed to obtain the closest Voronoi diagram while using the algorithm for implicitly obtain the Voronoi function is the time spent by painting the windows influence regions, which is done in constant time. Therefore, the time and space complexity are $O(N^2 \log N)$ and $O(N^2)$, respectively.

6. Discrete high order Voronoi diagrams

In this section we describe how to obtain the discrete k-order Voronoi diagram, $k = 1, \cdots, m-1$, for a set of m generalized sources S on the polyhedral surface $\mathcal{P}$ with obstacles. 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. They can be stored by rendering the depth buffer in a depth texture or stored in the CPU.

Closest Voronoi diagram. The closest Voronoi diagram can be obtained by painting, one after the other, the m distance functions and determining their lower envelope. To paint 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 painted 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.

To obtain the closest Voronoi diagram the m discrete distance function are computed and stored, then, they are transferred to a texture and painted. Therefore, the time and space complexity of the algorithm are $O(m(n^2 \log n + HW))$ and $O(n^2 + mHW)$, respectively.

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 and storing the distance in the depth of the fragment. The depth test is used to store in each pixel the maximum depth value. Each point is accordingly painted with the color of the fragment with maximal depth value, which is the color associated to the furthest site to each point. Now 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.

4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD 2007)
0-7695-2869-4/07 $25.00 © 2007 IEEE

k-order Voronoi diagram. Our algorithm for computing discrete k-order Voronoi diagrams uses a depth peeling technique similar to the one described on [5] for computing 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 painted 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. When it finishes, the depth buffer is transferred to a texture and send to the fragment shader. In the second pass all the distance functions are again painted. In the fragment shader the distance function that is being painted is compared 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 painted, 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 the m discrete distance function are computed and stored. For each of the k peeling passes, the depth buffer is copied to a texture and the m distance functions are transferred to a texture and painted. Therefore, the algorithm has $O(mn^2 \log n + kmHW)$ time and $O(n^2 + mHW)$ space complexities.

7. 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 3D polyhedral surface.

8. Error analysis

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

9. Results and future work

Results. We have implemented the proposed method 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 3GHz with 1GB of RAM and a GeForce 7800 GTX/PCI-e/SSE2 graphics board.

Figures 6 to 9 show some examples of Voronoi diagrams for generalized sources on polyhedral terrains with obstacles obtained using our implementation, which is being improved. We have considered a set S of ten sites: four points, two segments, two polygonal chains and two polygon sources, and a terrain with 800 triangular faces. The generalized sources, except for the polygon sources interior, are painted on the terrain surface and the remaining points of the surface are colored according to the Voronoi region they belong to. In Figure 6 we show the closest Voronoi diagram of S, it is obtained in 0.748(s). In Figure 7 each point is painted in the color of the 7th nearest site, we do not show the 7th order Voronoi because the image is difficult to understand due to the merged colors. The time needed to obtain and store the 10 distance fields is 7.811(s), and the extra time needed to visualize the 7th nearest site is 0.811(s). The error produced by the 32-bit precision of the depth buffer can be seen in this image, isolated pixels are painted in the color of the regions adjacent to the region they belong to. In Figure 8 the furthest Voronoi diagram is obtained in extra 0.129(s). Finally, we show in Figure 9 the closest Voronoi diagram of S when obstacles (two polygonal chains and a polygonal region), which are painted black, are considered. It was obtained in 0.756 (s).

Future work. We are implementing the algorithms for computing the distance and Voronoi functions and Generalized Voronoi diagrams for general polyhedral surfaces. We expect that in practice the algorithm for distances computation will run in sub-quadratic time as in [16]. We are studying alternative heuristics to the algorithm for triangle packing to obtain a more efficient packing.

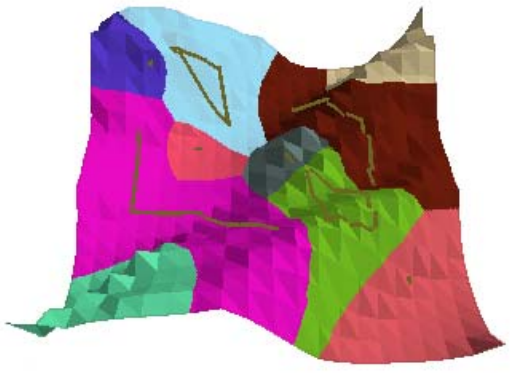

Figure 6. Closest Voronoi diagram of a set of ten generalized sites. Each Voronoi region is painted in a different color which is associated to the site defining the region.

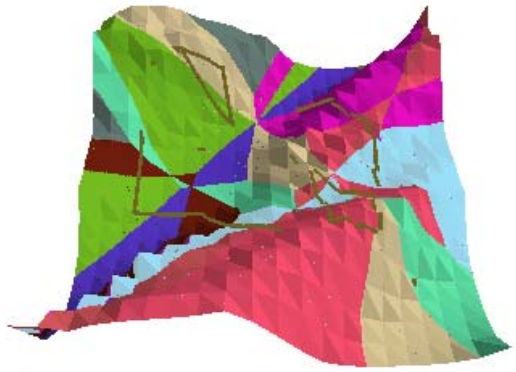

Figure 7. 7th-nearest diagram, each point is painted according to its 7th-nearest generalized site. See the associated colors in Figure 6.

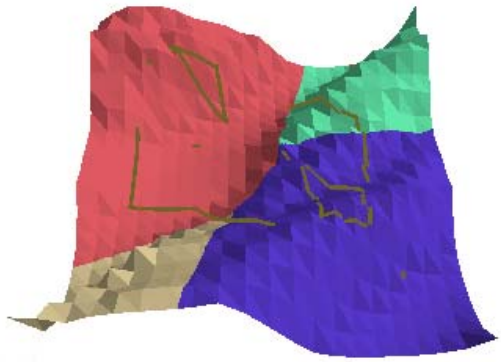

Figure 8. Furthest Voronoi diagram of a set of generalized sources. Each point is painted in the color of its furthest site. See the associated colors in Figure 6.

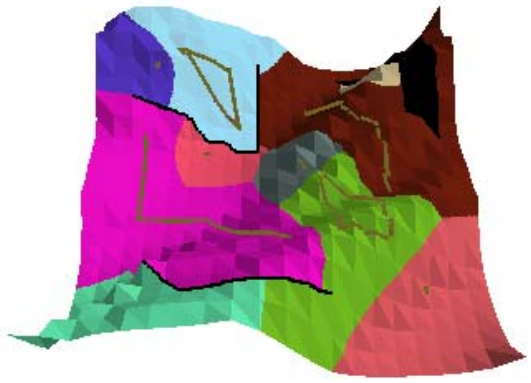

Figure 9. Closest site Voronoi diagram of a set of generalized sources for a polyhedral terrain with obstacles. Obstacles are painted black. Each Voronoi region is painted in a different color which is associated to its site.

References

[1] F. Aurenhammer. Voronoi diagrams: a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[2] F. Aurenhammer and R.Klein. *Handbook of Computational Geometry*, chapter Voronoi diagrams, pages 201–290. Elsevier, 2000.

[3] N. Carr, J. Hart, and J. Maillot. The solid map: Methods for generating a 2-d texture map for solid texturing. In *Proc. Western Computer Graphics Symposium*, pages 179–190, 2000.

[4] J. Chen and Y. Han. Shortest paths on a polyhedron. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pages 360–369, New York, NY, USA, 1990. ACM Press.

[5] I. Fisher and C. Gotsman. Fast approximation of high order voronoi diagrams and distance transforms on the GPU. *Journal of Graphics Tools*, 11(4):39–60, 2006.

[6] M. S. Floater and K. Hormann. Surface parameterization: a tutorial and survey. In N. A. Dodgson, M. S. Floater, and M. A. Sabin, editors, *Advances in multiresolution for geometric modelling*, pages 157–186. Springer Verlag, 2005.

[7] K. E. Hoff III, J. Keyser, M. Lin, D. Manocha, and T. Culver. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics*, 33(Annual Conference Series):277–286, 1999.

[8] B. Kaneva and J. O'Rourke. An implementation of Chen and Han's shortest paths algorithm. In *Proceedings of the 12th Canadian Conference on Computational Geometry*, pages 139–146, 2000.

[9] S. Kapoor. Efficient computation of geodesic shortest paths. In *STOC '99: Proceedings of the thirty-first annual ACM symposium on Theory of computing*, pages 770–779, 1999.

[10] J. Mitchell. *Handbook of Computational Geometry*, chapter Geometric shortest paths and network optimization, pages 633–701. Elsevier Science Publishers B. V., 2000.

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

[12] A. Okabe, B. Boots, K. Sugihara, and S. N. Chiu. *Spatial Tessellation: Concepts and Application of Voronoi Diagrams*. John Wiley and Sons, 2000.

[13] C. Sigg, R. Peikert, and M. Gross. Signed distance transform using graphics hardware. In *VIS '03: Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*, pages 83–90, 2003.

[14] A. Sud, N. Govindaraju, R. Gayle, and D. Manocha. Interactive 3d distance field computation using linear factorization. In *SI3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, pages 117–124, 2006.

[15] A. Sud, M. Otaduy, and D. Manocha. DiFi: Fast 3d distance field computation using graphics hardware. In *Eurographics*, volume 23, pages 557–566, 2004.

[16] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, and H. Hoppe. Fast exact and approximate geodesics on meshes. *ACM Trans. Graph.*, 24(3):553–560, 2005.

0-7695-2869-4/07 $25.00 © 2007 IEEE