Project number IST-25582

CGL

Computational Geometric Learning

"A parallel algorithm for computing the flow complex"

STREP

Information Society Technologies

Period covered:	November 1, 2012–October 31, 2013
Date of preparation:	October 31, 2013
Date of revision:	October 31, 2013
Start date of project:	November 1, 2010
Duration:	3 years
Project coordinator name:	Joachim Giesen (FSU)
Project coordinator organisation:	Friedrich-Schiller-Universität Jena Jena, Germany

A parallel algorithm for computing the flow complex

Joachim Giesen Lars Kühne
Friedrich-Schiller-Universität Jena, Germany

Abstract

We present a parallel algorithm and its implementation for computing the entire Hasse diagram of the flow complex of a point cloud in Euclidean space. Known algorithms for computing the flow complex in two and three dimensions compute the geometric realization of the flow complex and need to compute the Delaunay triangulation of the point cloud first. Our algorithm computes less information, namely only the Hasse diagram of the flow complex that is augmented with enough geometric information to allow the same topological multi-scale analysis of point cloud data as the alpha shape filtration without computing the Delaunay triangulation explicitly. We show experimental results for medium dimensions that demonstrate that our algorithm scales well with the number of available cores on a multicore architecture.

1 Introduction

The flow complex is a data structure on a finite set of points in Euclidean space $\mathbb{R}^d$. A flow complex has been introduced by Edelsbrunner [7, 8] in three dimensions for surface reconstruction with the WRAP algorithm and for applications in structural computational biology, or more specifically for finding pockets in proteins. The flow complex as defined by Edelsbrunner is always a sub-complex of the Delaunay triangulation of the point set. Here we study a variant of the flow complex that has been introduced by Giesen and John [10, 11]. Later a generalized characterization of this variant and an algorithm for computing it has been given by Buchin et al. [1]. Notably, this variant is not a subcomplex of the Delaunay triangulation anymore. In the following we will always refer to the latter variant just as the flow complex. The flow complex has been used for provably correct surface reconstruction [5] and for medial axis approximation with geometric and topological guarantees [12]. It is known that the flow complex can be used for the topological multi-scale analysis of the point cloud data, in fact the flow complex can be seen as the topologically sparsest encoding of the alpha shape filtration of the Delaunay triangulation of point cloud data [1, 4, 6]. Here topologically sparsest means that any combinatorial change during the filtration of the complex also corresponds to a topological change. This is not true for alpha shapes that can change combinatorially while keeping the homotopy type.

All known algorithms for computing the flow complex [3, 10, 11] compute the Delaunay triangulation of the point set first. So far only implementations in two and three dimensions exist. In fact, all these implementations compute the geometric realization of the flow complex which is a polyhedral complex by definition. Cazals and Cohen-Steiner [2] have pointed out that the most important information, i.e., all the information that is needed for a topological multi-scale analysis of the point cloud data, can be encoded in an augmented Hasse diagram of the flow complex. The Hasse diagram encodes the incidence structure between the cells of the flow complex, and its vertices represent critical points of the distance function to the point cloud. In an augmented Hasse diagram also the value of the distance function at the critical points is stored. Here we build on the observation by Cazals and Cohen-Steiner and devise an algorithm for computing the augmented Hasse diagram of the flow complex. Our algorithm avoids the explicit, global computation of the Delaunay triangulation or Voronoi diagram of the point set. The algorithm is essentially a graph exploration algorithm (breadth-first search) that computes implicit information about the Delaunay triangulation only locally. The algorithm borrows ideas from an algorithm by Fischer et al.[9] for computing the smallest enclosing ball of a point cloud in high dimensions. The latter algorithm uses fairly different primitives than the ones that are usually employed for computing Delaunay triangulations and Voronoi diagrams, i.e., in-circle and left-of predicates. In fact the primitives used in the algorithm are not predicates but constructions, namely finding the "nearest point" along a ray, and projecting a point onto the affine hull of a point set. It turns out that these constructions can be implemented with sufficient numerical accuracy using standard floating point arithmetic.

This paper is structured as follows. First, we provide necessary definitions and notations in Section 2. Then, in Section 3 we show how computing the combinatorial structure of the flow complex, i.e., its (augmented) Hasse diagram, can be reduced to an inherently parallel graph exploration. In Section 4 we prove the correctness of the graph exploration strategy. The exploration algorithm makes only use of the two primitive operations/constructions. In Section 5 we discuss how these primitive operations can be implemented and how to maintain the necessary data structures. In Section 6 we discuss our implementation of the algorithm and present some experimental results that show that our algorithm scales very well with the number of available cores on a multicore architecture. We conclude the paper in Section 7.

2 Flow complex

In this section we provide the necessary terminology that we will use later on to describe our parallel algorithm. Let $P \subset \mathbb{R}^d$ always be a finite point set. In order to simplify the exposition we assume that P is in general position and that P has at least $d+1$ points.

Distance function. The distance function $h : \mathbb{R}^d \to [0, \infty)$ induced by P is given as

$$h : \mathbb{R}^d \ni x \mapsto \min_{p \in P} \|x - p\|.$$

The distance function value at x is realized by the neighbors $N(x) = \{p \in P \,:\, \|x - p\| = h(x)\}$. The *driver* $d(x)$ of x is the center of the smallest enclosing ball of $N(x)$, and the gradient of the distance function at x is given as

$$\partial_h(x) = \frac{x - d(x)}{h(x)}, \quad \text{if} \quad x \neq d(x),$$

and 0 otherwise. The points $x \in \mathbb{R}^d$ with $\partial_h(x) = 0$, i.e., the points for which $x = d(x)$, are called the critical points of the distance function. Critical points can be defined equivalently by the condition $x \in \text{conv}(N(x))$, i.e., critical points are contained in the convex hull of their neighbors in P. The latter characterization can be used to assign an index $i(x)$ to a critical point x, namely the dimension of the affine hull of $N(x)$. Critical points with index 0 are the points in P, i.e., the minima of the distance function. Critical points with index d are maxima of the distance function, and all other critical points are saddle points of the distance function. With this definition the following index theorem holds [14],

$$\sum_{i=0}^{d} (-1)^i \, n_i \;=\; 1,$$

where n_i is the number of critical points of index i. Additionally, we add a critical point of index d, i.e., a maximum, at infinity. The alternating sum then always gives 1 (or 2 in even dimensions and 0 in odd dimensions if one includes a symbolic maximum at infinity).

Flow complex. The flow complex is a cell complex that consists of the *stable manifolds* of the flow induced by the gradient vector field ∂_h. The flow is a mapping

$$\phi : [0, \infty) \times \mathbb{R}^d \to \mathbb{R}^d$$

defined by the equations $\phi(0, x) = x$ and

$$\lim_{t \downarrow t_0} \frac{\phi(t, x) - \phi(t_0, x)}{t - t_0} = \partial_h(\phi(t_0, x)).$$

The set $\phi(x) = \{\phi(t, x) \,|\, t \geq 0\}$ is called the *orbit* or *flow line* of the point x. The stable manifold $S(x)$ of a critical point x is the set of all points in $\mathbb{R}^d$ that flow into x, i.e.,

$$S(x) = \{y \in \mathbb{R}^d \,:\, \lim_{t \to \infty} \phi(t, y) = x\}.$$

The flow complex is given by the unstable manifolds of all critical points together with the following *incidence information* that is defined via the *unstable manifolds* of critical points. Given a neighborhood U of a critical point x and setting

$$V(U) = \{y \in \mathbb{R}^d \,:\, \exists z \in U, t \geq 0 \text{ s.t. } \phi(t, z) = y\},$$

the unstable manifold of x is the set

$$U(x) = \bigcap_{\text{Neighborhood } U \text{ of } x} V(U).$$

The stable manifold of a critical point y is incident to the stable manifold of a critical point x if $S(x) \cap U(y) \neq \emptyset$, i.e., if there is a point in the unstable manifold of y that flows into x.

Hasse diagram. The incidence structure on the stable manifolds of the critical points is a binary relation that is

1. *reflexive* since $S(x) \cap U(x) = \{x\}$ for any critical point x.
2. *antisymmetric* since $S(x) \cap U(y) \neq \emptyset$ and $S(y) \cap U(x) \neq \emptyset$ implies $x = y$.
3. *transitive* since $S(x) \cap U(y) \neq \emptyset$ implies $U(x) \subseteq U(y)$, and hence if x is incident to z, i.e., $S(z) \cap U(x) \neq \emptyset$ then also y is incident to z.

Hence, the combinatorial structure of the flow complex induces a partial order on the set of stable manifolds which can be encoded in a Hasse diagram see Figure 2 (in the appendix). Any chain of this partial order is also ordered by the indices of the critical points. Note though that there may exist chains that do not correspond to consecutive indices and have no super chain with consecutive indices. Geometrically, this is caused by "dangling" stable manifolds for critical points of index k that are not in the boundary of any stable manifold of index $(k+1)$, see Figure 3 (in the appendix).

Maximum at infinity. We add a symbolic maximum at infinity to the flow complex. The maximum at infinity is incident to any critical point x whose unstable manifold $U(x)$ is unbounded.

Our goal is to compute only the Hasse diagram of the flow complex for a given point set P and not the geometrically realized complex. But additionally, we also compute the index and the distance function value for every critical point.
Since the flow complex is intimately linked to the Voronoi diagram and the Delaunay triangulation of the point set we also briefly restate their definitions here.

Voronoi diagram and Delaunay triangulation. The Voronoi cell $V(p)$ of a point $p \in P$ is defined as

$$V(p) = \{x \in \mathbb{R}^d \,:\, \forall q \in P \, \|x - p\| \leq \|x - q\|\}.$$

Since the Voronoi cell $V(p)$ is the intersection of a set of closed halfspaces it is convex. The non-empty intersection of k Voronoi cells is called a Voronoi $(d + 1 - k)$-facet, i.e., the Voronoi cells themselves are d-facets. The 0-facets are also called Voronoi vertices. The set of all Voronoi facets together with the incidence structure given by inclusion is called the Voronoi diagram of P. The Voronoi diagram corresponds to a subdivision of $\mathbb{R}^d$.
Since any Voronoi cell corresponds to exactly one point in P any Voronoi $(d+1-k)$-facet corresponds to k points in P. The convex hull of these k points is called the Delaunay $(k-1)$-facet dual to the Voronoi $(d+1-k)$-facet. The set of all Delaunay facets together with the incidence structure given by inclusion is called the Delaunay triangulation of P. The Delaunay triangulation corresponds to a subdivision of the convex hull of the point set P into simplices.
The connection with the flow complex is that any critical point of index k is the unique intersection point of a Voronoi $(d-k)$-facet and its dual Delaunay k-facet, if it exists. If a Voronoi $(d-k)$-facet and its dual Delaunay k-facet intersect, then we call both facets critical.

3 Algorithm—high level

The basic idea behind our algorithm is to explore (the Hasse diagram of) the Delaunay triangulation of the finite point set P in a breadth-first manner. The exploration of the Delaunay triangulation is done on the fly (i.e., the triangulation has not been computed a priori) in a geometric fashion that allows us to discover the incidence structure of the flow complex for the same point set.
Conceptually the exploration works as follows: starting from the maximum at infinity all maxima of the flow complex are enumerated. This is done be exploring the *maxima graph* whose vertices are the maxima of the flow complex and whose edges correspond to the index-$(d-1)$ critical points of the flow complex. The unstable manifolds of the index-$(d-1)$ critical points connect these critical points to maxima. Since the latter unstable manifolds are one-dimensional, they can be tracked algorithmically. The remaining critical points can be discovered by recursively computing the predecessors of a critical point. We refer to computing predecessors of a critical point as a *downflow* operation and to tracking the unstable manifold of an index-$(d-1)$ critical point as an *upflow operation*, see Figure 5 for a visualization of one step in the downflow and upflow operation, respectively (in the appendix). Before we describe these operations we introduce the representation of the maximum at infinity that we are going to use.

Representation of the maximum at infinity. To obtain this representation the point cloud P is augmented by $d+1$ points that are the vertices of a bounding simplex for P. Let $\hat{P}$ be the augmented point set. The additional $d+1$ points are by construction exactly the vertices of the convex hull of $\hat{P}$. The maximum at infinity is represented by a point for each Voronoi edge that is dual to the $d+1$ $(d-1)$-facets on the boundary of the convex hull of $\hat{P}$, i.e., there are $d+1$ representatives for the maximum at infinity. The representatives are chosen such that the ball centered at the representatives that has the vertices from $\hat{P} \setminus P$ of the corresponding facet on its boundary does not contain any point from P. Let x be a representative of the maximum of infinity and σ_x be the corresponding facet on the boundary of the convex hull of $\hat{P}$. By construction σ_x is a Delaunay $(d-1)$-facet in the Delaunay triangulation of $\hat{P}$ whose vertices are in $\hat{P} \setminus P$.

3.1 Downflow operation

In a downflow operation we compute predecessors of a critical point x in the Hasse diagram of the flow complex. To do so we are employing a breadth-first strategy, and thus our central data structure is a queue $\mathcal{Q}$.

Initialization. Let x be an index-k critical point with $k > 0$. We distinguish two cases, either (a) the point x is the critical point at infinity, or (b) it is not.

(a) The maximum at infinity has by our construction $d+1$ representatives $x_1, \ldots, x_{d+1}$ that correspond to Delaunay $(d-1)$-facets $\sigma_1, \ldots, \sigma_{d+1}$ on the boundary of the convex hull of $\hat{P}$. We initialize the queue $\mathcal{Q}$ with the $d+1$ tuples $(\sigma_j, x_j), j = 1, \ldots, d+1$.

(b) Since x is a finite critical point it is the center of the smallest enclosing ball of a Delaunay k-facet σ. The facet σ is the convex hull of $k+1$ points, namely the vertices of σ. The $(k-1)$-facets $\sigma_j, j = 1, \ldots, k+1$, incident to σ are the convex hulls of the k element subsets of the vertex set of σ, i.e., each σ_j can be obtained by "dropping" one vertex from the vertex set of σ. We initialize the queue $\mathcal{Q}$ with the $k+1$ tuples $(\sigma_j, x), j = 1, \ldots, k+1$.

Exploration. Assume that the queue $\mathcal{Q}$ is not empty, otherwise the exploration has been completed. Dequeue the tuple (σ, x) from $\mathcal{Q}$. Our exploration strategy ensures that we never enqueue

Delaunay 0-facets (vertices), i.e., the points of P, and thus σ is a Delaunay k-facet with $k > 0$. Let c be the circumcenter of σ, i.e., the center of the smallest ball that has the vertex set V of σ on its boundary. Note that c is contained in the affine hull of V. The exploration strategy is such that it ensures that the $k+1$ points in V are among the nearest points to x in P. We walk from x towards c until one the following two events occurs,

1. a point in $P \setminus V$ also becomes a nearest point to x in P (but not at the very beginning of the walk), or
2. we reach c.

In case of Event 1 let v be the additional nearest point. The convex hull of $V \cup \{v\}$ is a Delaunay $(k+1)$-facet τ, and at the end of the walk x is the circumcenter of $V \cup \{v\}$. As the circumcenter the point x is contained in the affine hull of $V \cup \{v\}$, and thus x can be written as an affine combination of the points in $V \cup \{v\}$, i.e.,

$$c = \sum_{p \in V} \lambda_p p \; + \; \lambda_v v, \qquad \text{with} \qquad \sum_{p \in V} \lambda_p \; + \; \lambda_v = 1,$$

where the index λ_v is negative because x and v are by construction on opposite sides of the affine hull of V (within the affine hull of $V \cup \{v\}$). Let $Q \subset (V \cup \{v\})$ be such that $\lambda_q < 0$ for all $q \in Q$, and let $V' = V \setminus Q$. We consider Delaunay k-facets $\sigma_j, j = 1, \ldots, k+1-|Q|$, incident to τ. These Delaunay facets are the convex hulls of the $k+1-|Q|$ element subsets of V' together with the points in Q, i.e., each σ_j can be obtained by "dropping" one vertex from the vertex set of τ that is not contained in Q. We enqueue the tuples $(\sigma_j, x), j = 1, \ldots, k+1-|Q|$, to $\mathcal{Q}$.

In case of Event 2 we distinguish two sub-cases, namely (a) either the circumcenter c of σ coincides with the center of the smallest enclosing ball of σ in which case it is contained in the convex hull of the vertex set V, or (b) it is not.

(a) Since c is contained in the convex hull of V we have found another critical point.

(b) Although c is not contained in the convex hull of V it is still contained in the affine hull of V. Hence, c can be written as an affine combination of the points in V, i.e., $c = \sum_{p \in V} \lambda_p p$ with $\sum_{p \in V} \lambda_p = 1$, where not all coefficients λ_p are non-negative. Let $Q \subset V$ be such that $\lambda_q < 0$ for all $q \in Q$, and let $V' = V \setminus Q$. We consider Delaunay $(k-1)$-facets $\sigma_j, j = 1, \ldots, k+1-|Q|$, incident to σ. These Delaunay facets are the convex hulls of the $k-|Q|$ element subsets of V' together with the points in Q, i.e., each σ_j can be obtained by "dropping" one vertex from the vertex set of σ that is not contained Q. We enqueue the tuples $(\sigma_j, c), j = 1, \ldots, k-|Q|+1$, to $\mathcal{Q}$.

Note that it is exactly this case that allows us to find *dangling* critical points, i.e., critical points that are not incident to any critical point whose index is just one larger. The index gap is reflected in a dimension gap in the exploration as we move from enqueueing k-facets to enqueueing $(k-1)$-facets.

3.2 Upflow operation

The implementation of the upflow operation is similar to the implementation of the downflow operation. A crucial difference though is that it only applies to index-$(d-1)$ critical points since they have one-dimensional unstable manifolds that can be tracked. The upflow operation tracks

exactly one branch of the unstable manifold of a given index-$(d-1)$ critical point, see [12]. We use the upflow operation to find a maximum that is incident to the given index-$(d-1)$ critical point.

We initialize the upflow operation with a tuple (σ, x), where σ is a Delaunay $(d-1)$-critical facet and x is a non-critical point in the Voronoi edge dual to σ, i.e., x is not the intersection of σ and its dual Voronoi edge.
Assume that we are processing the tuple (σ, x). Let V be the vertex set of σ, and let c be its circumcenter. The exploration strategy is such that it ensures that the points in V are among the nearest points to x in P. We walk from x in the direction $x-c$ until a point $v \in P \setminus V$ also becomes a nearest point to x in P. Let $\hat{V} = V \cup \{v\}$ and τ be the convex hull of $\hat{V}$, i.e., τ is a Delaunay facet. We distinguish two cases, namely (a) either x is contained in τ, or (b) it is not.

(a) In this case x is a critical point, more specifically a maximum (see [12]), and the upflow operation is finished.

(b) In this case x is contained in the affine hull but not in the convex hull of $\hat{V}$. Hence, x can be written as an affine combination of the points in $\hat{V}$, i.e., $x = \sum_{p \in \hat{V}} \lambda_p p$ with $\sum_{p \in \hat{V}} \lambda_p = 1$, where not all coefficients λ_p are non-negative. Let $Q \subset \hat{V}$ be such that $\lambda_q < 0$ for all $q \in Q$. We distinguish two subcases , namely (i) either the convex hull of $\hat{V} \setminus Q$ is a Delaunay $(d-1)$-facet on the boundary of the convex hull of $\hat{P}$, or (ii) it is not.

   (i) In this case we have reached a representative of the maximum at infinity, and the upflow operation is finished.

   (ii) Let σ' be the convex hull of $\hat{V} \setminus Q$, i.e., σ' is a Delaunay facet incident to τ. We continue the upflow operation with the tuple (σ', x)

3.3 Putting things together

With the downflow and upflow operations we have everything at hand to describe our algorithm. The algorithm maintains two data structures, a task queue $\mathcal{Q}$ and a list $\mathcal{L}$ of critical points that have been found already.

A task is a triple $(\sigma, x, \mathit{label})$, where σ is a Delaunay facet from the Delaunay triangulation of $\hat{P}$, x is a point from the Voronoi facet that is dual to σ and is not contained in the convex hull of σ, and *label* is either *down* if it is a downflow task, or *up* it is an upflow task. The task queue $\mathcal{Q}$ generalizes and replaces the queues used in the downflow operations.

Initialization. The task queue $\mathcal{Q}$ is initialized with a downflow task at the maximum at infinity (see Section 3.1, Case (b) of the initialization), i.e., it is initialized with

$$(\sigma_1, x_1, \mathit{down}), \ldots, (\sigma_{d+1}, x_{d+1}, \mathit{down})$$

where $x_1, \ldots, x_{d+1}$ are the $d+1$ representatives of the maximum at infinity that correspond to Delaunay $(d-1)$-facets $\sigma_1, \ldots, \sigma_{d+1}$ on the boundary of the convex hull of $\hat{P}$.

Exploration. Assume that the queue $\mathcal{Q}$ is not empty, otherwise the computation of the flow complex has been completed. Dequeue the triple $(\sigma, x, \mathit{label})$ from $\mathcal{Q}$. Depending on *label* either start a downflow or an upflow task. The tasks are basically handled as described in the downflow (Section 3.1) and upflow (Section 3.2) operations with some small modifications that we describe here. The modifications in the exploration step of the downflow operation are:

1. Whenever a tuple (σ, x) gets enqueued, then the task $(\sigma, x, \mathit{down})$ is enqueued to the task queue $\mathcal{Q}$ instead.

2. Any critical point x that is discovered during this step is added to the list $\mathcal{L}$ if it is not already stored there. Let σ be the critical Delaunay facet associated with x.

   (a) If x is an index-$(d-1)$ critical point, then the task $(\sigma, x + (x - x')/2, \mathit{up})$ is enqueued to the task queue $\mathcal{Q}$. Here, x' is the position from where we started the direct walk towards x, see Section 3.1. The point $x + (x - x')/2$ is contained in the Voronoi edge that is dual to the Delaunay facet σ, and x' and $x + (x - x')/2$ are on opposite sides of the hyperplane that is given as the affine hull of σ.

   (b) If x has not been stored in $\mathcal{L}$ already, then all the tasks $(\sigma_j, x, \mathit{down})$ are enqueued to the task queue $\mathcal{Q}$, where σ_j are the faces of σ.

The only modification in the exploration step of the upflow operation is: whenever a new tuple (σ, x) has to be processed, then the task (σ, x, up) is enqueued to the task queue $\mathcal{Q}$ instead.

Cleansing. A downflow operation may also discover critical points that are predecessors of the initializer x of the operation but are not direct predecessors, i.e., it can happen that a discovered predecessor y is also a predecessor of another predecessor of x, see Figure 4 (in the appendix). Hence, the edge computed by the downflow operation that connects y to x needs to be removed. We refer to removing these spurious edges as cleansing of the *computed* Hasse diagram.

3.4 Parallelization

Since the tasks in Section 3.3 are independent of each other they can be scheduled in parallel. The only shared resources are reading access to the point set P, and read/write access to the list $\mathcal{L}$ of critical points that have been found already.

4 Correctness

The key property that we need to prove is that (a) all predecessors of a critical point x that are computed in a downflow operation are actually predecessors of x in the Hasse diagram of the flow complex, and (b) that all direct predecessors of x in the Hasse diagram are found by the downflow operation. With this property all critical points and their incidences are found by our algorithm by starting the exploration from the maxima of the flow complex. The maxima are enumerated using upflow operations that track the unstable manifolds of index-$(d-1)$ critical points that connect these points to maxima of the flow (and hence connect the maxima of flow indirectly through index-$(d-1)$ critical points).

In the following we are going to analyze the downflow operation which traverses a tree whose root is the index-k critical point that initializes the operation. By definition of the downflow operation the leaves of the tree are critical points that have a strictly smaller index than the initializer. The edges of the tree are line segments of the form xx', where x' is enqueued to $\mathcal{Q}$ while processing x that has been dequeued before. Actually, x and x' are enqueued together with Delaunay simplices σ and σ', respectively, and we have the following observation.

Observation 1. *The vertices V_x of σ are among the nearest neighbors of x, and the union $V_x \cup V_{x'}$, where $V_{x'}$ is the vertex set of σ', contains only nearest neighbors of x'. Hence, x and x' are both contained in the Voronoi facet that is dual to σ.*

Let $x_n, \ldots, x_1$ be a path in the tree from the root node x_n to a leaf x_1, i.e., in the downflow operation we are always walking from x_{i+1} towards x_i. We need to show that x_1 is a predecessor of x_n in the Hasse diagram of the flow complex. Let $\sigma_i = \sigma_{x_i}$ and $V_i = V_{x_i}$. Consider an edge $x_{i+1}x_i$. There are two cases:

1. The projection of x_i onto the affine hull of V_{i+1} is the center c of the smallest enclosing ball of V_{i+1}, i.e., we are walking from x_{i+1} towards c. The center c is contained in σ_{i+1} and is the driver of x_i, i.e., x_i flows into x_{i+1} under the gradient flow.

2. The projection of x_i onto the affine hull of V_{i+1} is not the center of smallest enclosing ball of V_{i+1}. Hence, x_i does not flow into x_{i+1} under the gradient flow.

If there is no edge that falls under Case 2, then we are done since x_1 flows through all the x_i into x_n under the gradient flow. Assume now that there is an edge $x_{i+1}x_i$ that falls under Case 2. Our goal now is to eliminate x_i from the path. Note that the edge x_2x_1 does by the definition of the downflow operation always fall under Case 1, and thus it holds $i > 1$. Assume that i is the smallest index such that the edge $x_{i+1}x_i$ falls under Case 2, and consider the three points x_{i+1}, x_i and x_{i-1}. By the minimality assumption x_{i-1} flows into x_i under the gradient flow, i.e., the edge x_ix_{i-1} falls under Case 1. Since the projection of x_i onto the affine hull of V_{i+1} is not the driver of x_i, the driver of x_i must be the same as the driver that drives x_{i-1} into x_i (or connects x_{i-1} to x_i in the case that $x_{x-1} = x_1$). Note that all points in the interior of a Voronoi facet have the same driver, see [10]. That is, the driver of x_i is the driver for all points in the Voronoi facet that is dual to σ_{i+1}. Since this Voronoi facet contains the edge $x_{i+1}x_i$, see Observation 1, there are flow lines connecting x_{i-1} to all points on this edge. Thus there is a flow line that connects x_{i-1} to x_{i+1}. It follows that we can eliminate x_i from the path. If we do this iteratively for the edge that falls under Case 2 with smallest index among these edges, then we end up with a sequence of points that connect x_1 to x_n and are traversed under the gradient flow. Hence, x_1 is a predecessor of x_n in the Hasse diagram of the flow complex. We summarize this in the following lemma.

Lemma 1. *All predecessors of a critical point x that are computed in a downflow operation that has been initialized with x are predecessors of x in the Hasse diagram of the flow complex.* □

It remains to show that if a critical point y is a direct predecessor of the critical point x in the Hasse diagram of the flow complex, then the downflow operation initialized with x will discover y. A direct predecessor y of x in the Hasse diagram is connected to x by a flow line. Note that not every critical point that is connected to x by a flow line is a direct predecessor of x, namely if this point is also connected to critical point of higher index that is also connected to x by a flow line. Still, this does not imply that the direct predecessors of x have an index one less than the index of x. It also does not imply that all predecessors that are discovered by the downflow operation are direct predecessors in the Hasse diagram though they are—as we have seen (Lemma 1)—connected to x by a flow line.

Assume now that y is a direct predecessor of x, i.e., we have $S(x) \cap U(y) \neq \emptyset$. Let $y = x_1, \ldots, x_x = x$ be the vertex sequence of a flow line in $S(x) \cap U(y)$ that connects y to x. Remember that flow lines are always piecewise linear and every line segment x_ix_{i+1} is contained in a Voronoi facet V_i, see [10]. Let σ_i be the dual Delaunay facet of V_i. If the driver of the Voronoi facet V_i is contained in the line through the segment x_ix_{i+1} for all segments $i = 1, \ldots, n-1$, then the segments are traversed (in the opposite direction) by the downflow operation that has been initialized with $x = x_n$. Otherwise there exists a line segment x_ix_{i+1} such that the driver of V_i is not contained in the line through x_ix_{i+1}. Let x_ix_{i+1} be the line segment with this property that has the smallest index i. By

construction it must hold $i > 1$ since the line segment that connects x_1 to x_2 is driven by the critical point $y = x_1$ itself. Now, let c be the projection of x_i onto the affine hull of the vertex set of σ_i, and let $\hat{x}$ be the first point on the ray shooting from c to x_i that is contained in V_i. Note that it is possible that $\hat{x} = c$, for example when y is a *dangling* critical point. Replace the segments $x_{i-1}x_i$ and x_ix_{i+1} by the three segments $x_{i-1}\hat{x}$, $\hat{x}x_i$ and x_ix_{i+1}. The latter segments can be traversed (in the opposite direction) by the downflow operation. By repeating this construction as long as there are segments left that cannot be traversed by the downflow operation we can transform any flow line that connects y to x into a sequence of line segments that can be traversed by the downflow operation. Hence, y will be discovered by the downflow operation initialized with x. We summarize this in the following lemma.

Lemma 2. *All direct predecessors of a critical point x in the Hasse diagram of the flow complex are found by the downflow operation that has been initialized with x.* □

The correctness of the algorithm in Section 3.3 follows from Lemmas 1 and 2.

5 Algorithm—low level

The algorithm that has been described in the previous section makes use of only two primitive operations/constructions, namely finding a nearest point along a ray and projecting a point orthogonally onto an affine hull. See the appendix for a description of both operations.

6 Experiments

We have implemented our algorithm for computing the flow complex in C++ using the parallelization library Intel TBB [13]. We ran our experiments in a shared-memory environment on a 32-core AMD Opteron 6128 system with 256 GB of memory. Our implementation turned out to be memory efficient, i.e., in our experiments we never used more than 1% of the available memory.
We examined both the scalability of our algorithm with respect to the number of available cores as well as the correctness of the result in terms of the alternating sum formula (whose fulfillment of course is only a necessary condition). Table 6 shows for a selection of data sets both the distribution of critical points with respect to their indices as well as the time taken to compute the flow complex. One can verify the correctness of the alternating sum formula that, as it is required, always gives 1.
In Figure 1 we show the speedup of the computation with an increasing number of utilized cores. This experiment was run on datasets of randomly chosen points in a sphere in dimensions 4 and 5, respectively. As can be seen, the algorithm scales well in this shared-memory scenario. With an increasing number of cores the communication on the memory bus, that is necessary for insertion of critical points as well as for checking duplicate computation, becomes the bottleneck. The overall parallel overhead reaches its maximum of about 10% when all 32 cores are used.

7 Conclusions and future work

We have presented a new algorithm for computing the flow complex. The algorithm differs from known algorithms by avoiding an explicit computation of the Delaunay triangulation of the input point set. It is also inherently parallel. We implemented the algorithm that scales very well

data set	no. samples	dimension	distribution	no. cores	time (s)
random within sphere	50	3	50-137-122-34	32	0.03
random within sphere	50	4	50-199-284-178-44	32	2.66
curve	50	3	50-73-56-32	32	0.7
spiral-3d	30	3	30-64-61-26	32	0.03
spiral-3d	30	4	30-41-12-0-0	32	0.97
spiral-3d	30	5	30-41-12-0-0-0	32	182.4
beethoven-3d-model	500	3	500-1076-649-72	32	11.81
cup-3d-model	300	3	300-766-599-132	32	0.92

Table 1: This table shows the distribution of the number of critical points over their index for various data sets. The notation $a - b - c$ in the distribution column needs to be interpreted as a critical points of index 0, b critical points of index 1, and c critical points of index 2.

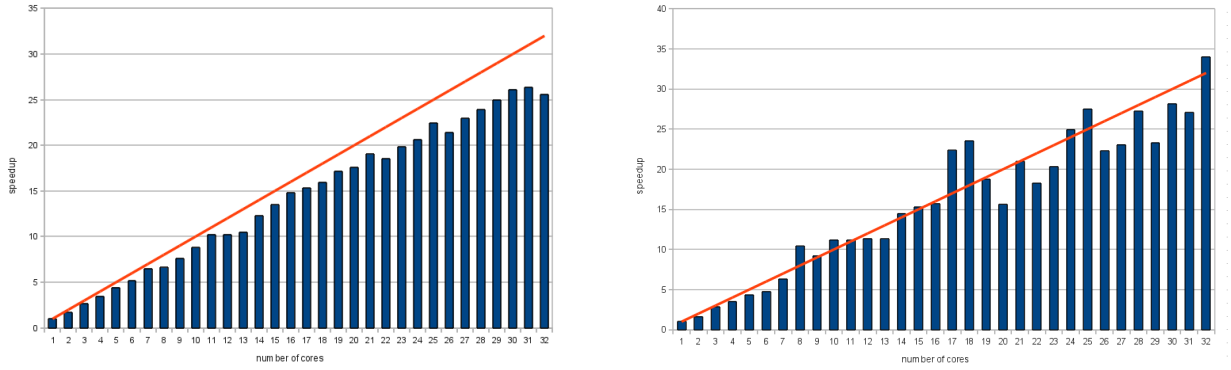

Figure 1: A plot of the speedup achieved using a growing number of cores. The red line illustrates the ideal linear speedup. On the left: computed on 200 random points in 4 dimensions. On the right: computed on 100 random points in 5 dimensions. The super-linear speedup that can be observed sometimes on the right is caused by the change of scheduling the tasks when more cores are used.

with the number of available cores on a multicore platform. Experimental results that have been obtained with the implementation show that numerical robustness is not a critical issue although the algorithm makes use of explicit constructions and does not only evaluate predicates like most algorithms for computing Delaunay triangulations.

A promising aspect of our algorithm is that it allows parallelism on an even higher level. Given a threshold parameter δ. The input point cloud can spatially subdivided into n slightly overlapping partitions where the overlap depends on δ. For these partitions all critical points whose distance function value is smaller than the threshold can computed individually. Also all critical points whose distance function value is larger than the threshold can be computed for the whole point set. Finally, the $n + 1$ result sets can be combined into the whole flow complex. In future work we want to explore the theoretical correctness and practical feasibility of this approach.

References

[1] Kevin Buchin, Tamal K. Dey, Joachim Giesen, and Matthias John. Recursive geometry of the flow complex and topology of the flow complex filtration. *Computational Geometry: Theory and Applications*, 40(2):115–137, 2008.

[2] Frédéric Cazals and David Cohen-Steiner. Reconstructing 3d compact sets. *Computational Geometry: Theory and Applications*, 45(1-2):1–13, 2012.

[3] Frédéric Cazals, Aditya G. Parameswaran, and Sylvain Pion. Robust construction of the three-dimensional flow complex. In Monique Teillaud, editor, *Symposium on Computational Geometry*, pages 182–191. ACM, 2008.

[4] Tamal K. Dey, Joachim Giesen, and Matthias John. Alpha-shapes and flow shapes are homotopy equivalent. In Lawrence L. Larmore and Michel X. Goemans, editors, *Proceedings of the 35th Annual ACM Symposium on Theory of Computing (STOC)*, pages 493–502. ACM, 2003.

[5] Tamal K. Dey, Joachim Giesen, Edgar A. Ramos, and Bardia Sadri. Critical Points of Distance to an epsilon-Sampling of a Surface and Flow-Complex-Based Surface Reconstruction. *International Journal on Computational Geometry and Applications*, 18(1/2):29–61, 2008.

[6] Herbert Edelsbrunner. The union of balls and its dual shape. *Discrete & Computational Geometry*, 13:415–440, 1995.

[7] Herbert Edelsbrunner. Surface Reconstruction by Wrapping Finite Sets in Space. In *The Goodman-Pollack Festschrift (Algorithms and Combinatorics)*, pages 379–404. Springer, 2003.

[8] Herbert Edelsbrunner, Michael Facello, and Jie Liang. On the Definition and the Construction of Pockets in Macromolecules. *Discrete Applied Mathematics*, 88:83–102, 1998.

[9] Kaspar Fischer, Bernd Gärtner, and Martin Kutz. Fast Smallest-Enclosing-Ball Computation in High Dimensions. In Giuseppe Di Battista and Uri Zwick, editors, *Annual European Symposium on Algorithms (ESA)*, volume 2832 of *Lecture Notes in Computer Science*, pages 630–641. Springer, 2003.

[10] Joachim Giesen and Matthias John. Computing the weighted flow complex. In Thomas Ertl, editor, *Vision, Modeling, and Visualization*, pages 235–243, 2003.

[11] Joachim Giesen and Matthias John. The flow complex: A data structure for geometric modeling. *Computational Geometry: Theory and Applications*, 39(3):178–190, 2008.

[12] Joachim Giesen, Edgar A. Ramos, and Bardia Sadri. Medial Axis Approximation and Unstable Flow Complex. *International Journal on Computational Geometry and Applications*, 18(6):533–565, 2008.

[13] http://threadingbuildingblocks.org.

[14] Dirk Siersma. Voronoi diagrams and morse theory of the distance function. In *Geometry in Present Day Science*, pages 187–208. World Scientific, 1999.

Appendix—low level constructions

Nearest point along a ray. The walks that have been described in Section 3 are piecewise linear and every linear piece is of the following form: given a point x whose nearest neighbors in P include the set $V \subset P$, where x is not contained in the affine hull of V. Let c be the projection of x onto the affine hull of V. Then we either walk from x towards c (as in Section 3.1 where the distance function value is decreasing along the walk) or away from c (as in Section 3.2 where the distance function value is increasing along the walk), i.e., we either walk from x into the direction $c - x$, or into the direction $x - c$. We stop walking when we reach c in the case that walk towards c, and in both cases we stop once another point from $P \setminus V$ becomes an additional nearest neighbor of x. Assume now that we are walking from x in direction v, where v is either $c - x$ or $x - c$. Let p be some point from V. If we compute t_q for any $q \in P \setminus V$ as the solution of the following equation,

$$\|(x + t_q v) - q\|^2 = \|(x + t_q v) - p\|^2 \qquad \text{which solves to} \qquad t_q = \frac{\|p\|^2 - \|q\|^2 - 2\langle x, p - q\rangle}{2\langle v, p - q\rangle},$$

then the nearest point q in $P \setminus V$ along the ray given by x and v is the one for which $t_q > 0$ is minimal. This nearest point stops our walk from x into direction v if we have not reached c before.

Projection onto an affine hull. The walks from Section 3 repeatedly need to project a point x onto the affine hull of a point set $V \subset P$. For computing such a projection we follow the strategy that has been introduced by Fischer et al. [9] for computing the smallest enclosing ball of a point set in high dimensions. At the heart of this strategy is a dynamic QR-decomposition that allows insertion into and deletion from V and supports computing the affine coefficients with respect to V of the projection of x in the affine hull of V. In [9] it has been shown that the orthogonal projections and affine coefficients can be computed in quadratic time in the dimension d by employing a QR-decomposition.
The nice thing about the strategy in [9] is that the QR-decomposition does not have to be computed from scratch every time along a piecewise linear path (like in the walks in Sections 3.1 and 3.2) but can be updated incrementally along the path. Setting up the initial QR-decomposition takes time cubic in the dimension d, but the incremental updates can be implemented using Givens rotations such that they need only quadratic time for every update. Note, that the updates that we consider here are either adding a point to or removing a point from V, see Section 3.
Fischer et al. [9] have already observed that the QR-decomposition behaves nicely with respect to numerical stability using standard floating point arithmetic when all updates on the factors Q and R are implemented with orthogonal Givens rotations.

Appendix—Figures

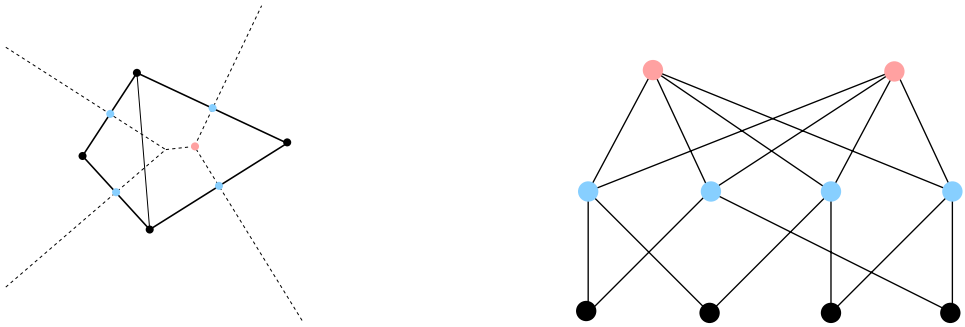

Figure 2: On the left: An example of a flow complex in two dimensions. Input are four points. Shown is the Delaunay triangulation of the points, their Voronoi diagram and its critical points. The flow complex has four index-0 critical points, four index-1 critical points, and two index-2 critical points (one of them is the maximum at infinity). On the right: The Hasse diagram of the flow complex.

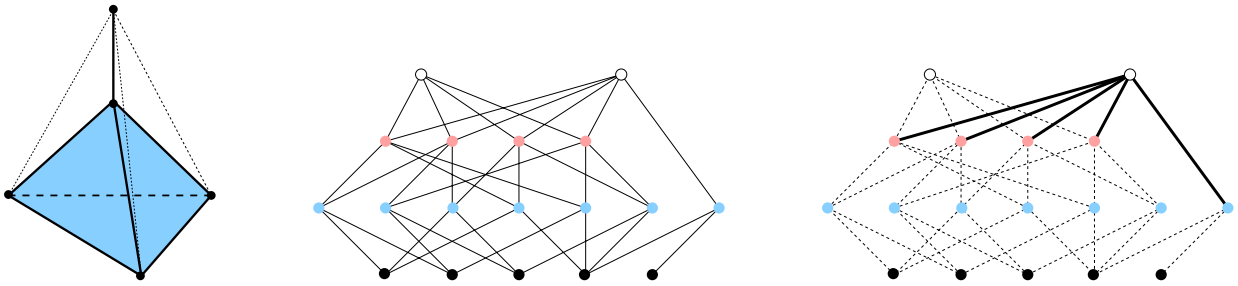

Figure 3: On the left: An example of a flow complex in three dimensions. Input are five points. Shown is the Delaunay triangulation of the points and its critical facets (bold). The flow complex has seven critical Delaunay 1-facets (one of them is dangling), four critical Delaunay 2-facets, and two critical Delaunay 3-facets (one finite maximum and the maximum at infinity). In the middle: The Hasse diagram of the flow complex on the left. On the right: The direct predecessors of the maximum at infinity.

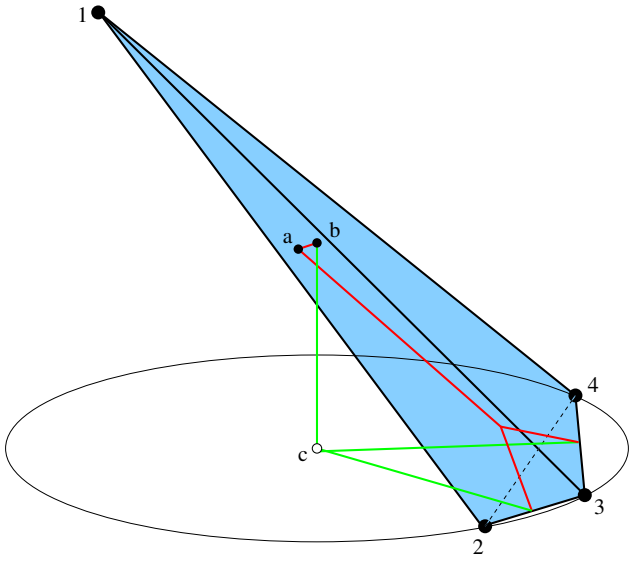

Figure 4: Shown is a point set in $\mathbb{R}^3$ with four points. The critical Delaunay facets (in terms of their vertices) are $1, 2, 3, 4, 12, 13, 14, 12, 34, 123, 124, 134, 1234$. The critical point corresponding to 124 is a, and the critical point corresponding to 1234 is b which is the circumcenter of $\{1, 2, 3, 4\}$. Note that a is a predecessor of b. The downflow operation initialized with a finds the critical points corresponding to 23 and 34 (red segments). These points are also found when the downflow operation is initialized with the maximum b (green segments) although they are not direct predecessors of b in the Hasse diagram of the flow. Note that c is the circumcenter of $\{2, 3, 4\}$ and the projection of b onto the affine hull of this set.

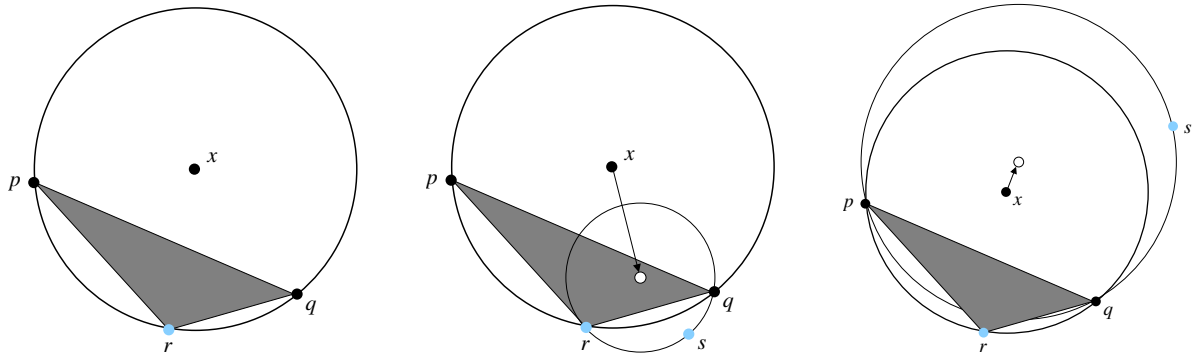

Figure 5: On the left: The point x can be expressed as an affine combination of the points p, q and r. The coefficient λ_r of r in this combination is negative. The other two coefficients λ_p and λ_q are positive. In the middle: The point p (with positive coefficient λ_p) has been dropped, and we have been walking from x towards the center of the smallest enclosing ball of $\{r, q\}$ until s also becomes a nearest neighbor of x. This operation is employed in the downflow operation in Section 3.1. On the right: The point r (with negative coefficient λ_r) has been dropped, and we have been walking starting at x away from the center of the smallest enclosing ball of $\{p, q\}$ until s also becomes a nearest neighbor of x. This operation is employed in the upflow operation in Section 3.2 for finding the maxima of the flow complex.