

# Generalized Voronoi Diagrams on Polyhedral Terrains

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 polyhedral terrain 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 terrain 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 polyhedral terrain which 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 terrain.

### 1.1 Preliminaries

Let  $\mathcal{T}$  be a polyhedral terrain, a triangulated surface which is intersected at most once by any vertical line, represented as a mesh consisting of  $n$  triangular faces. From now on, a generalized element on  $\mathcal{T}$  refers to a point, segment, polygonal chain or a polygon. We model obstacles in  $\mathcal{T}$  by a set of non-punctual generalized elements on  $\mathcal{T}$ .

A shortest path between a generalized source and a point on  $\mathcal{T}$  is a shortest path in the Euclidean metric between the source and the point such that stays on  $\mathcal{T}$  and avoids the obstacles. The shortest path distance function defined by a source point  $p$  on  $\mathcal{T}$  is a function  $D_p$  such that for any point  $q \in \mathcal{T}$ ,  $D_p(q)$  is the Euclidean length of the shortest path along  $\mathcal{T}$  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  $q \in \mathcal{T}$ ,  $D_s(q)$  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 terrain  $\mathcal{T}$ . The set of points of  $\mathcal{T}$  closer to a site  $s$  of  $S$  than to any other site of  $S$ , where distances are shortest path distances on  $\mathcal{T}$ , is called the Voronoi region of  $s$ .

### 1.2 Previous work

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

Computing shortest paths on triangulated non-convex polyhedral surfaces is a fundamental problem in computational geometry with important applications in computer graphics, robotics and geographical information systems [6, 8]. The single point source *shortest path* problem consists on finding a

---

\*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.

shortest path in the Euclidean metric from a source point to any target point such that the path stays on the triangulated non-convex surface  $\mathcal{P}$ . Mitchell et al. [5] 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. In [8] a simple way to implement the algorithm is described and shown to run much faster than the  $O(n^2 \log n)$  theoretical worst case time. In [2], Chen and Han, 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 triangulated polyhedron. The algorithm also answers single-source shortest path queries in  $O(\log n + n')$  time. In [3], Kaneva and O'Rourke 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. In [4], Kapoor presented an algorithm following "continuous Dijkstra" paradigm that computes a shortest path from a source point to a target point in  $O(n \log^2 n)$  time. The difficulties of such an algorithm make complicated its implementation.

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]. 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. Different distance function based algorithms have been proposed to compute 2D and 3D discretized Voronoi diagrams along a grid using graphics hardware, most of them are summarized in [7]. 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.

## 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, and consequently triangulated terrains, have the following characterization [5]: 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  $2\pi$  (*saddle* vertex). The basic idea of the algorithm of Mitchell et al. [5] for solving the shortest path problem from a mesh vertex  $v$ , as implemented by Surazhsky et al. [8], 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.

In this paper we extend the ideas developed in [8] for the efficient implementation of the algorithm of Mitchell et al. to the generalized source case. 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  $\mathcal{T}$ , the part of a shortest path interior to  $t$  starting at an interior point of  $s'$  is orthogonal to  $s'$ .

## 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 segment 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  $\mathcal{T}$  into windows. Geodesic paths that cross a window go through the same triangles and bend at the same vertices of  $\mathcal{T}$ . 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  $e$ . 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*.

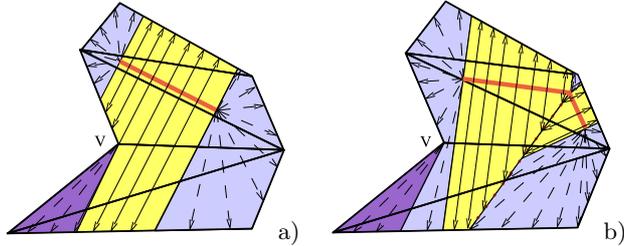


Figure 1: An unfolded strip of triangles with: a) a segment source; b) a polygonal chain source.

***p-windows***. Following the strategy of Surazhsky et al. [8], 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  $\tau$  specifies the side of  $e$  on which  $p$  lies, and  $\sigma$  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  $t$ , and to recover the distance function within  $w$ , by simulation 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  $w$ .

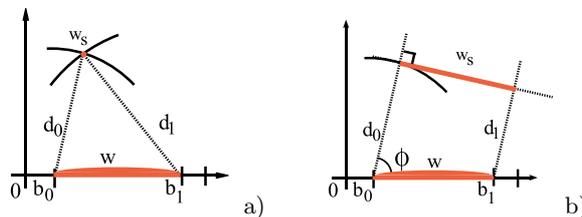


Figure 2: The virtual source  $s$  is positioned using the information stored in: a) a  $p$ -window; b) a  $s$ -window.

### 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 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  $\sigma'$  does not change, and for  $s$ -windows the angle  $\phi'$  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  $\sigma'$  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).

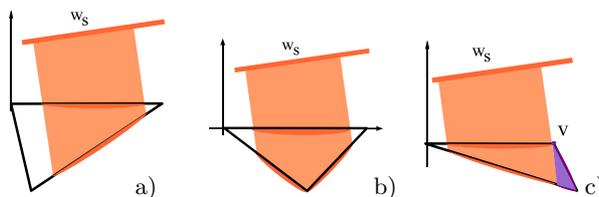


Figure 3: Examples of  $s$ -window propagation resulting in: a) a new single window; b) two new windows; c) a new single window and a pseudosource vertex  $v$ .

### 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  $\delta$  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 [8]. 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. 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.

## 2.2 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  $\mathcal{T}$  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  $r$ .

## 2.3 Polygonal obstacles

Given a possible non-convex polyhedral terrain  $\mathcal{T}$  (which may represent a terrain), we model obstacles as polygonal chains or polygonal regions (which may represent rivers, lakes, etc) on  $\mathcal{T}$ . The polyhedral terrain is retriangulated so that the obstacles are represented as several mesh triangles, edges and vertices. Abusing language, we still denoted  $n$  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, obstacle vertices are new pseudosource vertices regardless of their total angle.

## 2.4 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  $\mathcal{T}$  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 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.

## 2.5 Complexity analysis

Following a similar discussion as the one given in [5], it can be proven that the voronoi function for a set of generalized sources on a non-convex polyhedral terrain  $\mathcal{T}$  with polygonal obstacles represented

as a mesh consisting of  $n$  triangles can be intrinsically obtained in  $O(N^2 \log N)$  time and  $O(N^2)$  space, where  $N$ , from now on, denotes the maximum of  $n$  and the total number of segments conforming the generalized sources.

### 3 Discrete distance function computation

In this section we present our algorithm to efficiently compute discrete distance functions by using graphics hardware. The already obtained implicit distance function allows us to easily obtain the distance at the triangle edges. However, we are interested in obtaining distance at all the terrain points. To facilitate the distance computation we determine which points of  $\mathcal{T}$  can be reached by the geodesics encoded in a window, and also present a way to compute distances as vector lengths.

#### Influence region

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 vectors

The distance vector  $\vec{d}_q$  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  $D_{s,w}(q) = d(s, w_s) + \|\vec{d}_q\|$ , where  $d(s, w_s) \neq 0$  when  $w_s$  is a pseudosource. Distance vector properties, previously used in [7] for purposes similar to ours, allow us to compute  $\vec{d}_q$  interpolating the distance vectors of the vertices of  $P_w$ . Assume that point  $q$  of  $P_w$  belongs to the triangle of vertices  $p_1, p_2, p_3$  and now denote  $\vec{d}_1, \vec{d}_2, \vec{d}_3$  the distance vectors of  $p_1, p_2, p_3$  relative to a virtual source  $w_s$  (a segment or punctual virtual source). Point  $q$  can be univocally expressed as  $\vec{q} = \alpha_1 \vec{p}_1 + \alpha_2 \vec{p}_2 + \alpha_3 \vec{p}_3$ , with  $\alpha_1 + \alpha_2 + \alpha_3 = 1$  and  $\alpha_1, \alpha_2, \alpha_3 \geq 0$ . It can be proved that we have  $\vec{d}_q = \alpha_1 \vec{d}_1 + \alpha_2 \vec{d}_2 + \alpha_3 \vec{d}_3$  (See Figure 4).

#### 3.1 Process overview

To obtain the distance function, we project the terrain on the  $xy$ -plane (obtaining a bijection) and denote  $\mathcal{R}$  to the axis-parallel rectangular region bounding the projection. Region  $\mathcal{R}$  of the  $xy$ -plane is discretized as a rectangular grid of size  $W \times H$  that induces a discretization on the triangles of  $\mathcal{T}$ . We obtain a discrete representation on  $\mathcal{R}$  of the distance function defined by shortest paths on  $\mathcal{T}$ . This is achieved by keeping track of the explicit representation of the distance function while it is propagated along the terrain  $\mathcal{T}$  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 3.2). Since grid points can be contained in the influence region of different windows, during the process we store in each grid point of the  $xy$ -plane the minimum of the distances obtained for its corresponding point on  $\mathcal{T}$ .

The OpenGL pipeline triangulates input polygons, processes the triangle vertices and rasterizes the triangles into fragments by bilinear interpolation. Within the rasterization step all the parameters

associated to vertices such as texture coordinates, color, normal vectors, etc. are also bilinear interpolated from those associated to the triangle vertices. Consequently the value obtained in these channels in a fragment is the bilinear interpolation of the values associated to the triangulated polygon vertices. We use the fragment shader to compute the distance defined by the current window or pseudosource at each point and set 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.

### 3.2 Distance function computation

To compute the explicit distance we discretize into a grid of  $W \times H$  pixels the region  $\mathcal{R}$ . 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 current 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, for each propagated window  $w$  with virtual source  $w_s$ , we compute its distance function in the influence region  $P_w$  by painting the polygon  $P_w$ . 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  $w_s$ . 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 4). The influence region  $P_v$  of the pseudosource  $v$  is handled in the same way that window influence regions. 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 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 exact distance function defined by the source.

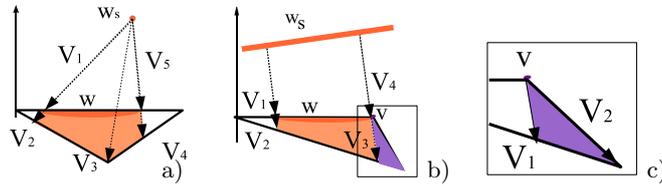


Figure 4: Examples of influence regions and distance vectors of: a) a  $p$ -window; b) a  $s$ -window; c) the vertex pseudosource obtained in b).

## 4 Discrete Voronoi diagrams

In this Section we propose a way to compute the discrete Voronoi diagram for a set of generalized sources  $S$  on the triangulated terrain  $\mathcal{T}$  with obstacles. The generalized Voronoi function is discretized and we properly determine the Voronoi regions. The discretization of the generalized Voronoi function is obtained slightly modifying the algorithm for computing discrete distance functions described in Section 3. We use a projection of the terrain  $\mathcal{T}$  into the  $xy$ -plane 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 Voronoi diagram. 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 terrain. The time and space complexity is, again,  $O(N^2 \log N)$

and  $O(N^2)$ , respectively because we have added a constant time operation in the window propagation stage.

## 5 Results and future work

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. Figure 5 show examples of Voronoi diagrams for generalized sources on polyhedral terrains without and 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 5a) we show the Voronoi diagram of  $S$ , it is obtained in 0.748(s). In Figure 5b) the 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).

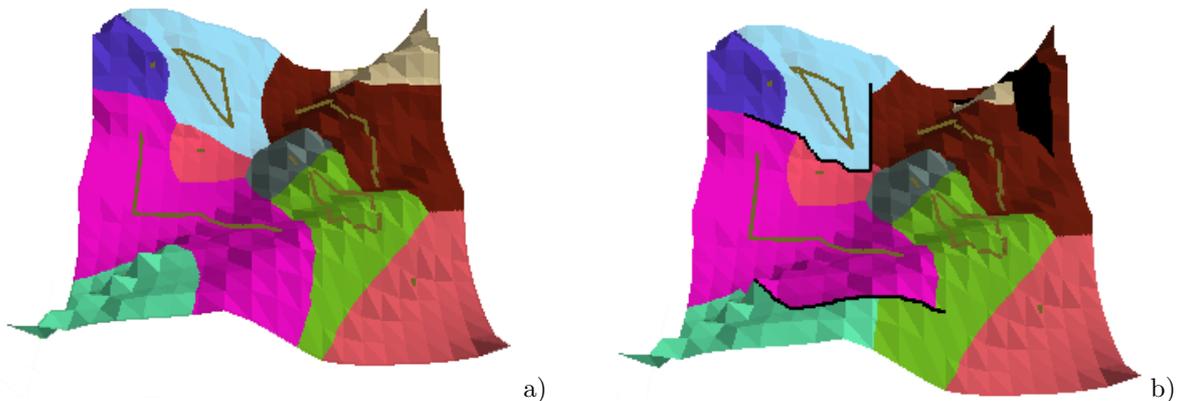


Figure 5: Voronoi diagram of 10 generalized sites on a terrain with a) no obstacles b) 3 obstacles.

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 [8]. We are also working in order to obtain higher order Voronoi Diagrams both for polyhedral terrains and general polyhedral surfaces.

## References

- [1] F. Aurenhammer, Voronoi diagrams: A survey of fundamental geometric data structure. *ACM Computer Surveys*, 1991, 23(3), 686–695.
- [2] J. Chen, Y. Han, Shortest paths on a polyhedron. *Int.J. Comput. Geom. Appl.*, 1996, 6, 127–144.
- [3] B. Kaneva, J. O’Rourke, An Implementation of Chen Han’s Shortest Paths Algorithm. *Proc. of the 12th Canadian Conf. on Comput. Geom.*, 2000, 139–146.
- [4] S. Kapoor, Efficient Computation of Geodesic Shortest Paths. *Proc. 32nd Annu. ACM Sympos. Theory Comput.*, 1999, 770–779.
- [5] J. Mitchell, D. Mount, H. Paparimitriou, The discrete geodesic problem. *SIAM J. Computation*, 16(4), 1987, 647–668.
- [6] J. S. B. Mitchell, Geometric shortest paths and network optimization. In *Jörg-Rüdiger Sack and Jorge Urrutia, editors, Handbook of computational geometry, Elsevier*, 2000, 633–701.
- [7] A. Sud, N. Govindaraju, R. Gayle, D. Manocha, Interactive 3D Distance Field Computation using Linear Interpolation. *SI3D ’06: Proceedings of the 2006 symposium on Interactive 3D graphics and games*, 2006, 117–124.
- [8] V. Surazhsky, T. Surazhsky, D. Kirsanov, S. J. Gortler, H. Hoppe, Fast Exact and Approximate Geodesics on Meshes. In *ACM Transactions on Graphics (TOG), Proc. of ACM SIGGRAPH*, 2005, 24(3), 553–560.