



# Level Set Diagrams of Polyhedral Objects

Francis Lazarus, Anne Verroust

► **To cite this version:**

Francis Lazarus, Anne Verroust. Level Set Diagrams of Polyhedral Objects. [Research Report] RR-3546, INRIA. 1998. <inria-00073138>

**HAL Id: inria-00073138**

**<https://hal.inria.fr/inria-00073138>**

Submitted on 24 May 2006

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

***Level Set Diagrams of Polyhedral Objects***

Francis Lazarus &amp; Anne Verroust

**No 3546**

\_\_\_\_\_ THÈME 3 \_\_\_\_\_

 ***Rapport  
de recherche***  
\_\_\_\_\_



## Level Set Diagrams of Polyhedral Objects

Francis Lazarus\* & Anne Verroust†

Thème 3 — Interaction homme-machine,  
images, données, connaissances  
Projet SYNTIM

Rapport de recherche n° 3546 — — 22 pages

**Abstract:** Shape descriptors and feature-based representations are of primary interests in the area of solid modeling. They allow us for easier storage, recognition and general treatments of objects. Axial structures such as skeletons are popular shape descriptors which have been widely studied. Most of the studies focus on a particular type of skeleton called the Medial Axis. Medial Axes can be extracted from discrete volumetric data as well as boundary-based representations. In the later case, however, no algorithm is known to perform well and accurately. We propose a new paradigm for constructing one dimensional axial structures associated with a polyhedral object. These structures, called the level set diagrams, are associated with scalar functions defined over the set of vertices of a polyhedron. We study in details the level set diagram associated with the shortest path distance to a source point. This particular association fits nicely into a theoretical framework and presents interesting properties for the purpose of shape description.

**Key-words:** skeleton, cylindrical decomposition, Euler Formula, Dijkstra's algorithm

*(Résumé : tsvp)*

\* I.R.C.O.M.-S.I.C. (U.R.A. C.N.R.S. 356) SP2MI Boulevard 3, Teleport 2, B.P. 179 86960 Futuroscope Cedex, France. [lazarus@sic.univ-poitiers.fr](mailto:lazarus@sic.univ-poitiers.fr)

† [Anne.Verroust@inria.fr](mailto:Anne.Verroust@inria.fr)

## Diagrammes des ensembles de niveaux d'objets polyédriques

**Résumé :** L'étude des descripteurs de forme et des caractéristiques géométriques des objets tridimensionnels est une composante du domaine de la modélisation géométrique. Ces descripteurs permettent un stockage aisé et facilitent le traitement et la reconnaissance des objets. Les structures axiales telles que les squelettes sont des descripteurs de forme couramment étudiés. La plupart des études se concentrent sur un type particulier de squelette appelé axe médian. Les axes médians peuvent être extraits à partir des données volumétriques discrètes aussi bien que des représentations par frontière des objets. Dans ce dernier cas, cependant, il n'existe pas actuellement d'algorithme exact et efficace. Nous proposons une nouvelle notion de structures axiales unidimensionnelles associées à un objet polyédrique. Ces structures, appelées diagrammes des ensembles de niveaux, sont associées aux fonctions scalaires définies sur l'ensemble des sommets d'un polyèdre. Nous étudions en détails les diagrammes des ensembles de niveaux associés à la distance de plus court chemin à un point source. Nous décrivons les propriétés propres à ces diagrammes et en déduisons un algorithme efficace de construction. Nous présentons finalement des résultats de construction sur des objets tridimensionnels de formes diverses.

**Mots-clé :** squelette, décomposition cylindrique, formule d'Euler, algorithme de Dijkstra

## 1 Introduction

We present a new paradigm for constructing one dimensional axial structures, the level set diagrams (LSDs for short), associated with a polyhedral object. These diagrams capture the overall shape as well as the topology of an object. They can be used for deforming or animating an object.

This construction is radically different from the classical Medial Axis [LL89] which does not necessarily produces a one dimensional structure. Formally, the Medial Axis is the set of the centers of maximal spheres contained in a given object. Algorithms that construct the Medial Axis of a polyhedral surface are generally complex. They start with adding points on the object boundary and computing their Delaunay triangulation [GYKD91]. Generally, the resulting Medial Axis needs to be pruned as it may contains numerous small branches [AM97]. Also, the Medial Axis can be composed of pieces of curves as well as pieces of surfaces. By construction, however, LSDs are simple one dimensional graphs. They relate to the level sets of scalar functions defined over the vertices of a polyhedral surface. More precisely, the points of an LSD correspond to “average points” associated with the connected components of the level sets of a given scalar function. Figure 1 shows an LSD of a three-fingered shape.

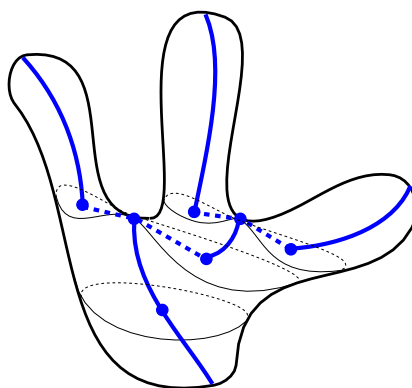


Figure 1: An LSD (bold lines) of an object is defined by the centers of the connected components of the level sets (thin lines) of a given function. The dashed bold lines indicate joints between LSD branches.

Shinagawa et al. [SKK91, TIS<sup>+</sup>95] have introduced a similar diagram in the context of medical and geographical applications. They call their diagram the *Reeb graph* in reference to a classical topological graph associated with level sets of a Morse function<sup>1</sup>,  $f$ , defined

<sup>1</sup>A Morse function, on a surface, is a twice differentiable function whose Hessian - the Jacobian of the gradient in a suitable coordinate system - has a non-zero determinant everywhere its gradient vanishes.

on a smooth surface [Mil63, Har97]. The vertices of the Reeb graph correspond to critical points of the function  $f$ , that is points where the gradient of  $f$  is null.

Shinagawa et al. are mainly interested in encoding the topological structure of a terrain while the LSD is a geometrical skeleton that describes both the topology and the geometry of an object.

LSDs depend on the particular function chosen to compute the level sets. Shinagawa et al. use the height function (the z-coordinate), relatively to a given orientation of the object to define their diagram. This is particularly meaningful for terrain data as critical points correspond to peaks, pits or passes of the terrain. We show that in our case the *shortest path distance to a source point* is often a better choice of function especially for describing the geometry of tubular and branching objects.

Section 2 presents some basic definitions and properties of the level sets of a function defined on a triangulated polyhedron. In Section 3, we describe the evolution of a the level-sets as the level is varied. This description is based upon the notion of index of a vertex. Section 4 studies the level sets of the shortest path distance to a source point. This study is used in Section 5 to define efficient algorithms that construct the LSD. Results are commented in Section 6. Finally, we discuss future works and conclude in Section 7.

## 2 Level Sets of Functions

This section provides some definitions and notation that are important to the construction and properties of the LSD. In the following, we only consider LSDs of triangulated polyhedra.

For a given triangulated polyhedron,  $\mathcal{P}$ , we consider a function,  $f$ , defined over its vertices  $v_1, \dots, v_n$  so that two adjacent vertices are given two different values. Actually, as noticed in [TIS<sup>+</sup>95], all that is required is to tell which of any two adjacent vertices gets the smaller value. In practice, we can order arbitrarily the vertices having the same value so that the above requirement can be alleviated.

**Definition 1** *An edge  $(v_i, v_j)$  is said to be a cross-edge relatively to a level  $l$  if*

$$f(v_i) < l < f(v_j).$$

*Similarly, a face  $(v_i, v_j, v_k)$  is said to be a cross-face relatively to a level  $l$  if the three values of its vertices encompass  $l$ , that is*

$$\begin{aligned} f(v_i) < l < f(v_j) < f(v_k), \text{ or} \\ f(v_i) < f(v_j) < l < f(v_k). \end{aligned}$$

Remark that a cross-face has exactly two cross-edges.

**Definition 2** *The dual graph of  $\mathcal{P}$  is the graph whose vertices are the faces of  $\mathcal{P}$  and whose edges correspond to pairs of incident faces.*

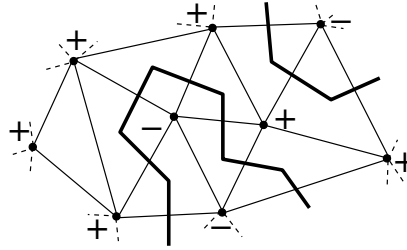


Figure 2: A triangulation with its cross-adjacency graph (bold lines) for a given level  $l$ . Vertices with values greater (smaller) than  $l$  are marked with a “+” (a “-”).

The preceding remark shows that a cross-face (relatively to a level  $l$ ) is adjacent to exactly two cross-faces in the dual graph (see Figure 2). It follows that

**Property 1** *For any level  $l$ , distinct from the values  $f(v_1), \dots, f(v_n)$ , the set of cross-faces constitutes a set of disjoint cycles in the dual graph.*

This property is to be compared with the fact that a level set of a Morse function defined on a closed smooth surface is composed of disjoint closed curves, if that level set contains no critical point.

These cycles of cross-faces define closed polygonal contours on the polyhedron  $\mathcal{P}$ . For this, we associate with each cross-edge  $(v_i, v_j)$  the point

$$p_{ij}(l) = \frac{l - f(v_i)}{f(v_j) - f(v_i)}v_i + \frac{f(v_j) - l}{f(v_j) - f(v_i)}v_j.$$

If  $(v_i, v_j, v_k)$  is a cross-face, we can draw the line  $p_{ij}(l)p_{ik}(l)$  as on Figure 3.

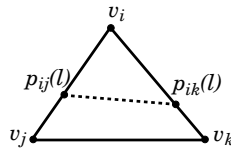


Figure 3: Segment  $(p_{ij}(l)p_{ik}(l))$  is the geometric realization of the cross-face  $(v_i, v_j, v_k)$ .

**Definition 3** *The geometric realization of a level set is the set of polygonal contours obtained by drawing the lines  $(p_{ij}(l)p_{ik}(l))$  in every cross-faces. We will note  $\mathcal{C}(l)$  the realization of the level set of level  $l$ .*



These polygonal contours are not necessarily planar. They will be further “averaged” in order to construct the LSD.

If  $l_1 < l_2$  are two levels, such that no vertex  $v$  verifies  $l_1 \leq f(v) \leq l_2$ , then the two corresponding level sets are traversed by the same cross-faces. It follows that  $\mathcal{C}(l_1)$  can be deformed continuously into  $\mathcal{C}(l_2)$  and vice versa. In other words,  $\mathcal{C}(l_1)$  and  $\mathcal{C}(l_2)$  are *homotopic*. In particular, they have the same number of contours.

This result will be strengthened in the next section. As we will see, the contours are still homotopic when the level  $l$  passes through a value  $f(v)$ , as long as  $v$  is a *regular* vertex.

### 3 Vertex Index and Euler Formula

The LSD encodes the set of contours  $\mathcal{C}(l)$  as  $l$  evolves. Branching of the LSD can only occur when  $\mathcal{C}(l)$  changes of homotopy type. It is thus important to know for which value of  $l$  the topology of the polygonal contours  $\mathcal{C}(l)$  is modified. As we just saw, this can only happen when  $l$  equals  $f(v)$  for some vertex  $v$  of the polyhedron  $\mathcal{P}$ .

In order to describe the topological change at a vertex  $v$  we introduce the *index* of a vertex  $v$ . Let  $w_1, w_2, \dots, w_k$  be the  $k$  neighbors of  $v$  enumerated counterclockwise around  $v$ . As Takahashi et al. [TIS<sup>+</sup>95], we consider the number of sign changes,  $Sgc_f(v)$ , in the sequence  $(f(w_1) - f(v), f(w_2) - f(v), \dots, f(w_k) - f(v), f(w_1) - f(v))$ . The index of  $v$  is defined as

$$Ind_f(v) = 1 - \frac{Sgc_f(v)}{2}$$

Figure 4 shows a vertex with index  $-1$ . This index tells how  $f$  is changing around  $v$ . It

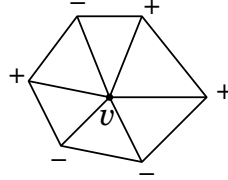


Figure 4: A “+” (a “-”) indicates a  $v$  neighbor,  $w$ , such that  $f(w)$  is greater (smaller) than  $f(v)$ . The number of sign changes  $Sgc_f(v)$  counts the number of “+” and “-” sequences.

actually records part of the topology of  $\mathcal{P}$  itself. By summing the indices over all the vertices, we get the very nice formula

#### Property 2

$$\sum_{v \in \mathcal{P}} Ind_f(v) = \chi(\mathcal{P})$$

where  $\chi(\mathcal{P}) = n_V - n_E + n_F$  is the Euler characteristic of  $\mathcal{P}$  and  $n_V$ ,  $n_E$  and  $n_F$  are respectively the number of vertices, edges and faces of  $\mathcal{P}$ .

**proof:**  $Sgc_f(v)$  counts the number of triangles  $(v_i, v, v_j)$  incident to  $v$  such that  $f(v)$  is between  $f(v_i)$  and  $f(v_j)$ . Since every triangle has exactly one vertex for which this occurs the sum of the  $Sgc_f(v)$  for all the vertices counts the total number of triangles:

$$\sum_{v \in \mathcal{P}} Sgc_f(v) = n_F,$$

so that

$$\sum_{v \in \mathcal{P}} Ind_f(v) = n_V - \frac{1}{2}n_F.$$

As  $\mathcal{P}$  is a triangulation, we have  $3n_F = 2n_E$  or equivalently  $n_F = 2(n_E - n_F)$ . It follows that

$$\sum_{v \in \mathcal{P}} Ind_f(v) = n_V - n_E + n_F = \chi(\mathcal{P}) \quad \square$$

The same formula can be found in [Hen79] in a slightly different context. Takahashi et al. [TIS<sup>+</sup>95] implicitly refer to a similar formula but they only consider vertices for which  $Sgc_f(v)$  equals 0 or 4. In fact, Takahashi et al. directly refer to Morse theory. In this theory a critical point may only get three different index values, two of which corresponding to local extrema. A more effective analogy is provided with the notion of *index of a critical point of a gradient vector field*. A critical point of a vector field is a (isolated) point where the vector field is null. Very roughly speaking the index of a critical point is the number of revolutions made by the vector field while turning around that point [Hen79]. In order to show the analogy with our notion of index we first orient the edges according to the  $f$ -values of their endpoints. This orientation corresponds to the *gradient* of  $f$  along the edges. Note that a triangle must have two edges oriented the same way and a third edge oriented the other way as shown in Figure 5. Then, in each triangle of  $\mathcal{P}$  we draw a continuous vector field, as sketched on Figure 5, so that the direction of the vector field on the triangle boundary agrees with the orientation of its edges. Now, the vertices of  $\mathcal{P}$  are the critical points of this vector field and it can be shown that their index, relatively to the vector field, coincides with our notion of index. The above Euler formula thus corresponds to a classical summation formula over the index of the critical points of a continuous vector field [Hen79].

The index of a vertex  $v$  describes how the level set realization  $\mathcal{C}(l)$  is modified around  $v$ , as the level  $l$  is raised from  $f(v) - \epsilon$  to  $f(v) + \epsilon$ , for a sufficiently small  $\epsilon$ .

If  $Ind_f(v)$  is null, then the  $v$  neighbors are partitioned into two connected sets  $\{v_- | f(v_-) < f(v)\}$  and  $\{v_+ | f(v_+) > f(v)\}$  as shown in Figure 6 on the left.  $\mathcal{C}(f(v) + \epsilon)$  is deduced from  $\mathcal{C}(f(v) - \epsilon)$  by substituting points on cross-edges of the type  $(v, v_-)$  to points on cross-edges of the type  $(v, v_+)$ . The change reduces to a sweep and a chain substitution of cross-edges so that  $\mathcal{C}(f(v) + \epsilon)$  is still homotopic to  $\mathcal{C}(f(v) - \epsilon)$ .

Several chain substitutions may occur at the same time when different non adjacent vertices get the same value  $f(v)$ .

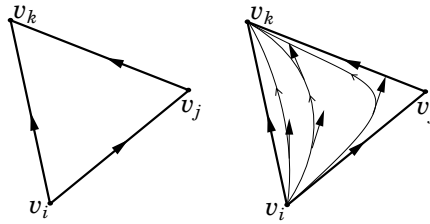


Figure 5: On the left: Orientation of the edges of a triangle induced by  $f$  ( $f(v_i) < f(v_j) < f(v_k)$ ). On the right: The tangent vectors to the oriented curves defines a vector field compatible with the edge orientation.

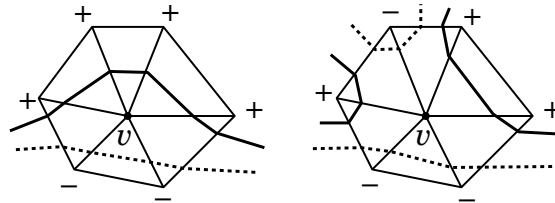


Figure 6: The neighborhood of a vertex  $v$  in two different configurations. a '+' ('-') indicates a  $v$  neighbor,  $w$ , such that  $f(w)$  is greater (smaller) than  $f(v)$ . The level set realization is shown for a level slightly below  $f(v)$  (dotted curve) and slightly above  $f(v)$  (plain curve). The index of  $v$  is 0 on the left and -1 on the right.

**Definition 4** A vertex with a null index is said to be regular and critical otherwise. Similarly the value  $f(v)$  of a critical vertex is called a critical value. Any other value is said to be regular.

Note that a positive index (+1) indicates a local extremum. A critical vertex which is not an extremum is called a *saddle* vertex. When traversing a critical value the topology of the level set is changing. The evolution of the level set depends on the index of the corresponding critical vertex (see Figure 6). The precise topological change not only depends on the index but also on the whole configuration of the level set. Figure 7 shows two possible changes in the level sets that depend on the configuration of  $\mathcal{C}(f(v) - \epsilon)$ . As will be shown in

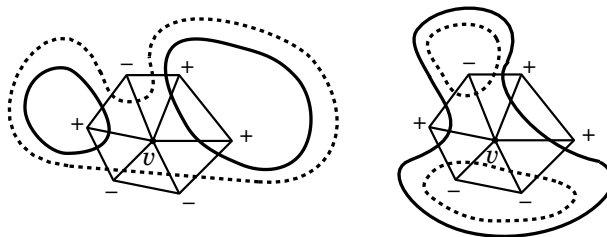


Figure 7: Two configurations of a vertex with index  $-1$ . The number of connected components of the level set increases on the left while it decreases on the right as the level is augmented from below  $f(v)$  (dotted curve) to above  $f(v)$  (plain curve).

the next section the topological changes of the level sets can be deduced directly from the vertex indices when considering the shortest path distance to a source point on a genus-0 polyhedron.

## 4 The Shortest Path Distance to a Source Point

Up to this point we have not considered any specific  $f$ . As we want the LSD to represent the object geometry, the definition of  $f$  should involve the geometry of  $\mathcal{P}$ . Takahashi et al. use the height function [TIS<sup>+</sup>95], so that any vertex is associated its own z-coordinate in a suitable coordinate system. In this case, the level sets correspond to cross-sections between  $\mathcal{P}$  and  $xy$ -planes. As noticed in the introduction this choice is relevant for terrain data as the critical vertices correspond to peaks, passes or pits of the terrain. However, for other classes of objects it may happen that the height function does not give an intuitive LSD in any coordinate system. Figure 13 shows a self-intersecting tubular objects whose “natural” axis could not be determined with any height function. A better choice for the level sets is provided with the moving front of some flow emanating from a tip of the tube. Such fronts correspond to the level sets of the geodesic distance to a tip of the tube. The computation of the geodesic distance to a source point on a polyhedron has been extensively studied (see [MMP87]). We prefer to compute the shortest path distance relatively to the graph induced

by the edges of a polyhedron. It provides an approximation, sufficient for our purpose, of the geodesic distance. It further allows us to simplify and accelerate the computation and fits nicely into a theoretical framework.

In the following, we study the level sets of the shortest path distance to a given source point. We will note the function  $d_s$  instead of  $f$ . For any vertex  $v$ ,  $d_s(v)$  is the length of a shortest path to a given source point  $s$ . (The choice of  $s$  will be discussed in the results section.)  $d_s$  can be computed efficiently with a modified breadth-first search starting from the source and using the classical Dijkstra's algorithm [CLR90]. Dijkstra's algorithm makes use of a priority queue to select the next vertex to be processed in the breadth-first search. Dijkstra's algorithm also returns for any vertex  $v$  a pointer  $\pi(v)$  to the preceding vertex on a shortest path linking  $v$  to the source  $s$ . By iterating the  $\pi$  pointer we can follow a strictly decreasing path from any vertex to the source.

$d_s$  presents interesting properties that allows for easy computation of the associated LSD. First of all, the source point  $s$  is the only local minimum of  $d_s$ . This means that from any vertex we can draw a path of edges leading to the source point with strictly decreasing vertex  $d_s$ -values. To obtain further nice properties we will assume from now on that the polyhedron  $\mathcal{P}$  has a 0-genus, that is  $\mathcal{P}$  is topologically equivalent to a triangulated sphere.

With this assumption, we get the following property that will provide a complete description of the topological change when the level passes through a critical value.

**Property 3** *The set of cross-faces incident to  $v$  for a level  $l$  lower than  $d_s(v)$  belong to a unique cycle.*

**proof:** Since  $\mathcal{P}$  has a 0-genus, the realization of any level set cycle splits  $\mathcal{P}$  into two connected components. If such a cycle contains a cross-face incident to  $v$ , then one of these two components must contains  $v$  while the other contains the source point. This is easily seen from the above decreasing path property (see also Figure 8). Now, any decreasing path  $p$

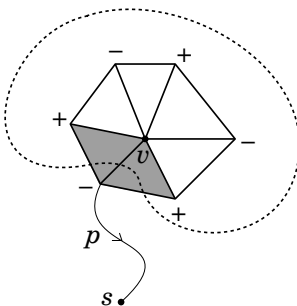


Figure 8: The path  $p : v \rightarrow s$  meets the realization of any level set cycle containing one or more cross-faces (shaded on the Figure) incident to  $v$  once and only once.

from  $v$  to  $s$  should meet every cycle containing a cross-face incident to  $v$ . Since  $p$  meets the whole level set  $\mathcal{C}(l)$  exactly once, there must be only one such cycle  $\square$

From this property it is seen that the configuration of Figure 7 on the right cannot occur. Also if  $c(l)$  counts the number of cycles of  $\mathcal{C}(l)$ , then for any vertex  $v$ , we have

$$c(d_s(v) + \epsilon) = c(d_s(v) - \epsilon) - \text{Ind}_{d_s}(v),$$

since any chain of  $v$  neighbors further than  $v$  from the source gives rise to a new cycle (see Figure 7 left).

Property 3 prevents the LSD from having loops. The vertex with the highest distance  $d_s$  on a loop would indeed have two branches going down the source which would contradict property 3. Since the LSD is connected (just follow a decreasing path to the source from any vertex on a cycle), the LSD must be a tree.

The tree structure of the LSD is recorded into a tree  $\mathcal{T}_s(\mathcal{P})$  whose root, internal nodes and leaves are respectively the source  $s$ , the saddle vertices, and the local maxima. The lines of  $\mathcal{T}_s(\mathcal{P})$  link critical vertices and correspond to cylindrical parts of  $\mathcal{P}$ . Figure 9 shows a simple example.

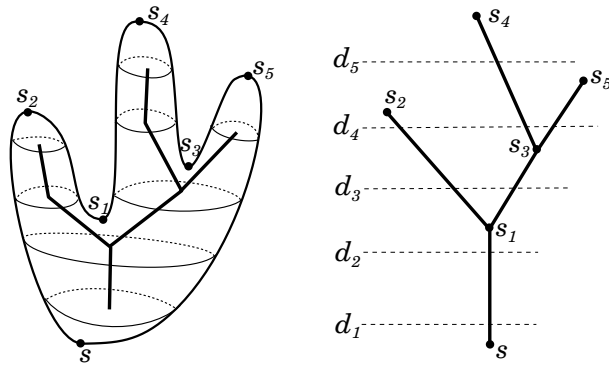


Figure 9: The LSD is an embedding of the tree  $\mathcal{T}_s(\mathcal{P})$ . On the left the LSD is computed from five level sets of  $d_s$  corresponding to the distances  $d_1, \dots, d_5$ .

In practice, we will use its tree structure to construct the LSD.

## 5 Algorithms

We present algorithms for computing the tree  $\mathcal{T}_s(\mathcal{P})$  and the LSD associated with the distance  $d_s$ . To clarify the presentation we assume that no two vertices of  $\mathcal{P}$  are at the same distance from the source. In case this would happen, we can still order the vertices according to their order of processing. We also assume that the neighbors of a vertex, the incident faces to an edge, and the edges and vertices of a face can be accessed in constant time  $\mathcal{O}(1)$ . The winged-edge data structure [Wei85], for instance, fulfills all these requirements.

To construct the LSD, we need to sweep the level from zero to its maximum value  $\max d_s$  and compute the “average” point of various level set cycles.

The set of cross-faces contained in a level set changes whenever the level passes through a vertex value. The number of different level sets is thus equal to the number of vertices. Since most of the vertices are ought to be regular most of these level sets will have the same topology. Also, polyhedral objects may have hundreds or thousands of vertices, and we do not need that many level sets to construct the LSD. In practice, we fix the number of levels (from 16 to 50 in our examples) and select the levels uniformly in the range  $[0, \max d_s]$ . For each selected levels, we must compute the corresponding level set realization. This requires finding a starting cross-edge in each cycle of the level set. Then, from faces incident to these starting cross-edges, we can walk the dual graph of  $\mathcal{P}$  along the level set and determine all the level set cycles.

The tree structure  $\mathcal{T}_s(\mathcal{P})$  of the LSD actually provides an efficient tool to determine a starting edge in each cycle of any level set. Precisely, we build the LSD while traversing the lines of  $\mathcal{T}_s(\mathcal{P})$ . For each line  $(s_i, s_j)$  of  $\mathcal{T}_s(\mathcal{P})$  we can compute a starting cross-edge in any cycle contained in the associated cylindrical part. This is done with a decreasing path starting at  $s_j$  as shown in Figure 10.

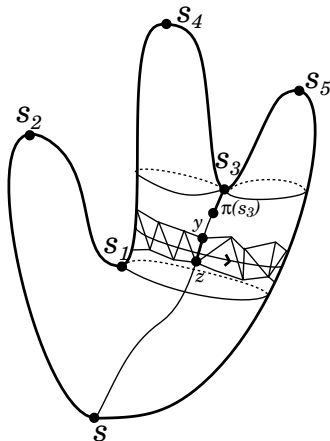


Figure 10: The  $\pi$  pointer provided by the Dijkstra's algorithm is used to find a starting cross-edge  $(y, z)$  of a cycle contained in the cylindrical region between  $s_1$  and  $s_3$ .

As the construction algorithm is guided by the tree structure  $\mathcal{T}_s(\mathcal{P})$ , we can assure that we do not miss any cycle in a level set and that each cycle is computed only once.

The overall LSD construction algorithm is thus based on two steps: in a first step, we build the tree  $\mathcal{T}_s(\mathcal{P})$ , and in a second step, we compute the LSD itself.

## 5.1 Computing $\mathcal{T}_s(\mathcal{P})$

The construction of  $\mathcal{T}_s(\mathcal{P})$  is based on Dijkstra's algorithm. It is performed while traversing the adjacency graph of  $\mathcal{P}$ . During this traversal, we compute the shortest path distance  $d_s$  of the visited vertices as well as the  $\pi$  pointer to the preceding neighbor on a shortest path. As for the classical Dijkstra's algorithm, we maintain a priority queue with respect to the distance. This priority queue contains the vertices adjacent to the already visited vertices, but whose distance have not yet been fixed by the algorithm.

Starting from the root, we visit the vertices of  $\mathcal{P}$  until we encounter a singular vertex  $v$ . At this point, we create a line  $(s, v)$  of  $\mathcal{T}_s(\mathcal{P})$ . If  $v$  is a local maximum the construction is stopped and  $\mathcal{T}_s(\mathcal{P})$  reduces to the single line  $(s, v)$ . Otherwise,  $v$  must be a saddle vertex, and we split the priority queue into  $1 - \text{Ind}_{d_s}(v)$  queues corresponding to the branches of  $\mathcal{T}_s(\mathcal{P})$ . We recursively continue this process on each branch until we reach all the maxima.

The algorithm proceeds as follows

```

0 (Initialize)
   $Q$  : a priority queue of vertices w.r.t.  $d_s$ 
  For each vertex  $v$  of  $\mathcal{P}$ 
    mark  $v$  as unprocessed.
   $d_s(s) = 0$ 
   $\pi(s) = \text{NULL}$ 
   $Q = \{s\}$ 
  insert  $s$  as the root of  $\mathcal{T}_s(\mathcal{P})$ 

```

```

1 (Main call)
begin
   $\text{addTreeLine}(Q, s)$ 
  return  $\mathcal{T}_s(\mathcal{P})$ 
end

```

$\text{addTreeLine}(L, \sigma)$  is a recursive procedure. Intuitively, it moves the front of vertices stored in the priority queue  $L$  until the next critical vertex  $v$  is met. The line  $(\sigma, v)$  is then added to  $\mathcal{T}_s(\mathcal{P})$  and  $L$  is split according to the index of  $v$ .

```

procedure  $\text{addTreeLine}(L, \sigma)$ 
begin
1  while  $L$  is not empty do
    begin
    extract  $v = \text{head of } L$  and dequeue  $L$ 
    Mark  $v$  as processed
5    for  $v_i$  adjacent to  $v$  and marked unprocessed do
        if  $v_i$  is not in  $L$  then
            begin
            set  $d_s(v_i)$  to  $d_s(v) + \|v - v_i\|$ 
            set  $\pi(v_i)$  to  $v$ 
10            insert  $v_i$  into  $L$ 
            end
        else if  $d_s(v_i) > d_s(v) + \|v - v_i\|$  then
            begin
            change  $d_s(v_i)$  to  $d_s(v) + \|v - v_i\|$ 

```



```

15         change  $\pi(v_i)$  to  $v$ 
           end
           compute the index  $Ind(v)$ 
           if  $Ind(v) = 1$  ( $v$  is a local maximum) then
             insert line  $(\sigma, v)$  into  $\mathcal{T}_s(\mathcal{P})$  with  $v$  as a leaf
20         else if  $Ind(v) < 0$  ( $v$  is a saddle point) then
           begin
             insert line  $(\sigma, v)$  into  $\mathcal{T}_s(\mathcal{P})$ 
             split  $L$  into  $1 - Ind(v)$  subqueues  $L_j$ 
             corresponding to the cycles incident to
25              $v$  of the level set  $\mathcal{C}(d_s(v) + \epsilon)$ 
             for  $1 \leq j \leq 1 - Ind(v)$  do
               addTreeLine( $L_j, v$ )
           end
         end
       end
     end

```

**Analysis** In the following we sketch some proof for correctness of the algorithm and analyze its complexity.

**Statement 1**  *$Ind(v)$  can be correctly computed at line 17 of the procedure  $addTreeLine$ .*

The classical Dijkstra's algorithm uses only one priority queue and vertices are dequeued in distance order [CLR90]. In our case this is still true during each execution of the procedure  $addTreeLine$ . In other words we get the same property while traversing a cylindrical part of  $\mathcal{P}$  associated to a branch of  $\mathcal{T}_s(\mathcal{P})$ .

When dequeuing a vertex  $v$  (line 3) all the adjacent vertices with a distance smaller than  $d_s(v)$  have already been processed while all the other adjacent vertices must lie in the queue with their distance set to a value greater than  $d_s(v)$ . It follows that  $Ind(v)$  can be correctly computed.

**Statement 2** *When  $L$  is dequeued (line 3 of  $addTreeLine$ ) it contains the extremity of all the cross-edges, relatively to the level  $d_s(v) + \epsilon$ , of a unique cycle, and at least one of these cross-edges has  $v$  as its origin.*

Suppose  $addTreeLine$  is run for the first time, so that the queue has never been split. As for Dijkstra's algorithm, vertices are dequeued in distance order. When  $L$  is dequeued in line 3, the origin of every cross-edge, relatively to the level  $d_s(v) + \epsilon$ , lies at a distance smaller than  $d_s(v)$  from the source ( $\epsilon$  is chosen so that no vertex other than  $v$  has its distance in the interval  $[d_s(v), d_s(v) + \epsilon]$ ). All these origins must therefore have been processed earlier. It follows that the extremity of these cross-edges have been enqueued at line 10. Also, as we have not yet encountered any critical vertex, the level set is composed of a unique cycle, that obviously contains a cross-edge incident to  $v$ .

Assume now, that  $addTreeLine$  is called with a queue,  $L$ , composed of a unique cycle containing a cross-edge incident to  $\sigma$ . A reasoning similar to the previous one shows that the statement remains true each time  $L$  is dequeued in line 3. In line 23, when the first

critical vertex is dequeued, we split  $L$  into  $1 - \text{Ind}(v)$  subqueues  $L_j$  corresponding to the cycles incident to  $v$  of the level set  $\mathcal{C}(d_s(v) + \epsilon)$ . To do so, we walk along each cycle of  $\mathcal{C}(d_s(v) + \epsilon)$  corresponding to a chain of '+'s (see Figure 7 Right) and store the extremity of the cross-edges in a separate queue. In other words we create the  $L_j$ 's from scratch each time we meet a critical vertex  $v$  rather than splitting  $L$  in effect. By construction, each queue  $L_j$  contains the extremity of all the cross-edges relatively to the level  $d_s(v) + \epsilon$  of a unique cycle, and at least one of these cross-edges has  $v$  as its origin. *addTreeLine* is further called with each queue  $L_j$  and the statement follows by induction on the calls to *addTreeLine*.

**Statement 3** *Procedure addTreeLine correctly computes the tree  $\mathcal{T}_s(\mathcal{P})$ .*

From statement 2 it is seen that each individual call to *addTreeLine* correspond to the sweep of a unique level set cycle between two critical vertices.

Finally, the recursive calls to *addTreeLine* perform a depth first search of  $\mathcal{T}_s(\mathcal{P})$  so that it visits the entire tree.

The complexity of the algorithm reduces to the complexity of the Dijkstra's algorithm plus the time for splitting the queue. Using a Fibonacci heap for the priority queue, Dijkstra's algorithm requires time  $\mathcal{O}(n \log n)$  for a polyhedron with  $n$  vertices<sup>2</sup>. The queue must be split each time a saddle vertex is met. The total time spent for splitting the queue is thus bounded by  $n_s \lambda_{max}$ , where  $n_s$  counts the number of saddle vertices and  $\lambda_{max}$  is the largest number of cross-edges in a level set. Hence, the computation of  $\mathcal{T}_s(\mathcal{P})$  requires time  $\mathcal{O}(n \log n + n_s \lambda_{max})$ .

## 5.2 Computing the LSD

The LSD constitutes a geometric embedding of  $\mathcal{T}_s(\mathcal{P})$ . In practice, we discretize the level interval  $[0, \max d_s]$  into a set  $\{l_1, l_2, \dots, l_r\}$  of  $r$  levels and compute the average point of each cycle of the level set realizations  $\mathcal{C}(l_1), \dots, \mathcal{C}(l_r)$ . The algorithm proceeds by traversing the tree  $\mathcal{T}_s(\mathcal{P})$ . For each line of the tree we compute and average the cycles of level  $l_1, \dots, l_r$  contained in the corresponding cylindrical part.

**(Main loop)**

```

begin
1 For each line  $(\sigma, \sigma')$  of  $\mathcal{T}_s(\mathcal{P})$  do
    begin
         $x = \sigma'$ 
        For each level  $l_i$  with  $d_s(\sigma) < l_i \leq d_s(\sigma')$ 
5         taken in decreasing order do
             $(c_i, x) = \text{processCycle}(l_i, x)$ 
            create a skeletal curve interpolating the points  $c_i$ 
        end
    end
end

```

<sup>2</sup>Dijkstra's algorithm works with any connected graph in time  $\mathcal{O}(n \log n + p)$  if  $p$  is the number of edges.

```

procedure processCycle( $l, z$ )
begin
1    $y = z$ 
    $z = \pi(y)$ 
   while  $d_s(z) > l$  do
     begin
5     $y = z$ 
        $z = \pi(y)$ 
     end
   traverse the cycle of level  $l$  from cross-edge  $(y, z)$ 
   and obtain a polygonal contour realization  $C$ 
10  compute  $c$  as the barycenter of  $C$ 
   return  $(c, y)$ 
end

```

Figure 10 summarizes the construction of a cycle in *processCycle*. In line 10 of *processCycle* we compute the barycenter of the polygonal curve  $C$  rather than the barycenter of its vertices. If  $C = (p_1, p_2, \dots, p_k)$  then

$$c = \sum_{j=1}^k \|p_j p_{j+1}\| \frac{p_j + p_{j+1}}{2} / \sum_{j=1}^k \|p_j p_{j+1}\|$$

This formula is independent of the discretization level and avoids the artifacts caused by non-uniform meshes.

Remark that in line 4 of the main loop we replaced the inequalities  $d_s(\sigma) < l_i < d_s(\sigma')$  involved in the definition of a cross-edge by the inequalities  $d_s(\sigma) < l_i \leq d_s(\sigma')$ . This avoids precision problems when  $l_i$  equals  $d_s(v)$  for some vertex  $v$ . In such a case, the *topology* of the level set will be the same as the topology of  $\mathcal{C}(l_i - \epsilon)$ , though the embedding might have a singular point at vertex  $v$ .

In line 7 of the main loop we can use a simple polygonal curve or a curve fitting algorithm [Sch90, Eil94]. Note that our algorithm provides a curve for each branch of the LSD but does not tell how the branches are geometrically linked together. The problem is that the barycenter becomes discontinuous each time a cycle is split into several cycles. In other words, the position of the barycenter of every new cycle changes abruptly from the position of its “parent” cycle. This branching information is yet important for animation and deformation purposes. Therefore, we keep virtual links between the branches according to the tree structure. In the Figures of the results Section branching links are visualized with line segments.

## 6 Results

We present and discuss the results of applying our LSD algorithm to a number of polyhedral objects.

**Source point location.** In our current implementation the source point can be selected either automatically or interactively. In the former case we apply a heuristic based on a

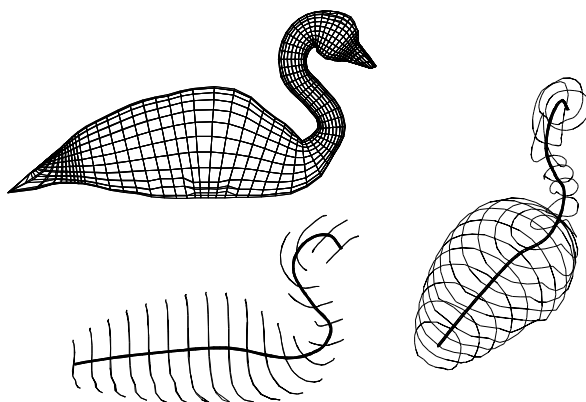


Figure 11: A swan and its LSD. The swan is composed of 1728 faces. 20 level lines have been built from 5% to 90% of the maximal distance from the source point which is located at the extremity of its beak (indicated by a white triangle on figure 12). The LSD was obtained in less than a second on a SGI INDIGO2.

simple algorithm to find a diameter of a tree. A diameter of a tree is a pair of vertices separated by the largest possible distance in the tree. Intuitively, the two vertices of a diameter are located at the tip of a longest branch of the tree. The following procedure determines a tree diameter:

1. Pick a vertex  $x$  at random in the tree,
2. Find  $y$  such that  $d_x(y)$  is maximal,
3. Find  $z$  such that  $d_y(z)$  is maximal.

It is relatively easy to show that  $(y, z)$  is a diameter. In our case we keep the two first steps of the procedure to find a source point  $s$  ( $= y$ ) which corresponds to the tip of some long branch of our polyhedral object. The source point was selected automatically on Figures 11, 15, and 16.

The shortest path distance is continuous with respect to the source point since

$$|d_s(x) - d_{s'}(x)| \leq d_s(s') = d(s, s').$$

As a consequence, the LSD is stable with respect to the source point location, as illustrated Figure 12. Note that the 0-level set coincides with the source point. In order to avoid small deviations due to the precise location of the source point we only draw part of the LSD. We actually remove a small percentage of the LSD at each of its extremities. On Figure 13, on the right, we can see the artifact caused by the selection of the source point on the border of the bottom knot section (the two end-sections are triangulated without any interior point).

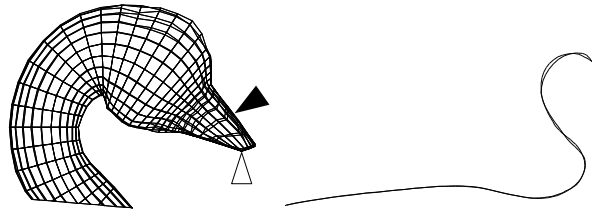


Figure 12: Left: the white and black arrows designate the two source points used to compute the LSDs. Right: the two corresponding LSDs are almost identical.

**Self intersecting objects.** Figure 13 shows that our LSD algorithm can handle self-intersecting objects. This would not be the case if we had chosen a height function instead of the shortest path distance. The Medial Axis transform would also fail to extract the correct structure of a self intersecting object. The reason is that shortest path distances only rely on edge connectivity and lengths (the metric) as opposed to a proper embedding of the vertices.

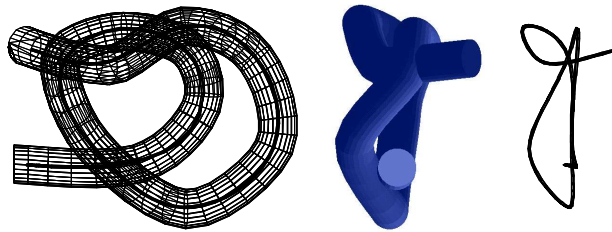


Figure 13: Left: front view of a self intersecting knot and its LSD. Middle: lateral shaded view of the knot. Right: the LSD seen from the same viewpoint.

**Extension to non triangulated objects.** Notice that the swan and the knot on Figures 11 and 13 are not triangulated. In [Laz95] an extension for extracting LSDs of non triangular meshes is proposed. In a first step we add virtual edges to the polyhedron graph so that the restriction of the new graph to any face is a complete graph. The shortest path distance is computed with this new graph. The definition of a cross edge is also modified to take into account the fact that a face may have more than two cross-edges (see [Laz95] for more details).

The LSDs of the man, the dolphin and the horse<sup>3</sup> of Figures 14, 15 and 16 have branches. 16 level have been used for the horse, 31 for the dolphin and 50 for the man.

<sup>3</sup>Copyright 1988, Rhythm & Hues Studios, Inc.

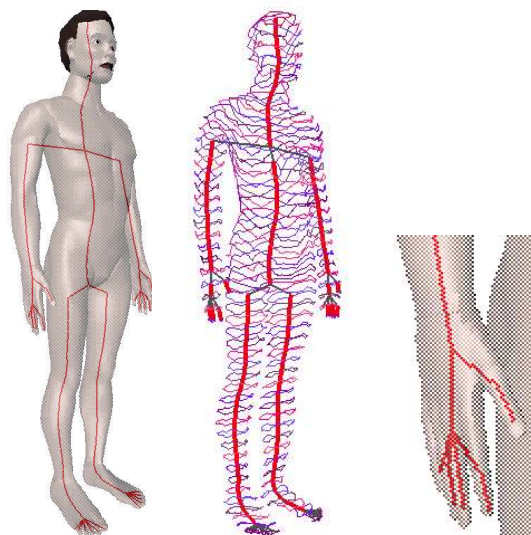


Figure 14: The man (left) is composed of 14946 faces. The source point was selected on the top of the man's head. The LSD (middle) was computed in 7.74 seconds on a SGI Indigo. Details of the LSD is shown on the right.

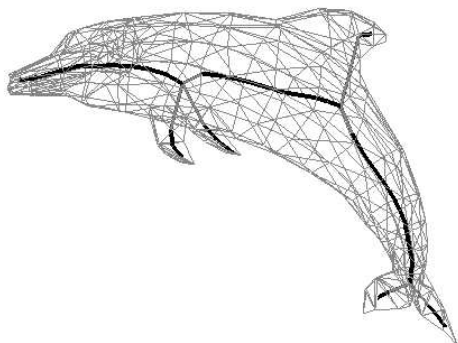


Figure 15: The dolphin is composed of 563 faces and the LSD was computed in 0.25 seconds on a SGI Indigo.

## 7 Discussions and Future Work

We have described an efficient algorithm for constructing the LSD associated to a genus-0 polyhedron whose vertices are valued by the shortest path distance to a source point. This

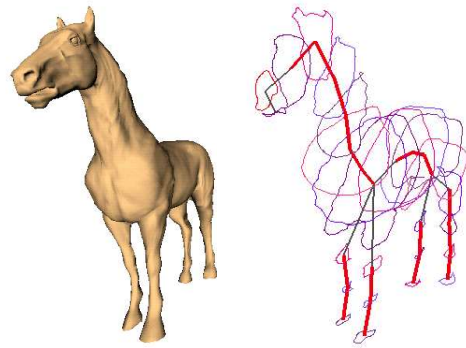


Figure 16: A horse and its LSD. The horse is composed of 22258 faces and the computation of the LSD lasted 5.41 seconds on a SGI Indigo.

algorithm is fast, robust, and easy to implement. It is mainly designed for tubular and branching objects although it works for any kind of shape. If the object has no obvious tree structure, as for the case of a spherical object, the LSD algorithm still produces a one dimensional structure. Depending on the application this may or may not be useful.

Note that the LSD may lie outside the object. Consider a punched sphere, forming a bowl, with the source point on its bottom (see Figure 17, left). The level sets are circles propagating from the source point up to the rim of the bowl and further shrinking down to the bottom on the interior surface of the bowl. The LSD of this bowl is composed of a straight line going outside the bowl interior up to the rim level, and turning back to the bottom. Although this might not correspond to the intuitive idea of a skeleton this LSD can be used for deforming the bowl (see Figure 17) with an axial deformation tool [LCJ94]. Generally

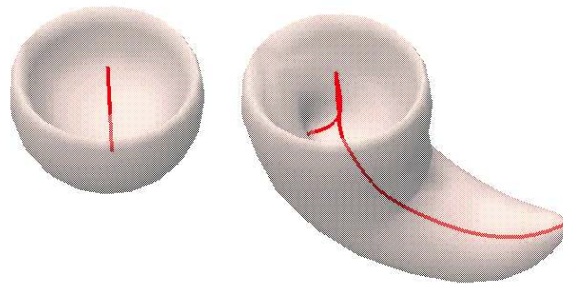


Figure 17: Left: the bowl and its corresponding LSD. Right: the deformed bowl after bending the LSD.

speaking, the LSD provides a set of tubular parameterizations together with articulations

that can be used for animation systems. For instance, Walter and Fournier's animation System [WF97] could certainly take advantage of the LSD.

Finding axial structures is also helpful for extraction of surfaces from scattered point sets (see [SK91]). In [VL97] we propose an algorithm, based on the level set paradigm, for extracting skeletal curves from 3D scattered data. It is especially relevant for medical data such as blood vessels.

We are currently working on extending this technique to non-0 genus surfaces. Shinagawa et al. [SKK91, TSK97] have described how to encode the level sets of a Morse function (the Reeb graph) in case of surfaces with non-0 genus, connected or not. However, except for geographical and cross-sectional data [SK91, TIS<sup>+</sup>95], they rather tackle the inverse problem of generating a surface from a given Reeb graph. When applied to scattered point sets, coupling our technique with the feature-based surface generation developed by Takahashi et al. [TSK97] would provide an efficient reconstruction tool.

## References

- [AM97] D. Attali and A. Montanvert. Computing and simplifying 2d and 3d semicontinuous skeletons of 2D and 3D shapes. *Computer Vision and Image Understanding*, 67(3):261–273, September 1997.
- [CLR90] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, New York, NY, 1990.
- [Eil94] P.H. Eilers. Smoothing and interpolation with finite differences. In *Graphics Gems IV*. Academic Press, 1994.
- [GYKD91] J.A. Goldak, X. Yu, A. Knight, and L. Dong. Constructing discrete medial axis of 3D objects. *International Journal of Computational Geometry and Applications*, 1(3):327–339, 1991.
- [Har97] J.C. Hart. Morse theory for implicit surface modeling. In *Visualization and Mathematics '97*, Berlin-Dahlem, September 1997.
- [Hen79] M. Henle. *a Combinatorial introduction to topology*. Freeman W.H., San Francisco, 1979.
- [Laz95] F. Lazarus. Courbes, cylindres et métamorphoses pour l'image de synthèse. Thèse de doctorat en sciences, Université PARIS VII, December 1995.
- [LCJ94] F. Lazarus, S. Coquillart, and P. Jancène. Axial deformations: an intuitive deformation technique. *Computer-Aided Design*, 26(8):607–613, August 1994.
- [LL89] F. Leymarie and M.D. Levine. Snakes and skeletons. Technical Report TR-CIM-89-3, McGill Research Centre for Intelligent Machines, January 1989.



- [Mil63] J. Milnor. *Morse theory*, volume 51 of *Annual of Mathematics Studies*. Princeton University Press, Princeton, NJ, 1963.
- [MMP87] J.S.B. Mitchell, D.M. Mount, and C.H. Papadimitriou. The discrete geodesic problem. *SIAM J. Comput.*, 16:647–668, 1987.
- [Sch90] P.J. Schneider. An algorithm for automatically fitting digitized curves. In *Graphics Gems*. Academic Press, 1990.
- [SK91] Y. Shinagawa and T.L. Kunii. Constructing a Reeb graph automatically from cross sections. *IEEE Computer Graphics and Applications*, 11(6):44–51, November 1991.
- [SKK91] Y. Shinagawa, T.L. Kunii, and Y. L. Kergosien. Surface coding based on Morse theory. *IEEE Computer Graphics and Applications*, 11(5):66–78, September 1991.
- [TIS<sup>+</sup>95] S. Takahashi, T. Ikeda, Y. Shinagawa, T.L. Kunii, and M. Ueda. Algorithms for extracting correct critical points and constructing topological graphs from discrete geographical elevation data. In *Eurographics'95*, pages C–181–C–192, Maastrich, The Netherlands, September 1995.
- [TSK97] S. Takahashi, Y. Shinagawa, and T.L. Kunii. A feature-based approach for smooth surfaces. In *Solid Modeling'97*, pages 97–110, Atlanta, Georgia, May 1997. ACM.
- [VL97] A. Verroust and F. Lazarus. Extracting skeletal curves from 3D scattered data. Rapport de Recherche INRIA no 3250, September 1997.
- [Wei85] K. Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, January 1985.
- [WF97] M. Walter and A. Fournier. Growing and animating polygonal models of animals. In *Eurographics'97*, Budapest, HU, September 1997.



---

Unit ´e de recherche INRIA Lorraine, Technop ˆole de Nancy-Brabois, Campus scientifique,  
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY  
Unit ´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex  
Unit ´e de recherche INRIA Rh ˆone-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN  
Unit ´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex  
Unit ´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

---

´Editeur  
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)  
<http://www.inria.fr>  
ISSN 0249-6399