Steepest descent paths on simplicial meshes of arbitrary dimensions

Mattia Natali[a,b], Marco Attene[a,*], Giulio Ottonello[c]

[a] *CNR-IMATI, Italy*
[b] *University of Bergen, Norway*
[c] *University of Genova, Italy*

Abstract

This paper introduces an algorithm to compute steepest descent paths on multivariate piecewise-linear functions on Euclidean domains of arbitrary dimensions and topology. The domain of the function is required to be a finite PL-manifold modeled by a simplicial complex. Given a starting point in such a domain, the resulting steepest descent path is represented by a sequence of segments terminating at a local minimum. Existing approaches for two and three dimensions define few ad-hoc procedures to calculate these segments within simplexes of dimension 1, 2 and 3. Unfortunately, in a dimension-independent setting this case-by-case approach is no longer applicable, and a generalized theory and a corresponding algorithm must be designed. In this paper, the calculation is based on the derivation of the analytical form of the hyperplane containing the simplex, independently of its dimension. Our prototype implementation demonstrates that the algorithm is efficient even for significantly complex domains.

1. Introduction

Being able to efficiently calculate the local minima of a real-valued function is of clear importance for numerous application contexts. In some domains, such as physics or chemistry, it is not rare to deal with problems where the function F to be minimized represents the status of a system, and its local minima represent particular configurations for which the system is *stable* or *in equilibrium*. It is thus interesting to know how such a system evolves towards a stable state when left unconstrained from an initial configuration. Such an evolution can be represented within the domain of F by a sequence of configurations that form a path connecting the initial state with the final stable state. Normally the system cannot jump from a configuration to another without continuity (though there are exceptions, e.g. in quantum mechanics) and thus the path is continuous as well.

For several natural phenomena, a given non-stable state tends to change towards equilibrium with maximum speed. Such a phenomenon can be modeled by a real-valued function F that associates an energy with each state of the system and, if F is differentiable, the direction of the maximum speed is the direction of the negative gradient of F. When the function is expressed through a closed analytical form, computing the gradient becomes a matter of symbolic calculus and thus is relatively easy. In contrast, when the function is described through a finite set of samples, one may need to employ more sophisticated techniques.

In this paper, we review the methods proposed so far to compute steepest-descent paths on piecewise linear surface meshes, and generalize them within a unified approach that works on simplicial complexes of any dimension. Furthermore, we provide a novel solution to resolve the cases of *indifferent equilibrium* that may arise when tracking the steepest descent path. Finally, we show how our solution can be exploited in the field of Material Sciences to study the crystallization of molten substances.

Note that generalizing existing algorithms is not as easy as it could appear at a first sight. On triangle meshes, for example, there are three possibilities: (1) the path passes through triangles, or (2) it may run along edges or (3) it may cross vertices. Thus, algorithms treating these models can enumerate a few possibilities and handle each of them as a particular case. Furthermore, the treatment of special cases of *indifferent equilibrium* can also be handled by enumerating the possibilities. Conversely, in a generalized approach such as the one introduced in this article, a dimension-

*Corresponding Author
Email address: marco.attene@ge.imati.cnr.it (Marco Attene)

independent solution must be provided which can treat all the cases based on a common approach. To the best of our knowledge no solution to this problem has been proposed so far, while there are application contexts (e.g. Material Sciences) that call for a solution.

1.1. Related work

The *steepest descent method*, also known as *gradient descent method*, is an algorithm to compute local minima of real-valued functions of n variables. While running, it constructs a piecewise-linear approximation of the steepest descent path connecting an initial point to a local minimum.

1.1.1. Steepest descent direction for differentiable functions

Formally, if $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ is a real function and $\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n})$ is its gradient, then the steepest descent direction for $\mathbf{x}_0 \in \mathbb{R}^n$ is given by the vector $d = -\nabla f(\mathbf{x}_0)$. When starting from an initial point $\mathbf{x}_0$, the steepest descent method computes a new point $\mathbf{x}_1 := \mathbf{x}_0 - \alpha \nabla f(\mathbf{x}_0)$. Then, in a second step it computes another point $\mathbf{x}_2 := \mathbf{x}_1 - \alpha \nabla f(\mathbf{x}_1)$, and so on. In general, the algorithm computes a sequence of points $\mathbf{x}_0 \ldots \mathbf{x}_k$ that is expected to converge to a local minimum. The positive parameter α might be kept constant or can change from step to step. Small values of α prevent the algorithm to oscillate around minima, but too small values might significantly slow down the entire process.

Near minima, the steepest descent method tends to go slowly and in some cases to have an erratic path. For these reasons, other minimization algorithms have been proposed: notable examples are the *conjugate gradient* that uses conjugate directions instead of the local gradient to take into account the previously-chosen directions, and the *Newton-Raphson* that exploits information of the second derivative to locate the minimum of the function.

With small modifications, the steepest descent method can be adapted to the case of piecewise-linear meshes [1] approximating differentiable functions. In this case, there is no need to choose the parameter α because each segment of the path is bounded by the size of mesh elements. As a consequence, the aforementioned drawbacks are no longer an issue. Unfortunately, though there are some methods for calculation on two ([1]) and three ([2, 3]) dimensional meshes, a complete adaptation is missing for the cases where differentiable functions are defined on n-dimensional simplicial domains.

1.1.2. Piecewise linear functions on simplicial meshes

When the domain is a two-dimensional simplicial mesh, several methods exist to calculate steepest descent paths. A big family of algorithms uses descent (and ascent) paths to bound the cells of the so-called *Morse-Smale complexes* [3, 2, 1]. Within this family, the methods described in [4, 5, 6] force the steepest descent paths to be sequences of mesh edges. Specifically, these approaches start at a given vertex, and at each step simply choose the neighboring vertex associated with the smallest function value. Though these algorithms are extremely efficient, they produce only approximate solutions. Conversely, in [7, 8] the paths are computed based on [1] and are free to cross the triangles, therefore produce much more precise results. These algorithms have applications in both Computer Graphics [9, 10] and Visualization [11].

Clearly, if the function is piecewise-linearly defined on a simplicial mesh, the gradient is generally undefined for points on edges and for the vertices, so it may be necessary to provide an estimate for these particular cases. If $\mathbf{p}_i$ is the position of the i^{th} vertex of a closed and manifold triangle mesh $\mathcal{M}$, then the gradient of the function $f : \mathcal{M} \longrightarrow \mathbb{R}$ can be estimated as [12]:

$$\nabla f(\mathbf{p}_i) := \sum_{j \in \mathcal{N}(i)} [f(\mathbf{p}_j) - f(\mathbf{p}_i)]\ \mathbf{w}_j\ , \qquad (1)$$

where the $\mathbf{w}_j$s are proper weights that do not depend on f, and $\mathcal{N}(i)$ is the set of vertices connected to i by an edge.

In contrast, the gradient ∇f can be computed exactly for points within triangles. Furthermore, ∇f does not change across the points within a given triangle. So, on a triangle t with vertices $(\mathbf{p}_i, \mathbf{p}_j, \mathbf{p}_k)$ and unit normal $\mathbf{N}$, the gradient is the solution $\nabla f|_t$ of the 3×3 linear system [13]:

$$\begin{pmatrix} \mathbf{p}_j - \mathbf{p}_i \\ \mathbf{p}_k - \mathbf{p}_j \\ \mathbf{N} \end{pmatrix} \nabla f|_t = \begin{pmatrix} f(\mathbf{p}_j) - f(\mathbf{p}_i) \\ f(\mathbf{p}_k) - f(\mathbf{p}_j) \\ 0 \end{pmatrix} \qquad (2)$$

Therefore, alternatively to Eqn. 1, the steepest direction at the vertex position $\mathbf{p}_i$ can be estimated by considering its incident triangle's gradient having largest magnitude.

The most detailed description of an algorithm that computes the steepest paths is given for the 2-dimensional case in [1]. Therein, the path can go along edges or pass through faces of the triangulation. Three cases occur for a starting point $\mathbf{p}$ of the path: (1) $\mathbf{p}$ is in the interior of a triangle - the steepest direction is unique and orthogonal to the level lines (eg. Eqn. 2); (2) $\mathbf{p}$ is

in the interior of an edge - one or two locally steepest directions are possible; (3) **p** is a vertex - there may be as many locally steepest directions as the number of its incident triangles.

The case of 3-dimensional manifold domains is treated in [2, 3], where smooth manifolds are approximated with piecewise linear simplicial complexes that linearly interpolate samples of the function f. Both in [3] and in [2] ascending/descending arcs are defined as piecewise linear curves (1-manifolds) between saddles and maxima/minima lying along edges of the input mesh.

2. Definitions and problem statement

In this section basic definitions are introduced to support a formal description of the algorithm. The following definitions are adapted from [14] and [15].

2.1. Simplicial complexes

A *k-dimensional simplex*, or *k-simplex*, A^k is a set $V = \{v_0, \ldots, v_k\}$ of $k+1$ objects called *vertices*, together with the set of real-valued functions $\alpha : V \to \mathbb{R}$ satisfying $\sum_{v_i \in V} \alpha(v_i) = 1$ and $\alpha(v_i) \geq 0$. A function α is called a point of A^k. The values $\alpha(v_0), \ldots, \alpha(v_k)$ are the *barycentric coordinates* of the point α.

A (proper) *face* B of A^k, denoted $B < A^k$, is a simplex determined by the (proper) subset $W \subset V$, whose points $\beta : W \to \mathbb{R}$ are identified with the points $\alpha : V \to \mathbb{R}$ such that $\alpha(v_i) = \beta(v_i)$ if $v_i \in W$ and $\alpha(v_j) = 0$ if $v_j \in V - W$. If B is a face of A, then A is said to be *incident* at B.

A *finite simplicial complex* K is a finite set of simplices such that:

i) if $A \in K$ and $B < A$, then $B \in K$;

ii) if $A, B \in K$, then $A \cap B$ is either empty or it is a face of both A and B.

From now on, we shall omit the term finite.

The *dimension* of K is the dimension of the largest dimensional simplex belonging to K. A simplicial complex of dimension n is *homogeneous* if it is made of n-simplices and their faces. Within an n-dimensional simplicial complex, a k-simplex is said to be *maximal* if $k = n$.

The *boundary* ∂A of a simplex A is the complex made of the proper faces of A. The *boundary* ∂K of a homogeneous n-dimensional simplicial complex K is the $(n-1)$-complex obtained as the sum mod 2 of the $(n-1)$-dimensional simplices of the boundary ∂A of each of the n-simplices $A \in K$ plus their faces.

L is a *subcomplex* of K if L is a complex and $L \subset K$. For $A \in K$, the (closed) *star* of A in K, $star(A, K)$, is the subcomplex of K made of all simplices of K having A as a face plus all their faces. If $A \in K$, then the *link* of A in K, $link(A, K)$, is the set of simplices in $star(A, K)$ whose intersection with A is empty.

A *geometric realization* $|A^k|$ of a simplex A^k in the Euclidean space $\mathbb{R}^n$, $n \geq k$, can be obtained by defining a bijection between the vertices of A^k and a set of $k+1$ affinely independent points $p_0, p_1, \ldots, p_k$ of $\mathbb{R}^n$, so that $|A^k| = \{(t_0 p_0 + t_1 p_1 + \ldots + t_k p_k) \in \mathbb{R}^n \mid t_i \geq 0,\ \sum_i t_i = 1\}$. Thus, $|A^k|$ is the convex hull of $p_0, \ldots, p_k$ and is said to be an *Euclidean simplex*.

The *underlying space* $|K|$ of K is the union $\cup_{A \in K} |A|$ of the geometric realization of its simplices. An underlying space $|K|$ of a k-dimensional simplicial complex K is unambiguously defined by a bijection between all the vertices of K and a corresponding set of points of $\mathbb{R}^n$, $n \geq k$, such that the image of the vertices of each simplex consists of affinely independent points.

2.2. Problem statement

Herewith, some novel definitions are introduced to better explain the problem and the proposed algorithm. In particular, we need to formally define sets of combinatorial entities that, after a geometric realization, assume a special relationship with a given Euclidean point.

Let $|K|$ be the underlying space of K, and let **x** be a point in it. We say that the *influence set* of **x** in K is the set of all the simplices whose geometric realization contains **x**. That is:

$$iset(\mathbf{x}, K) = \{\sigma_i \in K : \mathbf{x} \in |\sigma_i|\} \tag{3}$$

Furthermore, we say that the *boundary set* of **x** in K is the set of all the faces of simplices in $iset(\mathbf{x}, K)$ whose geometric realization does not contain **x**. That is:

$$bset(\mathbf{x}, K) = \{\sigma_i : \sigma_i < \sigma_j, \sigma_j \in iset(\mathbf{x}, K), \mathbf{x} \notin |\sigma_i|\} \tag{4}$$

For example, if we denote with t the geometric realization of a 2-simplex $\sigma \in K$, t is a triangle, the influence set of its barycenter is σ whereas its boundary set is made of the three vertices and the three edges being faces of σ. Notice that the influence set of the geometric realization of a vertex $|v|$ coincides with its open *star* (i.e. $star(v) \setminus link(v)$), and the boundary set of $|v|$ is exactly $link(v)$. It is worth mentioning, however, that we cannot simply replace *iset* and *bset* with the more standard *star* and *link*. Though a relation actually exists, in fact, *iset* and *bset* map arbitrary Euclidean points to

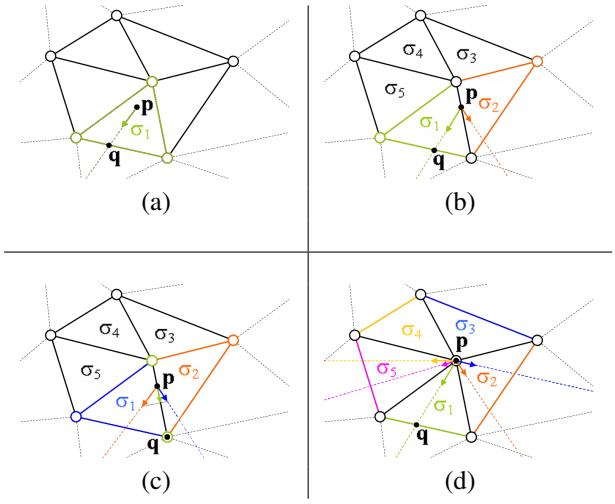


Figure 1: Possible positions of a starting point in a two dimensional complex. In (a), $iset(\mathbf{p}, K) = \{\sigma_1\}$ and just one direction must be evaluated to proceed to the next point $\mathbf{q}$. When $\mathbf{p}$ is on an edge, the i^{th} ray may (b), or may not (c), intersect a face of σ_i belonging to $bset(\mathbf{p}, K)$ (note that in (c) each of the two arrows points outwards wrt the corresponding σ_i). If $\mathbf{p}$ is a vertex (d), the cardinality of $iset(\mathbf{p}, K)$ increases and several directions must be considered.

subcomplexes, whereas the *star* and *link* map simplexes to subcomplexes, and a conversion makes sense only when the Euclidean point being mapped coincides with the geometric realization of a vertex, as in the aforementioned example.

Let K be a finite n-dimensional simplicial complex, V_K be the set of its vertices, and $r : V_K \to \mathbb{R}^n$ be a mapping function defining an underlying space $|K|_r$ which is an n-manifold with boundary (i.e. $|K|_r$ is a PL n-manifold [14] with boundary). Also, let us consider a function f^* which associates a real value with each vertex of K, and a corresponding map $f : |K|_r \to \mathbb{R}$ such that, if σ is a vertex $\mathbf{x} = r(\sigma) \implies f(\mathbf{x}) = f^*(\sigma)$, whereas if σ is a higher-dimensional simplex and $\mathbf{x} \in |\sigma|$ then the value of $f(\mathbf{x})$ is defined by linearly interpolating the value of f at its vertices.

Having said that, our problem can be summarized as follows: given an initial point $\mathbf{p}_0 \in |K|_r$, determine a sequence of points $\mathbf{p}_0 \dots \mathbf{p}_k$, $\mathbf{p}_i \in |K|_r$, such that $\mathbf{p}_{i+1} \in |bset(\mathbf{p}_i, K)|_r$ maximizes the quantity $\Delta_i = \frac{f(\mathbf{p}_i) - f(\mathbf{p}_{i+1})}{\|\mathbf{p}_{i+1} - \mathbf{p}_i\|_2}$ and $\Delta_i > 0$.

Thus, the objective of the algorithm is not just the determination of the local minimum, but the determination of the whole path that reaches that minimum while following the direction of steepest descent.

3. Algorithm description

In the easiest case, the algorithm starts with a given point $\mathbf{p}_0$ within a (maximal) n-dimensional simplex σ_0, computes the direction of steepest descent $-\nabla_0$ and shoots a ray from $\mathbf{p}_0$ in the direction of $-\nabla_0$. The ray intersects an $n-1$-dimensional simplex which is a face of both σ_0 and an opposite maximal simplex σ_1. Let $\mathbf{p}_1$ be the point of intersection and $-\nabla_1$ be the direction of steepest descent calculated within σ_1. Once again, the algorithm shoots a ray from $\mathbf{p}_1$ in the direction of $-\nabla_1$, intersects another $n-1$-dimensional simplex, and so on, as long as Δ_i is positive.

Actually, this is just a particularly easy case. In practice lower-dimensional simplices (i.e. of dimension $< n-1$) can be intercepted along the path, can contain the starting point $\mathbf{p}_0$, or can even contain whole parts of the path. Thus, a procedure is necessary for the calculation of the gradient vector within both maximal simplices (see section 4.1) and lower-dimensional simplices (see section 4.2). Furthermore, when the point $\mathbf{p}_i$ lies on a simplex of dimension $< n-1$, the gradient must be computed for all its incident simplices of any dimension, and the path proceeds on the simplex for which Δ_i is maximized.

Figure 1 illustrates the various possibilities for a starting point $\mathbf{p}$ when K is two-dimensional. In (a), $\mathbf{p}$ lies within a maximal simplex, so it is sufficient to calculate the gradient within the simplex to determine the next point $\mathbf{q}$ of the path, which might lie either on a vertex or on an edge of the simplex. In (b), $\mathbf{p}$ is on a 1-dimensional simplex, so it is necessary to compute the gradient for the simplex itself and for both its incident triangles, and choose the one that maximizes Δ_i; in this case Δ_i is maximum for one of the incident triangles. In contrast, in (c) $\mathbf{p}$ is on a *valley* line, so Δ_i is maximum on the 1-dimensional simplex. Note that in this case the ray shot from $\mathbf{p}$ towards the negative gradient of one of the incident triangles would not intercept any edge of the same triangle. Finally, in (d) $\mathbf{p}$ is on a 0-simplex, so all its incident simplices must be considered.

3.1. Construction pipeline

Observe that the assumption that $|K|$ is a manifold with boundary guarantees that $|bset(|v|, K)|$ is a topological sphere if v is an internal vertex, whereas it is a topological semi-sphere if v is on the boundary. Therefore, if v is internal, each ray emanating from it must necessarily intersect a point of $|bset(|v|, K)|$.

Let K be an n-dimensional mesh, with its geometric realization $|K|$ that is an n-dimensional PL-manifold with boundary in $\mathbb{R}^n$. Also, let $\mathbf{p}_0 \in |K|$ be the starting

point of the steepest descent path we are looking for. In the first step, the algorithm determines the set of simplices in the influence set of $\mathbf{p}_0$, that is, $iset(\mathbf{p}_0, K)$. Note that the cardinality of this set depends on the dimension of the minimal simplex whose realization contains $\mathbf{p}_0$. For example, if $\mathbf{p}_0$ corresponds to the realization of a vertex $|v|$, then all the simplices incident at v are in $iset(\mathbf{p}_0, K)$. On the other extreme, if $\mathbf{p}_0$ belongs to the interior of the realization of a maximal simplex, then only that simplex will be in $iset(\mathbf{p}_0, K)$. In any case, for each simplex σ_i in $iset(\mathbf{p}_0, K)$ we derive the gradient vector $(\nabla f)_{\sigma_i}$ as described in section 4, and compute the intersection $\mathbf{p}_1^i$ between $|bset(\mathbf{p}_0, K)|$ and the ray emanating from $\mathbf{p}_0$ in the direction of $-(\nabla f)_{\sigma_i}$. If such an intersection does not exist, it means that $\mathbf{p}_0$ is on the boundary of K and the computed direction would move the path outside the domain; thus, we declare $-(\nabla f)_{\sigma_i}$ to be an "invalid" direction. In general, a direction $-(\nabla f)_{\sigma_i}$ is declared to be "valid" if and only if the intersection $\mathbf{p}_1^i$ exists and belongs to a face of σ_i. For all the "valid" directions computed within $iset(\mathbf{p}_0, K)$, we compute the quantity $\Delta_i = (f(\mathbf{p}_0) - f(\mathbf{p}_1^i))/\|\mathbf{p}_1^i - \mathbf{p}_0\|_2$ and select the point $\mathbf{p}_1^j$ that maximizes it. If such a maximum quantity is negative, we have reached a local minimum. Otherwise, $\mathbf{p}_1 = \mathbf{p}_1^j$ becomes the new starting point for our path computation, and the algorithm iterates.

A pseudo-code describing the method is given in algorithm 1.

Algorithm 1 Steepest descent path construction.

Require: A simplicial complex K, the value of $f*$ at each vertex of K, and a point $\mathbf{p}_0$ on $|K|$.

Ensure: A sequence $\mathbf{p}_0 \dots \mathbf{p}_k$, $\mathbf{p}_i \in |K|$, s.t. $\mathbf{p}_{i+1} \in |bset(\mathbf{p}_i, K)|$ maximizes Δ_i and $\Delta_i > 0$.

```
1:  i = 0
2:  Δ_max = −∞
3:  compute iset(p_i, K)
4:  for each simplex σ_i^j in iset(p_i, K) do
5:      compute (∇f)_{σ_i^j}
6:      p_{i+1}^j := intersection between |bset(p_i, K)| and the
        ray from p_i in the direction of −(∇f)_{σ_i^j}
7:      if (p_{i+1}^j exists AND p_{i+1}^j is on a face of σ_i^j) then
8:          compute Δ_i = (f(p_i) − f(p_{i+1}^j))/||p_{i+1}^j − p_i||_2
9:          if (Δ_i > Δ_max) then
10:             Δ_max = Δ_i
11:             p_{i+1} = p_{i+1}^j
12:         end if
13:     end if
14: end for
15: if ( Δ_max > 0 ) then
16:     i++
17:     goto 2:
18: end if
```

3.1.1. Flat simplices

If the maximum Δ_i is zero, it means that we have reached a "flat" simplex having the same function value associated with all its vertices. In the case of flat simplices, we have adopted a strategy based on the following principle: if a simplex σ_F is flat but some of its neighbors are not, then such a *flatness* may be a consequence of the discretization of the domain of f, so we may consider it as a sort of *artefact*. In contrast, if both σ_F and all its neighbors are flat, then we assume that the function is well represented and is actually flat in that region. Therefore, when the path reaches a flat simplex σ_F (Fig. 2(a)), we consider the union of the influence sets of all its (realized) vertices, that we conveniently denote as $iset(|\sigma_F|, K)$ (Fig. 2(b)). For each simplex $\sigma_i \in iset(|\sigma_F|, K)$, $\sigma_i \neq \sigma_F$ and $\sigma_i \not< \sigma_F$, we check the flatness. If all the σ_is are flat, then the algorithm terminates. Otherwise, for each non-flat σ_i we compute the next point $\mathbf{p}_F$ as the barycenter of the maximal common face of σ_i and σ_F, then compute the successive point $\mathbf{p}_{F+1}$ and evaluate Δ_i based on it. If the maximum Δ_i among all the σ_is is negative the algorithm terminates, otherwise the corresponding $\mathbf{p}_F$ and $\mathbf{p}_{F+1}$ are added to the path (in this order) and the algorithm iterates (Fig. 2(c)).

This strategy to deal with flat simplices is based on the assumption that f^* is typically computed by sampling another continuous function g using a regular pattern. In cases such as those in Figures 3 and 12, for example, it is easy to find flat simplexes approximating parts of g which have an unambiguous steepness. This is even more evident when the samples are taken on contours: in Figure 6, for example, there are numerous flat triangles that actually *cut* descending crests and valley lines. Note that in all these cases, *isolated* flat triangles can be there just because of the regularity of the pattern used, and thus our approach to cross them gives the expected results.

4. Gradient evaluation

In order to implement the proposed algorithm, we need a procedure to derive the gradient vector of f at each point in its domain $|K|_r$. A point $\mathbf{p} \in |K|_r$ can either be the image of a vertex or belong to the image of a higher-dimensional simplex. We first consider the case where $\mathbf{p}$ belongs to the image of a *maximal* sim-

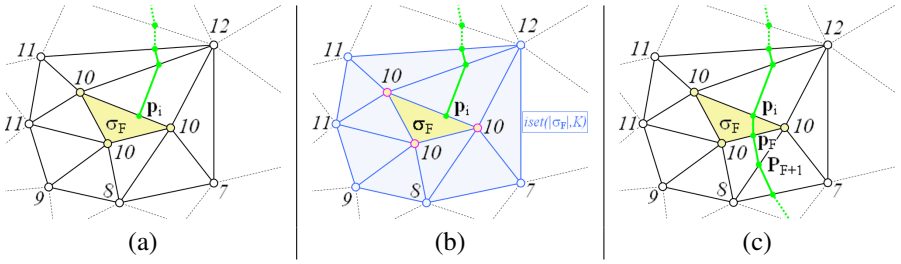


Figure 2: Proceeding on a flat region: (a) a flat simplex σ_F and (b) its corresponding $iset(|\sigma_F|, K)$; (c) p_F is the edge's midpoint (i.e. its barycenter) and p_{F+1} maximizes Δ_i.

plex (i.e. if K is n-dimensional, a simplex is maximal if its dimension is n). Afterwards, we generalize the result to lower-dimensional simplices through dimensionality reduction.

4.1. Evaluation in maximal simplices

Our aim is to find the gradient vector of f at all the points within an Euclidean simplex $|\sigma|$ defined by $n+1$ vertices, let them be $\mathbf{p}^1 = (p_1^1, \ldots, p_n^1), \ldots, \mathbf{p}^{n+1} = (p_1^{n+1}, \ldots, p_n^{n+1})$. In this section we derive the analytic form, inside $|\sigma|$, of the function f explicitly defined only at the vertices and assumed to be linear within the simplex. Such an analytical form is a hyperplane, thus the gradient vector is constant and can be easily derived by applying the definition:

$$\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \frac{\partial f}{\partial x_2}, \ldots, \frac{\partial f}{\partial x_n}).$$

To summarize, $f : |\sigma| \longrightarrow \mathbb{R}$ is explicitly defined just at the vertices and linear interpolation is assumed for the other (internal) points. Then, the analytic form of f within the simplex is computed (Section 4.1.1) and the n gradient vector coordinates are defined as the first n coefficients of such an analytic form.

4.1.1. Hyperplane by n points

Let $A_1, \ldots, A_n$ be n affinely independent points of the n-dimensional Euclidean space (eg. the vertices of an Euclidean simplex). The implicit equation of the hyperplane spanned by $A_1, \ldots, A_n$ can be obtained by expanding the determinant:

$$\begin{vmatrix} x_1 & \cdots & x_n & 1 \\ \leftarrow & A_1 & \rightarrow & 1 \\ & \vdots & & \vdots \\ \leftarrow & A_n & \rightarrow & 1 \end{vmatrix} = 0,$$

We observe that the value of f can be appended as an $(n+1)^{th}$ coordinate to each vertex of $|\sigma|$. Since f is piecewise-linear, the resulting n-dimensional Euclidean simplex spans a hyperplane Π_σ of $\mathbb{R}^{n+1}$.

To derive the analytic form of Π_σ we exploit the above determinant formulation by instantiating the points A_i with the enriched vertices of $|\sigma|$, that is:

$$\begin{vmatrix} x_1 & \cdots & x_n & f(x_1, \ldots, x_n) & 1 \\ p_1^1 & \cdots & p_n^1 & f(p_1^1, \ldots, p_n^1) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_1^{n+1} & \cdots & p_n^{n+1} & f(p_1^{n+1}, \ldots, p_n^{n+1}) & 1 \end{vmatrix} = 0.$$

By expanding this determinant, for example with respect to the first row, and making it equal to zero, we obtain:

$$\lambda_1 x_1 + \ldots + \lambda_n x_n - \lambda_{n+1} f(x_1, \ldots, x_n) + \lambda_0 = 0,$$

$\lambda_i \in \mathbb{R}$, which can be rewritten as

$$f(x_1, \ldots, x_n) = \frac{\lambda_1}{\lambda_{n+1}} x_1 + \ldots + \frac{\lambda_n}{\lambda_{n+1}} x_n + \frac{\lambda_0}{\lambda_{n+1}}.$$

The above equation is the analytic form of f inside the Euclidean simplex defined by points $\mathbf{p}^1, \ldots, \mathbf{p}^{n+1}$, thus its gradient vector is

$$\nabla f(\mathbf{x}) = (\frac{\partial f}{\partial x_1}, \ldots, \frac{\partial f}{\partial x_n}) = (\frac{\lambda_1}{\lambda_{n+1}}, \ldots, \frac{\lambda_n}{\lambda_{n+1}}).$$

From an algorithmic point of view, the entire process requires the computation of $n+1$ determinants of $(n+1) \times (n+1)$ matrices. If we denote

$$M = \begin{pmatrix} p_1^1 & \cdots & p_n^1 & f(p_1^1, \ldots, p_n^1) & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ p_1^{n+1} & \cdots & p_n^{n+1} & f(p_1^{n+1}, \ldots, p_n^{n+1}) & 1 \end{pmatrix},$$

thus λ_i is the determinant of the matrix obtained by eliminating the i-th column from M.

Example: We assume $n = 3$ and we want to find the gradient vector of function f inside a simplex defined by the four points $\mathbf{p}^1 = (\frac{1}{2}, \frac{3}{2}, 1)$, $\mathbf{p}^2 = (1, 1, 2)$,

$\mathbf{p}^3 = (2, 2, 0)$ and $\mathbf{p}^4 = (0, 0, 0)$, with respective values $f(\mathbf{p}^1) = 10$, $f(\mathbf{p}^2) = 2$, $f(\mathbf{p}^3) = 3$, $f(\mathbf{p}^4) = 0$. Thus the hyperplane equation (in $\mathbb{R}^4$, where unknows are x, y, z and $w = f(x, y, z)$) is defined by

$$\begin{vmatrix} x & y & z & f(x,y,z) & 1 \\ \frac{1}{2} & \frac{3}{2} & 1 & 10 & 1 \\ 1 & 1 & 2 & 2 & 1 \\ 2 & 2 & 0 & 3 & 1 \\ 0 & 0 & 0 & 0 & 1 \end{vmatrix} = 0.$$

that becomes

$$-30x + 36y + z - 4f(x, y, z) = 0.$$

Therefore

$$f(x, y, z) = -\frac{15}{2}x + 9y + \frac{1}{4}z$$

and

$$\nabla f(\mathbf{x}) = (-\frac{15}{2}, 9, \frac{1}{4}).$$

4.2. Evaluation in lower-dimensional simplices

When the simplex is not maximal, the approach described in Section 4.1 cannot be applied "as it is" because the matrix would be rectangular. Thus, our approach is based on a projection of the simplex onto a lower-dimensional space so that it becomes maximal. In such a new space we can derive the gradient as described in Section 4.1, and the resulting vector can be transformed back to the original high-dimensional space by inverting the projection matrix.

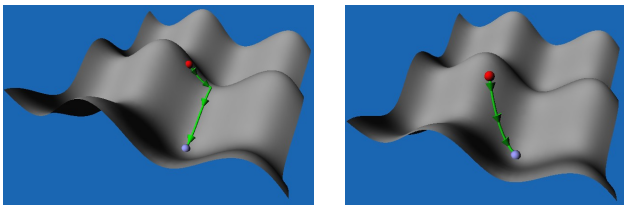
Thus, let $|\sigma|$ be defined by m vertices $\mathbf{p}^1 = (p^1_1, \ldots, p^1_n)$, ..., $\mathbf{p}^m = (p^m_1, \ldots, p^m_n)$, with $m < n + 1$. We observe that the set of $m - 1$ vectors $\mathbf{g}^1 = \mathbf{p}^2 - \mathbf{p}^1$, $\mathbf{g}^2 = \mathbf{p}^3 - \mathbf{p}^1$, ..., $\mathbf{g}^{m-1} = \mathbf{p}^m - \mathbf{p}^1$ constitutes a basis of the $(m - 1)$-dimensional Euclidean space where σ is maximal. So, we translate $|\sigma|$ so that its first vertex $\mathbf{p}^1$ coincides with the origin of $\mathbb{R}^n$. Then, we consider the $(m-1)\times(n)$ matrix G made of the $\mathbf{g}^i$s and orthonormalize it (eg. through Gram-Schmidt). By right-multiplying the orthonormal matrix $G_\perp$ by each vertex of $|\sigma|$ we obtain an $(m-1)$-dimensional Euclidean simplex in $\mathbb{R}^{m-1}$. After having derived the $(m - 1)$-dimensional gradient vector of f as described in Section 4.1, we right-multiply $G^T_\perp$ by ∇f to transform it back to the original space $\mathbb{R}^n$.

5. Implementation and results

Since the algorithm is dimension-independent, in our implementation we have employed an extended indexed data structure with adjacencies [16] enriched with function values at vertices. Namely, if the mesh is n-dimensional and made of N_V vertices and N_S maximal simplices, our data structure explicitly encodes:

- an array G of N_V n-uples of coordinates representing the vertex positions along with an array F of N_V corresponding function values;
- an array S of N_S $(n + 1)$-uples of indexes in G representing maximal simplices along with an array O of N_S $(n+1)$-uples of indexes in S representing the corresponding adjacent maximal simplices;
- an array M of N_V indexes in S representing, for each vertex, one of the incident maximal simplices.

Using this structure, we can extract all the geometrical and topological information required for the execution in optimal time. The algorithm has been implemented in C++.



(a) (b)

Figure 3: Path computed on a known function for verification. The surface is the image of $f(x, y) = sin(x) + cos(y)$. The path starts from $(0, 0)$ in (a) and from $(1, 1)$ in (b).

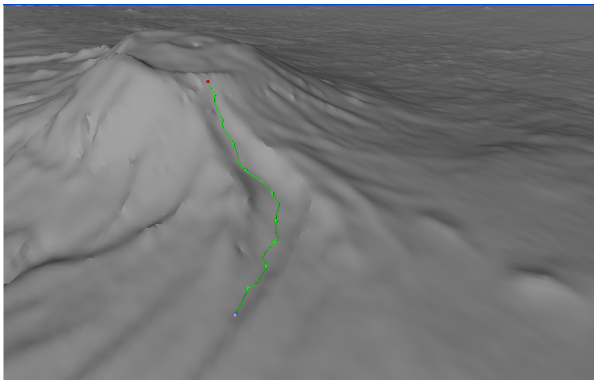


Figure 4: Steepest descent path computed on a large model of the Kilimanjaro volcano (1 million triangles, Model courtesy of Aim@Shape's repository). On volcanos, these paths can be used to predict possible lava flows.

Though we have tested our prototype on meshes of various dimensions, for $n \geq 4$ visualizing the results is

tricky and not very informative. Thus, with the exception of figure 12, we herewith show some of our results for domains of dimensions two and three only. Note that in some two-dimensional cases the value of f may represent a height (eg. Figures 4, 6 and 3), whereas in some other cases the function may have a completely different meaning (eg. a temperature, such as in Figure 5). Nevertheless, to better conceive the results through our illustrations, for any two-dimensional mesh we have used the value of f as a third coordinate, thus embedding the 2-dimensional complex in $\mathbb{R}^3$, even if f does not represent a height. In contrast, for most illustrations regarding three-dimensional domains (Figures 7, 8, 9 and 10) we have used the height (i.e. the z coordinate) to define the value of f, with the exception of Figure 11 where the function values are provided aside and represent a potential.

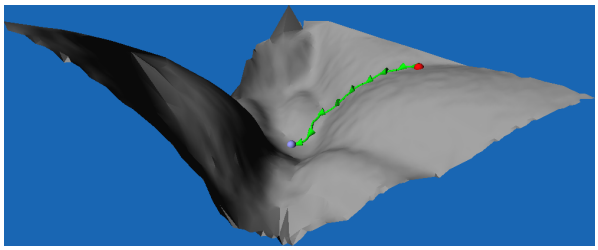


Figure 5: The so-called "liquidus surface" of a ternary chemical system. The local minimum reached by the path represents a stable state of the material.

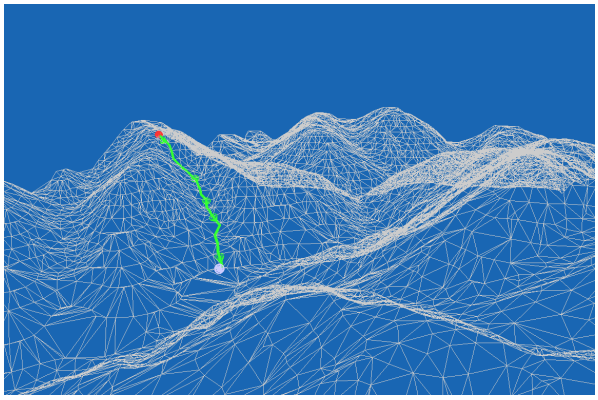
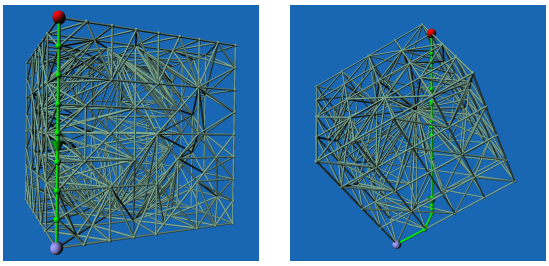


Figure 6: Steepest descent path on 2-dimensional simplicial complex representing a terrain.

5.1. Computational times

The size of the experimental data sets shown in this paper ranges from a few hundred to more than a million simplices. Experiments have been performed on a standard desktop PC equipped with a *Intel(R) Core(TM)2 6320 @ 1.87GHz* and *4096MByte RAM*. In table 1 and table 2 execution times are reported for meshes of dimension two and three respectively. To provide the reader an idea of the complexity of the input, the tables indicate both the number N_V of vertices and the number N_S of maximal simplices. However, the number of operations (and hence the execution time) performed by the algorithm depends mostly on the number of simplices traversed by the path.



(a) (b)

Figure 7: A cubical domain with an associated function $f(x, y, z) = z$. In (a) the cube is aligned with the coordinate system, and the path traverses only lower-dimensional simplices. In (b) the cube is rotated: as expected, the path descends vertically through the solid until it reaches the boundary, then it follows the steepest direction on the boundary of the domain.

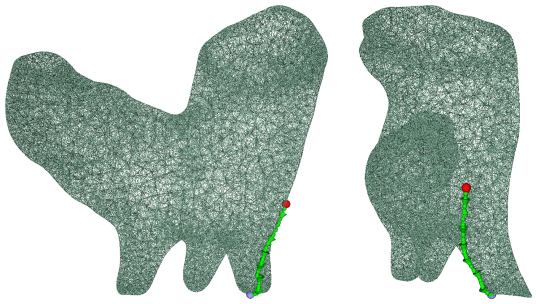


Figure 8: Two points of view of the same path computed on a tetrahedral mesh. Model courtesy of Aim@Shape's repository.

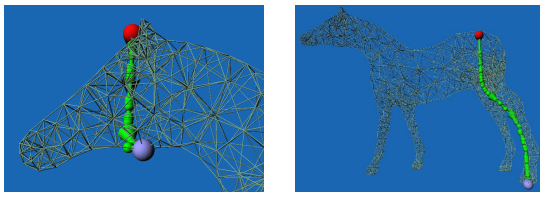


Figure 9: Paths computed starting from two different points of the same tetrahedral mesh. Model courtesy of Aim@Shape's repository.

Model	N_V	N_S	time
CAS (fig.5)	3777	7416	0.027
Quark (fig.3)	3969	7688	0.019
Vobbia (fig.6)	7745	15312	0.051
Kilimanjaro (fig.4)	490001	977204	0.78

Table 1: Execution time for the models of dimension two made of N_V vertices and N_S triangles. Time is expressed in seconds and does not include input/output operations.

Model	N_V	N_S	time
Cube (fig.7(a))	182	436	0.042
Rotated Cube (fig.7(b))	103	239	0.026
Aorta (fig.8)	9529	35551	0.219
Horse (fig.9)	837	2152	0.076
Dino (fig.10)	11979	61527	0.231
Bucky (fig.11)	262144	1250235	0.456

Table 2: Execution time for the models of dimension three made of N_V vertices and N_S tetrahedra. Time is expressed in seconds and does not include input/output operations.

5.2. Robustness, stability and accuracy

Instability is intrinsic at each point of the domain where the gradient vanishes, and small perturbations may lead to dramatically different gradient directions. In particular, if the point is a local maximum of a differentiable function, any direction in the domain is equally eligible to track a steepest-descent path. The most conservative approach would be to compute all the possible paths, but unfortunately they might be an infinite number. If f is a Morse function [17] algorithms exist to compute the so-called *descending Morse complex*; from such a structure one can easily understand which are the points that can be possibly part of a descent path starting at a local maximum. Unfortunately, these algorithms are known only for domains of dimension two and three. For this reason, for now we just ignore the problem and let the algorithm provide one of the possible solutions.

From the computational point of view, the *discretization* might be a further issue. In several cases, a smooth function f is only partially defined through a finite sampling of its domain which is linearly interpolated within a simplicial mesh. To be conservative, also in this case we may need to produce infinite solutions where the path crosses a *flat* simplex. However, as mentioned in section 3, depending on the surroundings we may interpret such a flatness as an artefact and let the algorithm provide one of the possible solutions.

It is also worth speculating about the accuracy of our algorithm when it is used to process a discretized version of a Morse function f. If x_0 is the starting point and

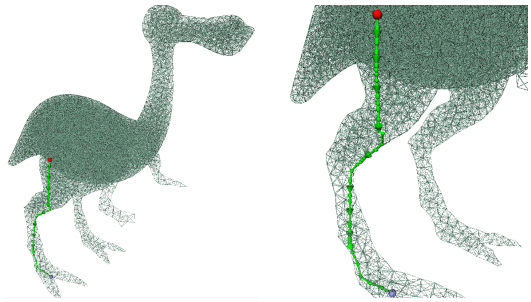


Figure 10: A path computed starting from an internal point of a dense tetrahedral mesh. As expected, the path goes vertically as long as it can, while it correctly follows the boundary when necessary. Model courtesy of Pierre Alliez.

h is the sampling step used, we may consider the following cases: if h is too large some simplices might actually *cancel* small local minima, and the resulting path might be longer than expected; if h is sufficiently small but x_0 is closer than h to the border of a maximal cell of the stable Morse complex of f (e.g. Fig 3(a)), the path may take a wrong direction and fall into a different minimum; in all the other cases the path ends within a maximum distance h from the ideal minimum.

Furthermore, when the samples of f are corrupted by noise, the function itself should be *smoothed* to prevent the path to be broken at a spurious local minimum. While advanced techniques exist for two dimensions [18, 19], the case of higher dimensions has been mostly left uninvestigated. Finally, the computation of the hyperplane of a nearly degenerate simplex is ill-conditioned, and also in this case existing work to remove such undesired degeneracies is mostly dedicated to two and three dimensions [20].

When the function is actually piecewise linear (i.e. it is not an approximation) our algorithm provides an exact result. This happens, for example, in the case of a linear program whose constraints define a convex polytope. In this case one may also use Dantzig's simplex algorithm while being guaranteed to reach the same correct minima. Differently from the simplex method, however, our algorithm is free to cut through higher-dimensional simplexes and can work on nonconvex domains. On convex domains, passing through edges only (as done by the simplex algorithm) is sufficient to reach the final exact minimum, though the path is not necessarily exact. In contrast, the same constraint on nonconvex domains might cause the path to reach completely wrong results.

In case of 2- and 3-dimensional meshes, existing methods can be applied to compute steepest paths.

These paths are sometimes called *integral curves* and they are mainly used to generate *Morse-Smale complexes*. Amongst these existing algorithms, some [7, 8] allow paths to cut through maximal simplices, as we do, and perform in similar time. In these cases our algorithm achieves exactly the same results while being more general. Conversely, other methods (e.g. [4], [5], [6]) are more efficient than ours, but only return an approximation of the steepest path since they build it as a collection of edges (1-simplices). For this reason, these methods often require an initial mesh refinement to reduce inaccuracy.

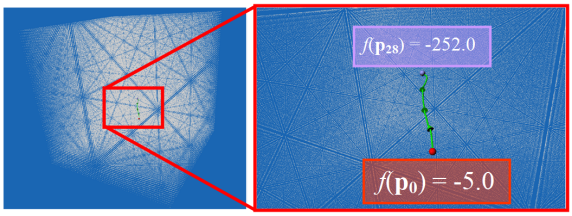


Figure 11: A complex tetrahedral mesh (1.25 million tets) discretizing a potential f around a carbon molecule with a path reaching a local minimum of f. Model courtesy of Aim@Shape's repository.

5.3. Generalization

The algorithm described in this paper has numerous practical applications, including the prediction of lava flow paths (Fig. 4) and the computation of stable states reachable by cooling molten substances (Fig. 5). In some cases, however, the knowledge of the function alone is not sufficient to accurately predict the evolution of a system, and further variables may be necessary to provide a more comprehensive and precise model of the phenomenon. For example, when trying to predict a lava flow, one should take into account that the flowing material has an inertia. Thus, to better predict the path, we should correct the steepest descent direction by using an additional inertial term. As a further example, in material sciences one can predict the result of a crystallization process by following descent lines on the so-called *liquidus* hypersurface (see Section 5.4), but such lines are not necessary locally steepest at all their points, because their direction depends on additional variables.

We may adapt our algorithm by providing a generalized approach to determine the direction to be followed at all the points. Specifically, the gradient field can be computed in advance (once for each simplex) and possibly corrected by considering additional terms. Thus, the input to the *adapted* version of the algorithm is a simplicial complex where each simplex is associated with a vector specifying the direction of movement when lying in it, and the magnitude of the vector corresponds to the *steepness* in that direction. In this setting, for each simplex σ the value of Δ_i is just the magnitude of the vector v_σ associated with σ. If the input vectors form a *nongradient* vector field [21] some of the paths may be closed loops, thus the visited simplices should be marked to avoid the algorithm to infinitely loop.

5.4. Applications in material research

Material research (ceramurgy, glass technology, slags industry), Geology (petrology, geochemistry) and engineering try to understand the characteristics of complex materials by studying the interrelationship of composition, microstructure and process conditions represented in *phase diagrams*, whose computation is one of the main objectives of computational thermodynamics. In these fields of research the absolute amount of pure components constituting a complex material is not very important, while it is much more interesting to know their *relative* amount. For example, *pseudowollastonite* is always composed of 50% CaO and 50% SiO_2 in molar proportions independently of the bulk amount of the substance. A convenient way to describe the domain of all the possible relative compositions is by using a simplex, where each vertex corresponds to one of the pure components. Thus, such a simplex is called the *compositional simplex*. For any molten substance, the temperature of incipient crystallization changes depending on the relative quantity of pure components. The function that associates the incipient crystallization temperature with each point of the compositional simplex is called the *liquidus*. Since the dimension of the simplex is arbitrary (i.e. it depends on the number of pure components) the liquidus is a hypersurface and, as such, can be represented piecewise-linearly through a simplicial complex [22].

By starting at a given point in the compositional simplex (i.e. a relative composition) it is possible to track so-called *descent lines* representing the crystallization induced by loss of heat toward the exterior or by compression. At each point, the direction of such lines is dictated by the *lowering of freezing point equation* [23]. Thus, by using this equation we may assign the correct direction to each simplex of the liquidus, and then exploit the algorithm described here to predict the composition that will be generated when the process reaches its stability point.

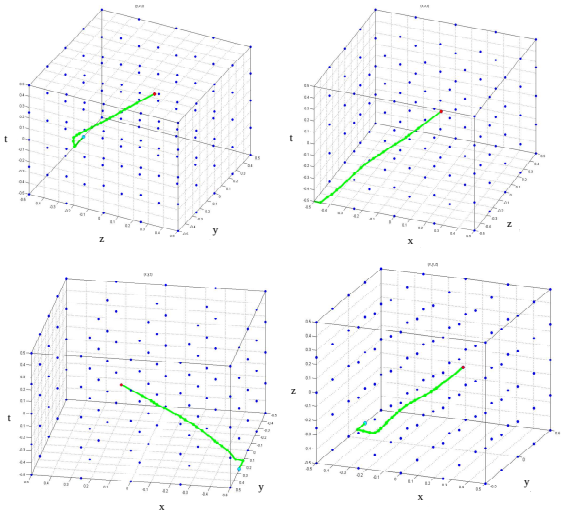


Figure 12: An example showing a path computed within a four-dimensional domain. The function is defined as $f(x, y, z, t) = (y - x^2)^2 + (x - 1) + z + t$. The domain is a unitary hypercube $[-0.5, 0.5] \times [-0.5, 0.5] \times [-0.5, 0.5] \times [-0.5, 0.5]$, and the images depict four axis-aligned 3D projections of the result. To better convey the result, edges of the simplicial complex are not shown.

6. Conclusions and future work

This research shows that it is possible to efficiently compute steepest descent paths on piecewise-linear functions defined on Euclidean domains of any dimension. In practice, such functions are defined through a finite sampling and the resulting discretization may lead to flat simplices for which we have proposed an appropriate treatment. In some cases, a vector field may be available instead of the function, and often such a field determines the path's direction at all the points; in these cases, we have shown that our algorithm can be easily adapted to compute the paths according to the input field, and have presented a practical application in the area of Material Sciences.

As it is, however, the algorithm presented here has some limitations that constitute the subject of our future research. First, when the steepness is the same in more than one direction at a given point our algorithm simply chooses one of the possibilities, whereas it would be certainly useful to have a more comprehensive result including possible branches and joins: the possibility to have even infinite alternatives (eg. when the starting point is within a flat simplex) makes this problem rather challenging. An even more challenging direction for future research is to consider higher-degree (i.e. non linear) functions to interpolate the samples, but this might require a substantially different algorithm.

Acknowledgements. This work is partly supported by the Italian MIUR-PRIN Project N. 2009B3SAFK (Topology of phase diagrams and lines of descent) and by the Petromaks program of the Norwegian Research Council through the Geoillustrator project (N. 200512). Thanks are due to the SMG members at IMATI for helpful discussions.

References

[1] H. Edelsbrunner, J. Harer, A. Zomorodian, Hierarchical morse complexes for piecewise linear 2-manifolds, in: SCG '01: Proceedings of the seventeenth annual symposium on Computational geometry, ACM, New York, NY, USA, 2001, pp. 70–79.

[2] H. Edelsbrunner, J. Harer, V. Natarajan, V. Pascucci, Morse-smale complexes for piecewise linear 3-manifolds, in: SCG '03: Proceedings of the nineteenth annual symposium on Computational geometry, ACM, New York, NY, USA, 2003, pp. 361–370.

[3] V. Natarajan, V. Pascucci, Volumetric data analysis using morse-smale complexes, in: SMI '05: Proceedings of the International Conference on Shape Modeling and Applications 2005, IEEE Computer Society, Washington, DC, USA, 2005, pp. 322–327.

[4] S. Takahashi, T. Ikeda, Y. Shinagawa, T. L. Kunii, M. Ueda, Algorithms for extracting correct critical points and constructing topological graphs from discrete geographic elevation data, Computer Graphics Forum 14 (3) (1995) 181–192.

[5] C. Bajaj, D. Schikore, Topology preserving data simplification with error bounds, Comput. Graph. 22 (1) (1998) 3–12.

[6] P. Bremer, V. Pascucci, A practical approach to two-dimensional scalar topology, in: Topology-Based Methods in Visualization, Springer, Berlin Heidelberg, Germany, 2007, pp. 151–169.

[7] P.-T. Bremer, H. Edelsbrunner, B. Hamann, V. Pascucci, A multi-resolution data structure for two-dimensional morse functions, in: VIS 03: Proceedings of the IEEE Visualization 2003, IEEE Computer Society Press, Los Alamitos, CA, 2003, pp. 139–146.

[8] V. Pascucci, Topology diagrams of scalar fields in scientific visualization, in: Topological Data Structures for Surfaces, John Wiley and Sons London, U.K., 2004, pp. 121–129.

[9] S. Dong, P.-T. Bremer, M. Garland, V. Pascucci, J. C. Hart, Spectral surface quadrangulation, ACM Trans. Graph. 25 (3) (2006) 1057–1066.

[10] X. Ni, M. Garland, J. C. Hart, Fair morse functions for extracting the topological structure of a surface mesh (2004) 613–622.

[11] J. L. Helman, L. Hesselink, Representation and display of vector field topology in fluid flow data sets, Computer 22 (8) (1989) 27–36.

[12] S. Biasotti, G. Patanè, M. Spagnuolo, B. Falcidieno, Analysis and comparison of real functions on triangulated surfaces, Curve and Surface Fitting Modern Methods in Mathematics (2007) 41–50.

[13] S. Biasotti, G. Patanè, M. Spagnuolo, B. Falcidieno, G. Barequet, Shape approximation by differential properties of scalar functions, Computers and Graphics 34 (3) (2010) 252–262.

[14] M. Attene, D. Giorgi, M. Ferri, B. Falcidieno, On converting sets of tetrahedra to combinatorial and PL manifolds, Computer-Aided Geometric Design 26 (8) (2009) 850–864.

[15] L. C. Glaser, Geometrical Combinatorial Topology, Vol. 1, Van Nostrand Reinhold, 450 West 33rd Street, New York, 1970.

[16] L. De Floriani, A. Hui, Data structures for simplicial complexes: an analysis and a comparison, in: SGP '05: Proceedings of the third Eurographics symposium on Geometry processing, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 2005.

[17] S. Biasotti, L. De Floriani, B. Falcidieno, P. Frosini, D. Giorgi, C. Landi, L. Papaleo, M. Spagnuolo, Describing shapes by geometrical-topological properties of real functions, ACM Comp. Surveys 40 (4) (2008) 12:1–12:87.

[18] H. Edelsbrunner, D. Morozov, V. Pascucci, Persistence-sensitive simplification of functions on 2-manifolds, in: Procs. of Symposium on Computational Geometry, 2006, pp. 127–134.

[19] G. Patane, B. Falcidieno, Computing smooth approximations of scalar functions with constraints, Computers and Graphics 33 (3) (2009) 399–413.

[20] M. Attene, A lightweight approach to repairing digitized polygon meshes, The Visual Computer 26 (11) (2010) 1393–1406.

[21] G. Chen, K. Mischaikow, R. S. Laramee, P. Pilarczyk, E. Zhang, Vector field editing and periodic orbit extraction using morse decomposition, IEEE Trans. on Visualization and Computer Graphics 13 (4) (2007) 769–786.

[22] M. Attene, G. Ottonello, Computational geometry meets material science, ERCIM News 84 (2011) 43–44.

[23] G. Ottonello, Principles of Geochemistry, Columbia University Press, N.Y., 1997.