Online Geometric Reconstruction

BERNARD CHAZELLE, Princeton University
C. SESHADHRI, Sandia National Labs, Livermore

We investigate a new class of geometric problems based on the idea of online error correction. Suppose one is given access to a large geometric dataset though a query mechanism; for example, the dataset could be a terrain and a query might ask for the coordinates of a particular vertex or for the edges incident to it. Suppose, in addition, that the dataset satisfies some known structural property $\mathcal{P}$ (for example, monotonicity or convexity) but that, because of errors and noise, the queries occasionally provide answers that violate $\mathcal{P}$. Can one design a filter that modifies the query's answers so that (i) the output satisfies $\mathcal{P}$; (ii) the amount of data modification is minimized? We provide upper and lower bounds on the complexity of online reconstruction for convexity in 2D and 3D.

1. INTRODUCTION

Classical error correction assumes the prior availability of exact data. In this way, we can encode the data in redundant form so as to allow its recovery after transmission through a noisy channel. But what if the data $\mathcal{D}$ comes to us already corrupted? If the clean data is out of reach, then obviously the very notion of corruption requires an assumption about prior state. Indeed, what justification would one have to call a signal noisy if we have no idea what clean data might look like. The prior could be a distribution or, more generally, a property $\mathcal{P}$. For example, the data could be the facial representation of a cell decomposition and $\mathcal{P}$ could specify that it be a Voronoi diagram. Or $\mathcal{D}$ could be a terrain that $\mathcal{P}$ constrains to be monotone or convex. Or $\mathcal{D}$ might consist of an architectural design, with $\mathcal{P}$ enforcing certain angular constraints
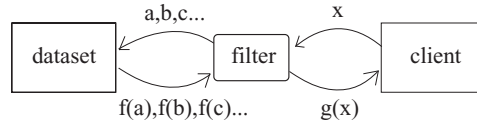
Fig. 1.   The online filter responds to a query x.

(for example, right angles between adjacent walls). Or $\mathcal{D}$ could be a cloud of points in high dimensions that $\mathcal{P}$ forces on or near a low-degree algebraic manifold.

In all cases, the same question arises: Is it possible to filter the data online so that (i) it satisfies $\mathcal{P}$ and (ii) the amount of modification is minimized? Such a filter can be used as the front-end to geometric codes whose correctness depends on certain properties (such as convexity, monotonicity, axis-parallelism). The offline version of the problem is a form of generalized regression: among all datasets satisfying $\mathcal{P}$, find the one nearest $\mathcal{D}$. The problem assumes a metric between datasets, which could be geometric, combinatorial, or a mixture of both. In this article, it will be purely combinatorial. The true novelty of our setting is its online nature. To see why being online changes everything, think of an architectural scenario where objects are required to be isothetic. Suppose that the first query reveals the position of a door and its adjacent wall. If the door is ajar, the online filter has two equally valid options in order to enforce isotheticity: either it can move the door or it can move the wall. The latter option would have dire consequences, however, likely to lead to the modification of the entire structure. This simplistic example points to the main challenge of online filtering: early decisions are crucial.

We use the filtering model introduced in Ailon et al. [2004b]. Access to the dataset is provided by means of an oracle f. The client specifies a query x and the oracle returns $f(x)$. The query could specify the index x of a wall, with $f(x)$ providing the coordinates of its vertices; or the index of an adjacent wall. One should not think of f as a single function but rather as the set of methods (in the OOP sense) available to access the dataset and its underlying data structures. The *filter* works like this: given a query x, instead of simply returning $f(x)$, the filter provides the client with the cleaned-up answer $g(x)$. Figure 1 illustrates its inner workings. Upon receiving the query x, the filter spawns auxiliary queries $a, b, c, \ldots$ and computes $g(x)$ on the basis of of $f(a)$, $f(b)$, $f(c)$, $\ldots$ The filter may go through several rounds before producing $g(x)$ and adaptively produce queries based on the previous answers. The set of all possible values $g(x)$ specifies a dataset $\mathcal{D}^f$ that satisfies property $\mathcal{P}$. The filter's decisions are *irreversible*: once a feature of $\mathcal{D}^f$ has been established, it can never be undone.

The quality of a filter is determined by how close $\mathcal{D}^f$ is to the object that satisfies $\mathcal{P}$ and differs from $\mathcal{D}$ as little as possible. This requires a metric Δ between datasets (defined on a case-by-case basis). For the purpose of this article, we require that all filters obey the *constant-distortion* condition:

$$\Delta(\mathcal{D}^f, \mathcal{D}) = O(\min\{\Delta(\mathcal{D}, \mathcal{D}') \mid \text{all } \mathcal{D}' \text{ satisfying } \mathcal{P}\}).$$

Obviously, the true quality of a filter will also depend on how big the big-Oh constant factor is. In other cases, for the sake of speed, one might allow nonconstant distortions (but we won't in this article). Note that, most often, setting the constant to 1 makes the reconstruction NP-hard. There are two aspects to a filter's complexity which we might sometimes want to distinguish: the *lookup complexity* is the number of dataset lookups (that is, $a, b, c, \ldots$); the *processing complexity*, which cannot be smaller, is the running time of the filter per query. The distinction is useful in instances where dataset access is slow or expensive (as is often the case in biology, medicine, geology, etc.). In this article, however, all query times refer to the processing complexity. Our emphasis also is on worst-case query times. Obviously, it is also interesting to keep the amortized

complexity low—and we'll do that—but keep in mind that to focus only on amortized complexity would render the online problem meaningless (since we could always solve the offline version and charge the first query for it). Worst-case complexity is, indeed, an *essential* feature of online reconstruction.

1.1. Our Results and Previous Work

In the following, n denotes the size of the input and the distance between two polygons (terrains) is defined as the minimum number of edges (faces) whose coordinates need to be modified to transform one into the other. All the algorithms below are randomized and succeed with high probability.[1] We give three filters for reconstructing convexity in two and three dimensions.

(1) An optimal $\widetilde{O}(\log^3 n)$ time filter for reconstructing the convexity of a simple polygon is presented as a balanced binary tree.

(2) An optimal $\widetilde{O}(\sqrt{n})$ time filter for reconstructing the convexity of a simple polygon is presented as a doubly linked list. We assume that each vertex is labeled with its rank in the list. We also give an $\widetilde{O}(n^{1/3})$ time ε-tester[2] and prove an $\Omega(n^{1/3})$ lower bound.

(3) A filter for reconstructing the convexity of a bounded aspect ratio[3] terrain is presented in triangulated DCEL format with a worst case query time of $O(n^{12/13+\delta} + \varepsilon_{\mathcal{D}}^{-O(1)})$ and an amortized time of $O(n^\delta)$. Here, δ is an arbitrarily small positive constant and $\varepsilon_{\mathcal{D}}n$ is the terrain's distance to convexity, that is, the minimum number of faces whose coordinates need to be modified in order to make the terrain convex. For the harder problem of reconstructing convexity of *arbitrary* terrains, we prove a lower bound that depends on $\varepsilon_{\mathcal{D}}$. This shows a separation between 2 and 3 dimensions.

Our first result should be thought of as a warmup. Indeed, it follows fairly directly from procedures given in Ailon et al. [2004a, 2004b]. The remaining results use completely new techniques. To get a taste of why the results are surprising, consider the fact that exact 3D reconstruction offline is not known to be in polynomial time, so it is remarkable that allowing a constant factor approximation in the distance should allow us to answer online queries in *sublinear* time. Many tools are required to achieve this result, including the planar separator theorem, balanced trapezoidal decompositions, and approximation algorithms for vertex cover. A completely new technique that we apply here is sampling in range spaces of *unbounded* VC dimension. By showing that online 3D reconstruction requires $\Omega(\varepsilon_{\mathcal{D}}^{-1})$ time, we prove that convexity reconstruction represents yet another complexity gap between 2D and 3D. We leave the online reconstruction of general terrains and, most interesting, of arbitrary polytopes as an open problem. Of independent interest, we also give a sublinear time algorithm that computes small balanced separators for geometrically embedded planar graphs. It is unlikely that this construction is optimal and we pose an intriguing problem—the construction of optimal separators in sublinear time.

We are not aware of any line of work that our results can be compared with directly—except for Ailon et al. [2004b], where online property-preserving data reconstruction

---

[1]Throughout this article, "with high probability" is shorthand for: with probability at least $1 - n^{-c}$, for an arbitrarily large constant $c > 0$, where n is the size of the input.

[2]Unless indicated otherwise, all our algorithms are randomized. An ε-tester for a property $\mathcal{P}$ determines whether an input of size n satisfies $\mathcal{P}$ or requires at least εn modifications to do so. We use the notation $\widetilde{O}(f) = O(f)(\log f)^{O(1)}$ and our claims of optimality are to be understood up to polylogarithmic factors.

[3]A terrain is said to have *bounded aspect ratio* if the *xy*-projections of the faces have bounded sides and angles.

was introduced and polylogarithmic time filters were provided for reconstructing monotone functions. Of course, offline geometric reconstruction has been studied before, but usually the metric is geometric (like the Hausdorff distance) and not combinatorial. Examples include finding the best approximation of surfaces satisfying certain criteria [Agarwal et al. 2002; Agarwal and Desikan 1997; Agarwal and Suri 1998; Amenta et al. 2000]. On the other hand, a notion of combinatorial distance is certainly present when studying the computational aspects of the Erdős-Szekeres theorem [Chvatal and Klincsek 1980] or other Ramsey-like results. Geometric properties have been well studied within the purview of property testing [Czumaj and Sohler 2001; Czumaj et al. 2000], program checking [Mehlhorn et al. 1996], and sublinear algorithms [Chazelle et al. 2003]. Efficient testers have been given for convexity [Czumaj and Sohler 2001; Czumaj et al. 2000; Ergun et al. 1998], clustering [Alon et al. 2000; Indyk 1999b, 1999a; Mishra et al. 2001], and Euclidean MST [Czumaj et al. 2003; Czumaj and Sohler 2004], but there too the relevance to our work is only tangential.

Subsequent to this work, there has been some study of sublinear reconstruction for other properties in Saks and Seshadhri [2010], Kale et al. [2008], and Austin and Tao [2010]. Some extensions to this model of sublinear reconstruction were discussed in Saks and Seshadhri [2010] and Austin and Tao [2010]. The specific property of monotonicity has also received attention in Bhattacharyya et al. [2010] and Chakraborty et al. [2011].

2. CONVEXITY FILTERS FOR POLYGONS

In this section, we shall deal with the problem of reconstructing convexity for polygons. For simplicity, we shall begin with reconstructing *polygonal chains* instead of polygons. For simplicity of presentation, we will make the assumption that these chains are terrains (the projections of the edges on the x-axis do not overlap). The filter for chains will then be extended to handle general polygons. The input is a 2D polygonal chain $\mathcal{D}$, ie, a polygonal curve with points $p_1, \ldots, p_n$, where each consecutive set of points is joined by a directed line segment. These segments are the *edges* of $\mathcal{D}$. The set of edges will be denoted by E. The chain $\mathcal{P}$ induces a natural ordering on the edges – given two edges $e = p_i p_{i+1}$, $f = p_j p_{j+1}$, $e \preceq f$ if $i \leq j$ ($e \prec f$ if $i < j$). We can define the interval $[e, f] = \{g \in E | e \preceq g \preceq f\}$.

We study the two following ways of representing $\mathcal{D}$.

(1) The edge set E is stored in a binary search tree, using the ordering on edges.
(2) The edge set E is stored in an ordered doubly linked list, from which we can access a random edge and walk from it in either direction. In addition, there exists an oracle that give the ordering of edges (given edges e, f, g, the oracle outputs their order in $\mathcal{D}$). Note that for the case of terrains, this is trivial to implement.

Note that the first model allows binary searches among the edges, whereas the second one does not. Let $\varepsilon_{\mathcal{D}} n$ denote the minimum number of edges that need to be modified to make $\mathcal{D}$ lower convex, without changing the ordering of E. This modification only refers to changing the actual coordinates of the points of $\mathcal{D}$. Lower convexity means that (after the modification) for any edge e, it must point towards the increasing x direction, and all edge must lie to the left halfspace defined by e. Therefore, two edges are in convex position with each other if they both point from left to right and lie to the left of each other.

The first case, as mentioned earlier, can be easily solved using tools developed in Ailon et al. [2004a, 2004b]. This will hopefully give the reader a flavor of geometric reconstruction. Building on some of these ideas and introducing some new tools, we can develop filters for the second case.

What makes the 2D case remarkable is that a certain easily testable property allows us to classify any given edge in one of two categories (good or bad) in a way that leads to a filtering mechanism with a constant approximation factor. (An almost identical classification was used for filtering monotone functions in Ailon et al. [2004b].)

Definition 2.1.   A pair $\langle e, f \rangle$ is a *violation* if e and f are not in convex position. We also say that e violates f and vice-versa.

The following transitivity relation is immediate: if $e \prec f \prec g$ and $\langle e, g \rangle$ is a violation, then so is at least one of $\langle e, f \rangle$ or $\langle f, g \rangle$. This is a simple consequence of the properties of convexity. Note that for an edge e that points from right to left, the pair $\langle e, f \rangle$ is a violation for all $f \in E$.

Definition 2.2.   Given any $0 < \delta < 1/2$, an edge e is called δ-*bad* if there exists an edge f such that either (i) $e \prec f$ and the number of $g \in [e, f]$ that violate e is at least $(1/2 - \delta)|[e, f]|$ or (ii) $f \prec e$ and the number of $g \in [f, e]$ that violate e is at least $(1/2 - \delta)|[f, e]|$. The edge f is referred to as a *witness* to e's badness. An edge that is not δ-bad is called δ-*good*.

The next lemma is crucial for proving the correctness of the filter.

LEMMA 2.3.

(i) The 0*-good edges have no violating pairs;*
(ii) at least $\varepsilon_{\mathcal{D}} n$ edges are 0*-bad; and*
(iii) no more than $(3 + 8\delta/(1 - 2\delta))\varepsilon_{\mathcal{D}} n$ edges are δ-bad.

PROOF (FROM AILON ET AL. [2004A]).   Note that by transitivity, for any $e \prec f$ such that $\langle e, f \rangle$ is a violating pair, either e or f (or both) is 0-bad. Therefore, if we were to remove all the 0-bad edges, the remaining edges would be in convex position; hence (i) and (ii).

We start by assigning to each δ-bad e a witness f_e to its badness (if many witnesses exist, we just choose any one). If $f_e \succ e$, then e is called *right-bad*; else it is *left-bad*. (Obviously, the classification depends on the choice of witnesses.)

Let C be a set of $\varepsilon_{\mathcal{D}} n$ edges where $\mathcal{D}$ can be modified to make it convex. To bound the number of right-bad edges, we charge C with a credit scheme. (Then we apply a similar procedure to bound the number of left-bad edges.) Initially, each element of C is assigned one unit of credit. For each right-bad $e \notin C$ (in reverse order from right to left), *spread* one credit among all the g such that $e \preceq g \preceq f_e$ and $\langle e, g \rangle$ is a violation (note that g must be in C). We use the word "spread" because we do not simply drop one unit of credit into one account. Rather, viewing the accounts as buckets and credit as water, we pour one unit of water one infinitesimal drop at a time, always pouring the next drop into the least filled bucket.

We now show that no edge in C ever receives an excess of $2 + 4\delta/(1 - 2\delta)$ units of credit. Suppose by contradiction that this were the case. Let e be the right-bad that causes some edge g's (g belongs to C) account to reach over $2 + 4\delta/(1 - 2\delta)$. By construction e is not in C; therefore, the excess occurs while right-bad e is charging an edge g such that $e \prec g \preceq f_e$ and $\langle e, g \rangle$ is a violation. Note that, because $e \notin C$, any g satisfying these two conditions belongs to C (let us denote the number of such edges by l) and thus gets charged. With the uniform charging scheme, this ensures that all of these l elements of C have the same amount of credit by the time they reach the excess value, which gives a total greater than $l(2 + 4\delta/(1 - 2\delta))$. By definition of right-badness, $l \geq (1/2 - \delta)|[e, f_e]|$. But none of these accounts could be charged before step f_e; therefore,

$$(1/2 - \delta)|[e, f_e]|(2 + 4\delta/(1 - 2\delta)) < |[e, f_e]|,$$

which is a contradiction.

```
light-test(𝒜, a, b, k)

for each x ∈ 𝒜
   sample it k times and compute the average x̂
form 𝒜' = { x ∈ 𝒜 | x̂ > (a + b)/2 }
if |𝒜'| = 0 then
   output "a-light"
if |𝒜'| ≥ |𝒜|/2 then
   output "b-heavy"
light-test(𝒜', a, b, k + c)
/* c is sufficiently large constant */
```

Fig. 2.   light-test.

Of the total of at most $2 + 4\delta/(1-2\delta)\varepsilon_{\mathcal{D}}n$ units of credit, $\varepsilon_{\mathcal{D}}n$ units of credit came from initially assigning the edges of C one unit of credit each. Therefore, there are at most $1 + 4\delta/(1-2\delta)\varepsilon_{\mathcal{D}}n$ right-bad edges. By applying a similar argument for left-bad edges (this time charging from left to right), we prove (iii).   □

With this classification of edges, we now give an outline of the filter to convexity. Our aim is to preserve δ-good edges and modify only δ-bad edges. This will ensure convexity and that at most $(3 + O(\delta))\varepsilon_{\mathcal{D}}n$ edges will be modified. When a query comes for a δ-bad edge e, we use a modification procedure to "fix" e. This is done by finding the closest δ-good edges f and g on both sides of e and then simply replace all edges in between by linearly interpolating between f and g.

Any query for an edge that has not been previously queried will lead to a committing of the value assigned to the edge. Changes will occur in intervals (a whole interval $[e, f]$ may be changed in one query) and all changed values will be committed. We will assume that a data structure stores all committed intervals (and their corresponding values) and given any edge, we can tell in $O(\log n)$ time whether it has been committed, and if not, find the closest committed values.

2.1. Edges in a tree

In the first model, where E is stored in a binary search tree, the filter for convexity will be almost the same (just a slight variation) as the $O(\log^2 n \log \log n)$ filter for monotonicity given in Ailon et al. [2004b]. In the case of monotonicity, randomly sampling from an interval was trivial - in the case of E being stored in a binary search tree, choosing one random sample from some interval of edges takes $O(\log n)$ time. Therefore, when the monotonicity filter is modified for handling convexity, the per query time becomes $\widetilde{O}(\log^3 n)$. For completeness, we describe the procedures required to construct this filter.

Definition 2.4.   Let $\mathcal{A}$ be the joint distribution of m independent 0/1 random variables $x_1, \ldots, x_m$, which can be sampled independently. If $\mathbf{E}[x_i] \le a$ for all i, then $\mathcal{A}$ is called *a-light*; else it is *a-heavy*.

LEMMA 2.5 [AILON ET AL. 2004A].   *Given any fixed $a < b$, if $\mathcal{A}$ is either a-light or b-heavy; then, with probability $2/3$, we can tell which is the case in $O(m)$ time. If $\mathcal{A}$ is neither, the algorithm returns an arbitrary answer.*

The algorithm of Lemma 2.5 is given in Figure 2. The test is run by calling *light-test*$(\mathcal{A}, a, b, c')$ for sufficiently large constant c'. In the following we use the algorithm of Lemma 2.5 to test, with high probability of success, whether a given edge e is δ-bad

```
goodness-test (𝒟, eᵢ, δ, k)

   repeat the following c log k times for a big enough constant c
        for each 1 ≤ j ≤ (2/δ) ln n
            define x_{2j-1}^{(i)} = α[i, i + (1 + δ)^j] and x_{2j}^{(i)} = β[i - (1 + δ)^j, i]
        let 𝒜 be the distribution (x₁, x₂, ⋯)
        run light-test(𝒜, (1/2 − 2δ), (1/2 − 3δ/2), c′)
   if majority of above tests output (1/2 − 2δ)-light
        then output 2δ-good
        else output δ-bad
```

Fig. 3. Testing if an edge i is δ-bad or 2δ-good.

or 2δ-good, for any fixed $\delta > 0$. Given an interval $[e, f]$, we define two 0/1 random variables $\alpha[e, f]$ and $\beta[e, f]$: given a random edge $g \in [e, f]$, $\alpha[e, f] = 1$ (respectively, $\beta[e, f] = 1$) iff $\langle e, g \rangle$ (resp. $\langle f, g \rangle$) is a violation. Note that sampling from this interval takes $O(\log n)$ time. The algorithm *goodness-test* (Figure 3) tests if a given edge e_i (this means that e_i is the ith edge in terms of the ordering of edges) is δ-bad or 2δ-good.

LEMMA 2.6 [AILON ET AL. 2004B]. *Given any fixed $\delta > 0$ and a parameter k, if e is either δ-bad or 2δ-good, then* goodness-test *will tell which is the case in time $O(\log^2 n \log k)$ and with probability at least $1 - 1/k$.*

We now have a procedure that separates from 2δ-good edges from δ-bad ones. The strategy for reconstruction is as follows – given a query for e, first test it using *goodness-test*. If the output is "2δ-good," keep the edge. Otherwise, change the edge. By Lemma 2.3, we know that such a procedure will only change $(3 + O(\delta))\varepsilon_f n$ edges. Changing an edge will involved changing the endpoints – we will refer to that as changing the value of an edge.

Given a query for e, suppose we decide that e is not 2δ-good. Our aim now is find some replacement value for this edge. The first step to doing this is to attempt to find the closest 2δ-good edges both to the left and right of e. Because of the sublinear time constraint, this cannot be done exactly. Instead, the idea is to find an interval I_0 around e such that the fraction of 2δ-good edges in I_0 is at least $\Omega(\delta)$, but their fraction in a slightly smaller interval is $O(\delta)$. Once such an interval is found, we can find 2δ-good edges in this interval and use them to reconstruct e. This is described in Figure 4. The procedure has been modified from the one given in Ailon et al. [2004b] to handle convexity. The following lemma is essentially taken from Ailon et al. [2004b]. The proof needs some marginal modifications for our current setting.

LEMMA 2.7. *The procedure find-good-edge returns, with probability $1 - 1/n^3$, an edge e that is in convex position and to the right of a δ-good edge in I. This requires the existence of at least a δ-fraction of 2δ-good edges in I. The running time of find-good-edge is $\widetilde{O}(\log^3 n)$, for fixed δ.*

PROOF. The running time bound is obvious to see. The expected number X of δ-bad samples for which *goodness-test* outputs "2δ-good" is at most $c\delta(1 - \delta) \log n$, by Lemma 2.6. The expected total number Y of samples for which *goodness-test* outputs "2δ-good" is at least $c(1 - \delta^2) \log n$. The probability that X exceeds $Y/3$ is at most $1/n^4$ if c is chosen large enough, using Chernoff bounds. Therefore, with probability at least $1 - 1/n^4$, more than a 2/3-fraction of the values appended to the list L are δ-good edges.

```
find-good-edge (𝒟, I, δ)

   set L as an empty list
   randomly select cδ⁻¹ log n edges from I for a big enough constant c.
   for each e in random sample
        if goodness-test (𝒟, f, δ, δ⁻²) outputs ''2δ-good''
            then append e to L
   arrange L in reverse order f₁,···,fₖ as they appear in 𝒟
   let L' be an empty stack
   for i from 1 to k
     if fᵢ is in convex position with head of L' or L' is empty
       push eᵢ onto L'
     else pop L'
   return tail of L'
```

Fig. 4. Finding a good edge in an interval, assuming that I has at least a δ-fraction of 2δ-good edges.

Note that all δ-good edges are in convex position. Whenever edge f_i causes a pop from L', either the head that was popped or f_i are δ-bad. By transitivity, when the procedure ends, the stack L' consists of edges in convex position. Also, there must be at least $|L|/3$ edges in L' and at least one of them is δ-good. This completes the proof. □

In *find-good-edge*, we could also order edges in the opposite direction and find an edge that is in convex position and to the left of a δ-good edge. We now have all the necessary tools to show the reconstruction of $\mathcal{D}$. The reconstruction algorithm *convexify* is practically the same as the procedure *monotonize* in Ailon et al. [2004b], used for reconstructing monotonicity.

To find a good interval, we do a binary search among all the intervals of length $(1+\delta)^j$ ($j = 0, 1, \ldots$) starting or ending at index i, that is, edges with indices in $[i, i + (1 + \delta)^j]$ and $[i - (1 + \delta)^j, i]$. There are $O(\log n)$ such intervals, and thus the running time is $O(\log\log n)$ times the time spent for each interval. The overall algorithm *convexify* is shown in Figure 5. The following claim together with a suitable rescaling of δ concludes the proof of correctness of *convexify*. The following proof is taken from Ailon et al. [2004b].

CLAIM 2.8. *Given any $0 < \delta < \frac{1}{2}$, with probability $1-1/n$,* convexify *outputs a convex polygon that is within distance $(2 + \delta)\varepsilon$ to $\mathcal{D}$. Given a query e, the output is computed online in time $\widetilde{O}(\log^3 n)$, when δ is assumed to be fixed.*

PROOF. First we analyze the running time. The first *goodness-test* takes $\widetilde{O}(\log^3 n)$ time. If the algorithm determines that e is δ-bad, then the while-loops run $O(\log\log n)$ times. In one iteration of the while-loop, the algorithm calls *goodness-test* $O(\log n)$ times. Each call takes $O(\log^2 n)$ time by Lemma 2.6. Therefore, the time complexity of the while-loop is $\widetilde{O}(\log^3 n)$. By Lemma 2.7, the running time of the call to *find-good-edge* is $\widetilde{O}(\log^3 n)$. The time complexity of the algorithm is therefore $\widetilde{O}(\log^3 n)$.

Let us first look at the while-loop. If I has more than 2δ-fraction of 2δ-good edges, then the number of "2δ-good" outputs is $< \frac{3}{2}c \log n$ with inverse polynomial probability. This can be shown through Chernoff bounds. On the other hand, if I has less than

```
convexify (D, δ, e_i)

    if e_i was already committed to then output committed edge
    let l be closest committed edge on left of e_i
    let r be closest committed index on right of e_i
    if goodness-test(D, e_i, δ, n^3) outputs ''2δ-good''
       and e_i is in convex position with l and r
          then commit to e_i and output it

    (*)
    set j_max = ⌊ln(i − l)/ ln (1 + δ)⌋ and j_min = 0.
    while j_max − j_min > 1
          set j = ⌊(j_max + j_min)/2⌋,  I = [i − (1 + δ)^j, i]
          choose random sample of size cδ^{−1} log n from edges corresponding to I, for large c
          for each edge e' in sample
                run goodness-test(D, e', δ, c_1), for large c_1
          if number of "2δ-good" outputs is ≥ 3/2 c log n
                then set j_max = j
                else set j_min = j
    if j_max ≠ ⌊ln(i − l)/ ln (1 + δ)⌋
          then set I_l = [i − (1 + δ)^{j_max}, i] and set f_l = find-good-edge(f, I_l, δ)
          else set f_l = l
    if f_l is not in convex position with l
       then set f_l = l
    (**)
    repeat lines (*)...(**) for right side of i, obtaining f_r

    join right endpoint of f_l and left endpoint of f_r by straight line
    commit to all edges between f_l and f_r and output accordingly
```

Fig. 5. Convexity reconstruction.

δ-fraction of 2δ-good edges, then the number of "2δ-good" outputs is $> \frac{3}{2} c \log n$ with inverse polynomial probability. Consider the following events.

—The intervals I_l and I_r have at least a δ-fraction of 2δ-good edges and the call to *find-good-edge* succeeds.
—The interval $I_{min} = [i − (1 + \delta)^{j_{min}}, i]$ has at most 2δ-fraction of 2δ-good edges.

Both these events hold with probability $> 1 − 1/n^4$. The intervals I_l, I_r, and I_{min} are constructed at most $O(n^2)$ times (over all queries). Now consider the event that the call to *goodness-test* (in line 4) correctly distinguishes between δ-bad and 2δ-good edges. As shown in Lemma 2.6, this happens with probability $> 1 − 1/n^3$. This is totally called at most n times. By a union-bound, all of these events occur (for every query) with probability $> 1 − 1/n^d$, for some positive constant d. Therefore, we henceforth assume that these events always occur (in other words, the probability of something "bad" happening is polynomially small).

It is obvious that the output is always convex, since we never add edges that are not in convex position with already committed edges. We now show that the number of edges modified is at most $(3 + O(\delta))\varepsilon_D n$ edges. Edges that are changed are either 2δ bad, 2δ-good edges that violate already committed edges, or 2δ-good edges that are present in an interval that gets modified. The total number of 2δ-bad edges is less than $(3 + 17\delta)\varepsilon_D n$, by Lemma 2.3. We can assume that for $I_{min} = [i − (1 + \delta)^{j_{min}}, i]$, the fraction of 2δ-good edges in I_{min} is at most 2δ. Since by the end of the algorithm $j_{max} \leq j_{min} + 1$, the fraction of 2δ-good edges in $I_r = [i, i + (1 + \delta)^{j_{max}}]$ (or $I_l = [i − (1 + \delta)^{j_{max}}, i]$) is at most 4δ. Therefore, when *find-good-edge* is called on an interval, the number of 2δ-good edges in it is at most 4δ. Let us focus on the call of *find-good-edge* for f_l. By Lemma 2.7, the edge

---

goodness-test $(\mathcal{D}, e, \delta)$

```
walk along √n edges to both sides of e - call this sequence of edges S
if some [e,f] (or [f,e]) ⊆ S has ≥ (1/2 − δ) fraction of violations
    output "δ-bad"
select (c/δ²)√n edges at random - call this set R
put R in order (from now on, [e,f] denotes [e,f] ∩ R)
if, for any [e,f](or [f,e]) ⊆ R, [e,f] (or [f,e]) has ≥ (1/2 − 3δ/2) fraction of violations
    output "δ-bad"
output "2δ-good"
```

Fig. 6.   Checking 2δ good-vs-δ bad.

output f_l is in convex position with some 2δ-good edge (which is to the left) in I_l. Any 2δ-good edge to the left of f_l that violates it must be inside I_l. Similarly, any 2δ-good edge to the right of f_r that violates it must also be inside I_r. Any 2δ-good edge that is changed by *convexify* must lie in an interval contain at most a 4δ-fraction of 2δ-good edges. The total number of edges modified is at most $(3 + 17\delta)\varepsilon_{\mathcal{D}} n/(1 - 4\varepsilon) \leq (3 + O(\delta))\varepsilon_{\mathcal{D}} n$.   □

This completes the proof of the following theorem.

THEOREM 2.9.   *For any sufficient small $\delta > 0$, there exists an $(\widetilde{O}(\log^3 n), 3 + \delta)$-filter for 2-dimensional convexity, where the input is a terrain represented by a balanced search tree.*

2.2. Edges in Linked List

We now look at how to extend these ideas when the polygonal chain $\mathcal{D}$ is represented as a doubly-linked list. We assume that there is an oracle that gives the order of edges in the curve. In other words, given a set of edges, we can set in the proper order with an extra logarithmic overhead. The first step is to describe *goodness-test* (refer to Figure 6) which separates 2δ-good edges from δ-bad ones. This essentially does the same as the old *goodness-test* - the main difference being that a sample of size $\sqrt{n}$ is used to detect violations.

LEMMA 2.10.   *The procedure* goodness-test$(\mathcal{D}, e, \delta)$ *differentiates between δ-bad and 2δ-good edges with probability $> 2/3$ in time $O(\sqrt{n} \log n)$.*

PROOF.   The time complexity is easy to see. Checking for violations in S can be done in $O(\sqrt{n})$ time. To put R in cyclic order, we can use the oracle that gives the order of edges and a binary search to complete this in $O(\sqrt{n} \log n)$ time. Once this is done, analyzing R takes $O(\sqrt{n})$ time. Note that the number of lookups is still $O(\sqrt{n})$.

In this proof, violator will refer to an edge that violates e. Let us assume that e is 2δ-good. It will obviously pass the violation test in S. For any interval I of size l (with e at one end), the probability (by Chernoff) that the number of nonviolators chosen from I is $\leq (1/2 + 7\delta/4)(c/\delta^2)l/\sqrt{n}$ is less than $e^{-c'l/\sqrt{n}}$ ($c' = \Theta(c)$). Similarly (again by Chernoff), the probability that the total number of edges chosen from I is $\geq (1 + \delta/4)(c/\delta^2)l/\sqrt{n}$ is less than $e^{-c'l/\sqrt{n}}$. Therefore, with probability at most $2e^{-c'l/\sqrt{n}}$, the sample from I has $\geq (1/2 - 3\delta/2)$ fraction of violators. By taking a union bound over all such intervals of size $> \sqrt{n}$ (and choosing c large enough), we can ensure that e passes the violation test in R with probability $> 2/3$.

Suppose e is δ-bad. There exists some interval I (with e at one end) such that the number of violators is more than a $(1/2 - \delta)$ fraction. If the size is $\leq \sqrt{n}$, e will be declared "δ-bad" during violation search in S. If the interval is $> \sqrt{n}$, then by an

---

find-closest (S, e, δ)

```
for j from 1 to δ⁻¹ log |S|
    take the interval Iⱼ of edges of size (1+δ)ʲ with left end e
    choose random sample of size (c/δ) log n in I
    for each edge e′ in sample
        if goodness-test(𝒟, e′, δ, n⁻⁴) outputs ''2δ-good''
            output e′ and exit procedure
```

Fig. 7.   Find a close good edge.

argument similar to the one given here, we can show that some interval in R will have more than a $(1/2 - 3\delta/2)$ fraction of violations with probability $2/3$. □

We will henceforth assume that *goodness-test* takes a fourth argument k, and boosts the probability of correctness to $(1 - 1/k)$ - the time complexity will now be $\widetilde{O}(\sqrt{n} \log k)$. The boosting can be done by repeating the algorithm $\log k$ times and then taking a majority vote. We now have a procedure that separates from 2δ-good edges from δ-bad ones. The strategy for reconstruction is the same as before: If the output is "2δ-good," keep the edge. Otherwise, change the edge.

Before we describe the main procedure *convexify*, we need a procedure *find-closest* (refer to Figure 7) that, given an input of an ordered set of edges S and another edge e, tries to find the closest 2δ-good edge to e that is larger in the order (it is easy to modify so that we find a smaller good edge).

CLAIM 2.11.   *The procedure find-closest(S, e, δ) (where S is an ordered set of edges and e is 2δ-bad) either outputs nothing or it outputs a δ-good edge $e′$. With probability $> 1 - n^{-3}$ - the interval $[e, e′]$ contains at most a 2δ-fraction of 2δ-good points. If it outputs nothing, then S has less than a 2δ-fraction of 2δ-good edges. The running time is $\widetilde{O}(\sqrt{n})$.*

PROOF.   The running time is easy to see. The proof here is essentially a repetition of arguments used in the proof of Lemma 8. Using Chernoff arguments, with probability $> 1 - n^{-3}$, if any of intervals I_j has more than a δ-fraction of 2δ-good edges, then some $e′ \in I$ will be output as "2δ-good." Therefore, e cannot be δ-bad. If nothing is output, S certainly has less than a 2δ-fraction of 2δ-good edges. On the other hand, if some interval I_j has more a 2δ-fraction of 2δ-good edges, then I_{j-1} will have more than a δ-fraction of 2δ-good edges. This completes the proof. □

Call an edge *unsafe* if it belongs to an interval that contains less than a 2δ-fraction of 2δ-good points.

CLAIM 2.12.   *Number of unsafe edges $\leq 16\delta \times$ Number of 2δ-bad edges.*

PROOF.   Consider an unsafe edge e that belongs to $[u, u′]$. We can argue that at least one of $[u, e]$ and $[e, u′]$ has less than a 4δ fraction of 2δ-good edges. Let all the e's that satisfy the first condition be called *downward*. By a charging argument used in the proof of Lemma 2.3, we can show that the number of downward edges is $\leq 4\delta \times$ number of 2δ-bad edges. Similarly, we can show the same for nondownward edges. Adding all the bounds, we prove the lemma. □

We now complete the proof of the main theorem of this section. The main filter procedure is *convexify*, described in Figure 8.

```
                          convexify(𝒟, e, δ)

    if e has been committed, output committed value
    if goodness-test(𝒟, e, δ, n³) outputs "2δ-good", output and commit current value
    find closest committed edges to the back, b and the front, f
    walk along √n edges to both side of e - call this sequence of edges S
    use find-closest(S, e, δ) to find closest good edge to back, b_S and front, f_S
    select (c/δ²)√n log n edges at random - call this set R
    put R in cyclic order
    use find-closest(R, e, δ) to find closest good edge to left, b_R and right, f_R
    take closest (to e) of all b, b_S, b_R (and of f, f_S, f_R) obtained
    linearly interpolate to assign values to all edges in between
    output interpolated value and commit interval
```

Fig. 8.   Correcting edge e in $\mathcal{D}$, with parameter δ.

THEOREM 2.13.   *There exists a $(\widetilde{O}(\sqrt{n}), 3+\delta)$-filter for 2-dimensional convexity, where the input is a terrain given as a linked-list and there is an ordering oracle present.*

PROOF.   The first call to *goodness-test* takes $\widetilde{O}(\sqrt{n})$ time. The calls to *find-closest* takes $\widetilde{O}(\sqrt{n})$, proving the running time bound. Since the calls to *goodness-test* and *find-closest* error with probability n^{-3} and there are at most $O(n^2)$ such calls (over all queries), we can assume that they always output the correct answer. To show that the output is convex, note that any edge that is committed initially must be δ-good. Furthermore, the interpolation is done only between δ-good edges (Claim 2.11). Therefore, the output is always convex.

Suppose e is 2δ-bad (when e is 2δ-good, it is committed). Let F_e be the smallest interval of size $(1 + \delta)^j$ (for some j such that the with left end e) which has more than a 4δ-fraction of 2δ-good points. If $|F_e| \le \sqrt{n}$, then *find-closest*(S, e, δ) will find a δ-good edge f_s such that $[e, f_s]$ contains at most a 2δ-fraction of 2δ-good edges. Suppose $|F_e| > \sqrt{n}$. By a Chernoff bound, we can show that with probability $> 1 - n^{-3}$, the number of 2δ-good edges in $R \cap F_e$ is at least a 2δ-fraction of $R \cap F_e$. This implies that a δ-good edge in F_e will be detected when *find-closest*(R, e, δ) runs. A similar argument can be made for intervals with e at their right end. The only edges that are changed are those that are either 2δ-bad or are unsafe. By Lemmas 2.12 and 2.3, the number of 2δ-good points changed is $O(\delta \varepsilon_{\mathcal{D}} n)$. This shows that *convexify* is a $(\widetilde{O}(\sqrt{n}), 3 + O(\delta))$-filter. (Rescale δ to complete proof.)   □

2.3. Testing

First, we describe a convexity tester. The input polygon $\mathcal{D}$ is given as in linked list format with an ordering oracle. A *tester* is a randomized algorithm which, given $\mathcal{D}$ and $0 < \varepsilon < 1$, will do the following in sublinear time.

—If $\mathcal{D}$ is convex, output "convex" with prob $> 2/3$.
—If $\mathcal{D}$ is ε-far from being convex, output "not convex" with prob $> 2/3$.

We prove that the procedure *is-convex*, described in Figure 9, is a valid property tester.

THEOREM 2.14.   *There exists a tester for convexity of polygons given in linked list format (and with ordering oracle present) using $O(\varepsilon^{-1} n^{1/3})$ lookups and takes $\widetilde{O}(\varepsilon^{-1/2} n^{1/3})$ time.*

We contrast this with the convexity tester obtained in Czumaj et al. [2000]. Their running time's dependence on n is $O(n^{2/3})$. In their model, there is "polygonal information"

```
                              is-convex (𝒟, ε)

choose cε^{-1/2}n^{1/3} edges at random (for large enough constant c)
if they are not in convex position, output "not convex"
repeat c'ε^{-1} times (for large enough constant c')
    choose random edge e and walk in 𝒟 for n^{1/3} steps forwards and backwards
    if these edges are not in convex position, output "not convex"
output "convex"
```

Fig. 9.   Testing if polygon $\mathcal{D}$ is convex.

and the input is just a set of points in the plane. Hence, they have a higher running time (a matching lower bound was also given in Czumaj et al. [2000]). It is interesting to note that the time complexity comes down when we use the linked list structure of the edges.

We will analyse the procedure *is-convex* and show that it satisfied the requirements of Theorem 2.14. If $\mathcal{D}$ is convex, then the tester obviously accepts with probability 1. From now on, we assume that $\mathcal{D}$ is ε-far from being convex.

The set of edges is denoted by E. Consider a set of εn edges whose removal makes $\mathcal{D}$ convex. Let these edges be called *unsound* edges (this set will be denoted by U, $|U| \geq \varepsilon_{\mathcal{D}} n$). The remaining edges are *sound* (this set will be S). Note that all of S is in convex position and that there cannot be a set of convex edges whose size is $> |S|$.

LEMMA 2.15.   *There exists a one-to-one function $\alpha : U \to E$ such that $\forall u \in U$, $\alpha(u)$ violates u.*

PROOF.   We will construct α through an iterative process. Initially, we start with the set $T = U$. If there exists a violating pair $\langle u_1, u_2 \rangle$ $(u_1, u_2 \in T)$, we assign $\alpha(u_1) = u_2$ and $\alpha(u_2) = u_1$. Then we remove u_1, u_2 from T and repeat. In the end, we end up with a set T of unsound edges which do not violate each other. In other words, T is in convex position. Let us now construct a bipartite graph G with T on one side and S on other. Any two violating edges are connected. Take any set $T' \subseteq T$. T' is in convex position and is totally unsound. Therefore, T' should have at least $|T'|$ violators in S. If not, we could replace all these violators in S by T', and we would have a set of convex edges whose size is greater than $|S|$. The number of neighbours of T' in G is $\geq |T'|$. By Hall's Theorem, G has a perfect matching. For every $u \in T$, we set $\alpha(u)$ to be the edge in S that matches u. This completes the construction of α.   □

A contiguous set of unsound edges in convex position will be called a *stretch*.

CLAIM 2.16.   *For any stretch L of length k, there exist sets of edges A_L, B_L such that $A_L \subseteq L$, $B_L \cap L = \emptyset$, $|A_L|, |B_L| \geq k/4$, and every edge in A_L violates every edge in B_L. Furthermore, for two different stretches L and L', B_L and $B_{L'}$ are disjoint.*

PROOF.   Let α be the function given in Lemma 2.15. For every u in the middle $k/2$ edges of L, consider $\alpha(u)$. Without loss of generality, we can assume that at least half of these edges lie in front (along $\mathcal{D}$) of L. Let all these edges be B_L, and let the rightmost $k/4$ edges of L be A_L. By transitivity, every edge in A_L violates every edge in B_L. For the last part of the claim, we observe that α is a one-to-one function.   □

We now need some notation for certain probabilistic events. Consider the edges sampled in the first step of *is-convex*. For stretch L, let k_L denote the size of L. Let $\mathcal{E}_L$ denote the event that some element of A_L is chosen in the first $32\varepsilon^{-1/2}n^{1/3}$ samples and that some element of B_L is chosen in the next $32\varepsilon^{-1/2}n^{1/3}$ samples. (We will denote the former event as $\mathcal{E}_L(A)$ and the latter as $\mathcal{E}_L(B)$. Hence, $\mathcal{E}_L = \mathcal{E}_L(A) \cap \mathcal{E}_L(B)$.) For every

stretch L, let random variable X_L be the indicator variable for the event $\mathcal{E}_L$. We use $var(X)$ to denote the variance of random variable X.

CLAIM 2.17.

(1) $\mathbf{E}[X_L] = \Pr(\mathcal{E}_L) > [1 - e^{-8k_L \varepsilon^{-1/2} n^{-2/3}}]^2$
(2) $var(\sum_L(X_L)) \le \mathbf{E}[\sum_L X_L]$.

PROOF. The probability that $\mathcal{E}_L$ occurs is:

$$\left[1 - \left(1 - \frac{k_L}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}}\right]^2 > \left[1 - e^{-8k_L \varepsilon^{-1/2} n^{-2/3}}\right]^2.$$

We now bound the variance of $\sum_L X_L$. Observe that the X_L's are negatively correlated. Consider two distinct stretches L and L'. Conditioned on $X_L = 1$, the probability that $X_{L'} = 1$ decreases. This is because $A_L \cap A_{L'} = B_L \cap B_{L'} = \emptyset$. For a more formal treatment, consider $\mathcal{E}_L(A)$ and $\mathcal{E}_{L'}(A)$. We have

$$
\begin{aligned}
\Pr(\mathcal{E}_L(A) \cap \mathcal{E}_{L'}(A)) &= \Pr(\mathcal{E}_L(A)) + \Pr(\mathcal{E}_{L'}(A)) + \Pr(\overline{\mathcal{E}_L(A) \cup \mathcal{E}_{L'}(A)}) - 1 \\
&= 1 - \left(1 - \frac{k_L}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}} - \left(1 - \frac{k_{L'}}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}} + \left(1 - \frac{k_L + k_{L'}}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}} \\
&= \left[1 - \left(1 - \frac{k_L}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}}\right]\left[1 - \left(1 - \frac{k_{L'}}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}}\right] \\
&\quad - \left[\left(1 - \frac{k_L}{4n}\right)\left(1 - \frac{k_{L'}}{4n}\right)\right]^{32\varepsilon^{-1/2} n^{1/3}} + \left(1 - \frac{k_L + k_{L'}}{4n}\right)^{32\varepsilon^{-1/2} n^{1/3}} \\
&\le \Pr(\mathcal{E}_L(A)) \Pr(\mathcal{E}_{L'}(A)).
\end{aligned}
$$

We can argue analogously for $\mathcal{E}_L(B)$ and $\mathcal{E}_{L'}(B)$. Since $\Pr(\mathcal{E}_L(A) \cap \mathcal{E}_{L'}(A))$ and $\Pr(\mathcal{E}_L(B) \cap \mathcal{E}_{L'}(B))$ are independent, we get that $\mathbf{E}[X_L X_{L'}] \le \mathbf{E}[X_L]\mathbf{E}[X_L]$. We can conclude that $var(\sum X_L) = \mathbf{E}[(\sum X_L)^2] - \mathbf{E}[\sum X_L]^2 \le \mathbf{E}[\sum X_L]$. □

PROOF OF THEOREM 2.14. The lookup complexity bound is obvious. The time complexity of checking if $c\varepsilon^{-1/2} n^{1/3}$ edges are in convex position requires $\widetilde{O}(\varepsilon^{-1/2} n^{1/3})$ time.

We need to show that with high probability, a violation will be detected. The tester has two stages - the first involves sampling $O(n^{1/3})$ edges, and the second involves walks of length $n^{1/3}$. We split into three cases.

Case 1. There exists a stretch L of length $k > \delta\varepsilon^{1/2} n^{2/3}$ (for some small enough constant $\delta > 0$). By Claim 2.17, the probability that $\mathcal{E}_L$ occurs (and hence a violation is detected) is at least $2/3$.

Case 2. At least $\varepsilon n/2$ unsound edges lie in stretches of length $< n^{1/3}$. Such an edge will be found with high probability in the second stage. Walking for $n^{1/3}$ steps in both directions along $\mathcal{D}$ ensures that a violation will be detected.

Case 3. At least $\varepsilon n/2$ unsound edges lie in stretches of length $\ge n^{1/3}$ and all stretches have length $\le \delta\varepsilon^{1/2} n^{2/3}$. By Claim 2.17,

$$\mathbf{E}\left[\sum_L X_L\right] > \sum_L \left[1 - e^{-8k_L \varepsilon^{-1/2} n^{-2/3}}\right]^2 \ge \frac{\varepsilon n}{2n^{1/3}} \frac{64n^{2/3}}{4\varepsilon n^{4/3}} \ge 8.$$

(Since no stretch has length $> \delta\varepsilon^{1/2} n^{2/3}$ and $x > 0$, we can use the inequality $e^{-x} < 1 - x/2$. A standard convexity argument then gives us the above bound.) By

Claim 2.17, $var(\sum X_L) \leq \mathbf{E}[\sum X_L]$. By Chebyschev's inequality, $\sum X_L$ will take a positive value (and therefore a violation will be detected) with probability at least 2/3. □

2.4. General Polygons

So far, we made the simplifying assumption that $\mathcal{D}$ looks like a function in 2-dimensions. We now show how to remove this assumption, for both testing and reconstruction. The algorithms given will remain the same - only the correctness proofs need to be changed. We choose some arbitrary edge (referred to as e_0) and orient the axes such that e_0 is parallel to the y-axis and points in the negative direction.

We now modify the definition of a violation. First, we call an edge *forward* if it points in the positive x direction (otherwise, it is called *backward*). Given edges e_1, e_2, the pair (e_1, e_2) is a violation if any of the following holds.

(1) e_1 and e_2 are not in convex position.
(2) e_1 and e_2 are forward edges, they are in the cyclic order e_0, e_1, e_2 and the x-projection of e_2 is not strictly in front (along the x-axis) of the x-projection of e_1.
(3) e_1 and e_2 are backward edges, they are in the cyclic order e_0, e_1, e_2 and the x-projection of e_1 is not strictly in front (along the x-axis) of the x-projection of e_2.
(4) e_1 is a backward edge, e_2 is a forward edge, and they are in the cyclic order e_0, e_1, e_2.

We can now modify the proof of Claim 2.16. (Note that the proof of Theorem 2.14 will then hold for general polygons.)

PROOF OF CLAIM 2.16. For any u in the middle $k/2$ edges of L, consider $\alpha(u)$. Note that if the cyclic order is $e_0, u, \alpha(u)$, then the cyclic order is $e_0, v, \alpha(u)$ for any $v \in L$. (A similar statement can be made if the order is $e_0, \alpha(u), u$.) Using this (and the new definition of a violation), it can be shown that $\alpha(u)$ violates either the first $k/4$ or the last $k/4$ edges. Without loss of generality, we can assume that half of the $\alpha(u)$'s (let this be B_L) violate the first $k/4$ edges of L (let this be A_L). □

Proving that reconstruction still works is relatively easy. Essentially, we need to prove that Lemma 2.3 is still correct. The proof of (i), (ii) only uses the transitivity property of convexity. Given edges e_1, e_2, the transitivity property now holds in either $[e_1, e_2]$ or $[e_2, e_1]$. In the proof of (iii), we perform a charging procedure that starts from the rightmost end of $\mathcal{D}$. Now, since $\mathcal{D}$ now has no "rightmost" end, we need to choose some place to start the charging. We can use an argument similar to the one used in the proof of Lemma 2.12 to show there exists some e' that does not belong to any badness interval. The charging argument used to prove (iii) can now be done by starting from e' (and moving in cyclic and reverse cyclic orders) without affecting the proof.

2.5. Lower Bounds

Our first lower bound is for convexity reconstruction for 2-dimensions when the polygonal chain is given as a balanced binary tree of edges. We prove that any filter that guarantees to change at most $O(\varepsilon_{\mathcal{D}} n)$ edges has a worst-case lookup time of $\Omega(\log^3 n)$. All our lower bounds are in terms of *lookup complexity*. We do not deal with the actual computing model. Our main assumption is that knowledge of any set of input edges tells us nothing about edges that we have not looked up. Our lower bounds are essentially information theoretic in nature.

THEOREM 2.18. *Consider filter for 2-dimensional convexity where the polygon is expressed as a balanced binary tree which errors with probability $< 1/n$. The filter requires $\Omega(\log^3 n)$ lookups per query.*
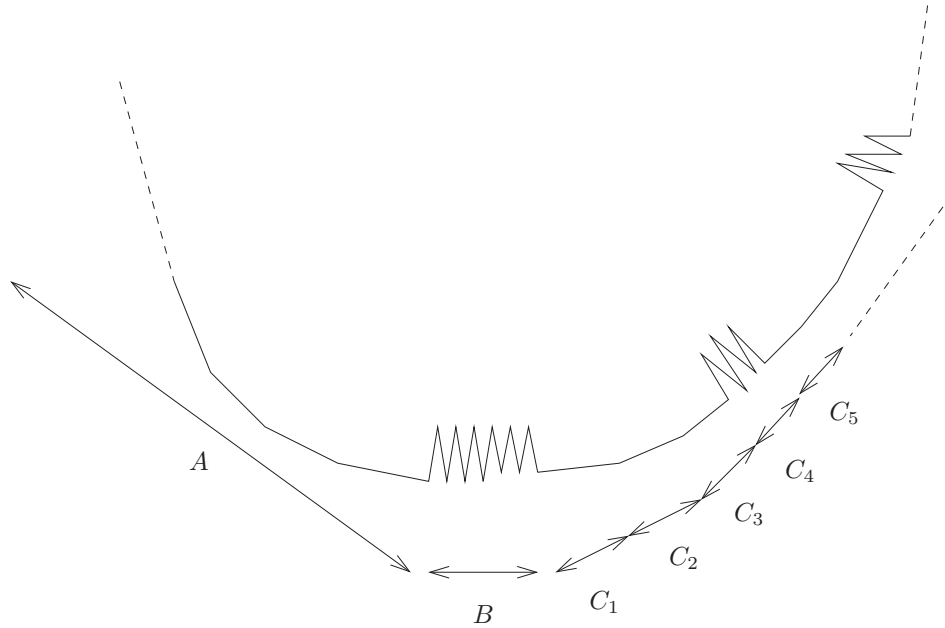
Fig. 10.   An input in Σ_i.

PROOF.   The proof essentially uses Yao's minimax lemma. We define a distribution of polygons of size n, which are to be reconstructed. We then consider a deterministic filter that guarantees with probability $> 1 - 1/n$ to change $< c\varepsilon_{\mathcal{D}}n$ edges, where $\mathcal{D}$ is the input polygon and c is any constant. We then show that there exists a query that will require $\Omega(\log^3 n)$ time to process. We will make the natural assumption that we have no geometric information about edges that we have not seen.

The final distribution Σ is a mixture of distributions Σ_i, where $\sqrt{n} \le (2c)^i \le \delta n$ (for δ a sufficiently small constant). We assume henceforth that c is a power of 2, and denote $2c$ by d. Let us first describe Σ_i. First, we start from a convex polygon for every Σ_i. Also, for every different possible input, this initial polygon is different. These two facts ensure that there does not exist one fixed geometric solution that works for a query. In other words, we are forced to actually search the polygon for reconstruction. The query will be made for edge e_0, and we modify e_0 to ensure that it is not in convex position with any subsequent edge. To construct an input in Σ_i, we will make some changes to this starting polygon. These changes will be made randomly, generating the distribution of polygons Σ_i. Figure 10 shows one such input.

We now describe how the edges are changed. The first $n/2$ edges will not be changed. In Figure 10, this is the group of edges A. Let us denote the remaining edges $e_0, e_1, \ldots, e_{n/2}$ in polygonal order. The interval of edges from $e_0, \ldots, e_{d^i}$ is where all the changes are made; these edges will be modified according to some distribution to generate Σ_i. First, the edges $e_0, \ldots, e_{d^{i-1}}$ are rendered "useless" - they are modified to ensure that these edges are not in convex position with any of the other edges. This is denoted by group B in Figure 10.

The remaining edges $(e_{d^{i-1}+1}, \ldots, e_{d^i})$ are divided into 2^i contiguous groups, each of length of $(d-1)d^{i-1}2^{-i}$. Each of these groups consists of the leaves of subtrees of size depth i. These are denoted by $C_1, C_2, \ldots$ in the figure. A $(1/d)$-fraction of these are chosen uniformly at random and are rendered "useless." In Figure 10, the group C_4 is

made useless. Therefore, groups of useless edges are interspered in the convex regions at random. The number of edges that need to be modified to make the input convex is $< d^i/c$.

The distribution Σ is generated by choosing $\log_{2c}(\sqrt{n}) \le i \le \log_{2c}(\delta n)$ uniformly at random and then choosing a random input from Σ_i. A query is made for e_0. Since the filter errors with probability $< 1/n$, for at least half of these values i, the filter errors on Σ_i with probability $< 2/n$. Consider any such Σ_i. The edge e_0 needs to be replaced, and edges used to choose this replacement must come from $e_1, \ldots, e_{d^i}$. If no edge from here was detected, then some edge outside this range is used. In that case, since we assume that we have no knowledge about the geometric structure of edges in this range, the replacement will essentially destroy all edges from $e_1, \ldots, e_{d^i}$. In other words, all these edges will need to be changed. This cannot happen, since that would mean that at least $c\varepsilon_{\mathcal{D}} n$ edges are changed. Therefore, a nonmodified edge from $e_1, \ldots, e_{d^i}$ must be chosen. This involves finding a group with unchanged edges. Given a group, the probability that it is unchanged is $\Theta(1/c)$. To ensure probability of error $< 2/n$, $\Omega(\log n)$ groups needs to be chosen (and an edge from group needs to be taken). Since each of these groups are in different subtrees of depth i, the total time to get these edges is $\Omega(\log n)$. Finally, there are $\Omega(\log n)$ distributions Σ_i on which the filter errors with probability $< 2/n$. Note that the replacement edges required for each distribution Σ_i is different (an edge which can be used for reconstruction for some input of Σ_i cannot be used for an input of any other Σ_j). This gives that the total number of edges that the filter needs to query is $\Omega(\log^3 n)$. $\quad\square$

Before giving a lower bound for filters reconstructing convexity when the input is given in linked list format, we show the optimality of the convexity tester (upto polylogarithmic factors in n).

THEOREM 2.19. *Testing convexity in 2 dimensions (in the DCEL model) requires* $\Omega(\varepsilon^{-1/2} n^{1/3})$ *lookups.*

PROOF. We will give a lower bound for the following problem: Given an input polygon in DCEL format which is ε-far from being convex, output a violation. We use Yao's minimax principle. We will consider a distribution on the input, and prove a lower bound on any deterministic algorithm that gives the correct answer with probability $2/3$ over the input.

Suppose some deterministic algorithm makes $\delta\varepsilon^{-1/2} n^{1/3}$ lookups (δ is a sufficiently small constant). The input is a linked list of all the edges. In our model (taken from Chazelle et al. [2003]), the list is accessed through a table $T[1 \cdots n]$, where the i-th element is stored in location $\sigma(i)$ - so $T[\sigma(i)] = i$. Consider the following polygon - $\mathcal{D}$ has $\varepsilon n^{2/3}$ stretches of length $n^{1/3}$. Each stretch violates $3n^{1/3}$ edges. Figure 11 shows a portion of $\mathcal{D}$ with the stretches in bold. $\mathcal{D}$ is constructed by stitching together many copies of this structure. The stretches occur far apart and therefore don't violate with each other.

The input distribution is formed by choosing the permutation σ uniformly at random from the symmetric group on n elements. Any deterministic algorithm can be seen as executing a sequence of steps of the form: (A) choose a location $T(l)$ and look up $T(\sigma(i \pm 1))$, where $l = \sigma(i)$ (this is equivalent to walking along the polygon); (B) compute a new index l based on previous indices and look up $T(l)$. We can see that $\sigma^{-1}(l)$ is equally likely to lie anywhere in the unvisited portion of $\mathcal{D}$.

For every stretch L, let X_L be the indicator variable for the event that a violation from L is detected by 2 B-steps. X_L is 1 iff an edge from L and an edge from the violations of L (having size $3n^{1/3}$) are chosen.
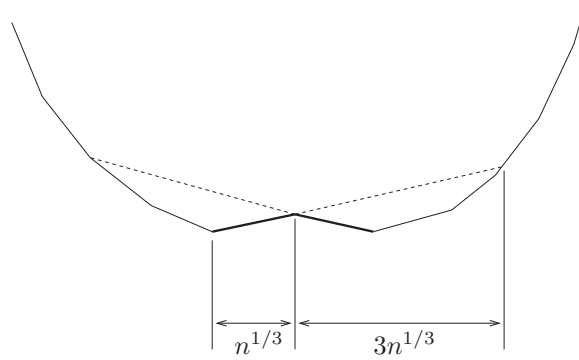
Fig. 11.   Testing lower bound.

$$\mathbf{E}\left[X_L\right] < \left[1 - \left(1 - \frac{4n^{1/3}}{n}\right)^{\delta\varepsilon^{-1/2}n^{1/3}}\right]^2 < \left[1 - e^{-8\delta\varepsilon^{1/2}n^{-1/3}}\right]^2 < \frac{64\delta^2}{\varepsilon n^{2/3}}$$

$$\mathbf{E}\left[\sum_L X_L\right] \le \varepsilon n^{2/3}\mathbf{E}\left[X_L\right] < 64\delta^2.$$

By Markov's inequality (and choosing a small enough δ), the probability that $\sum X_L$ exceeds 1 is less than 1/10. In the following, c_1 and c_2 denote some sufficiently large constant. Let Y_L be the indicator variable for the event that a B-step falls within $\varepsilon^{-1/2}n^{1/3}/c_1$ of either boundary of L.

$$\mathbf{E}\left[Y_L\right] < 1 - \left(1 - \frac{4n^{1/3}}{c_1\sqrt{\varepsilon n}}\right)^{\delta\varepsilon^{-1/2}n^{1/3}} < \frac{\delta/c_2}{\varepsilon n^{1/3}}$$

$$\mathbf{E}\left[\sum_L Y_L\right] \le \varepsilon n^{2/3}\mathbf{E}\left[Y_L\right] < (1/c_2)\delta n^{1/3}.$$

Again by Markov's inequality, the probability that $\sum Y_L$ exceeds $(1/10)\delta n^{1/3}$ is less than 1/10. With probability at least 8/10, B-steps by themselves did not detect a violation, and at most a 1/10 fraction of edges visited by B-steps fall within $\varepsilon^{-1/2}n^{1/3}/c_1$ of a stretch boundary. Assume that this happens. Any violation detected must involve an A-step. With probability at most 1/10, $\le \varepsilon^{-1/2}n^{1/3}/c_1$, A-steps were needed for detection. Since the total number of steps is $\delta\varepsilon^{-1/2}n^{1/3} < \varepsilon^{-1/2}n^{1/3}/c_1$, (a union bound shows that) the algorithm will fail with probability $> 7/10$.   □

We show that the filter for convexity (when the input is a linked list) is essentially optimal with respect to n (within $\log n$ factors).

THEOREM 2.20.   *Any filter for 2-dimensional convexity requires $\Omega(\sqrt{n})$ lookups per query, where the input is given in linked list format.*

PROOF. We prove a lower bound for the following problem - given $\mathcal{D}$ and an edge e, output whether e is good or bad. If e is deemed bad, then a violating edge must be provided. All good edges must be in convex position, and the number of edges deemed
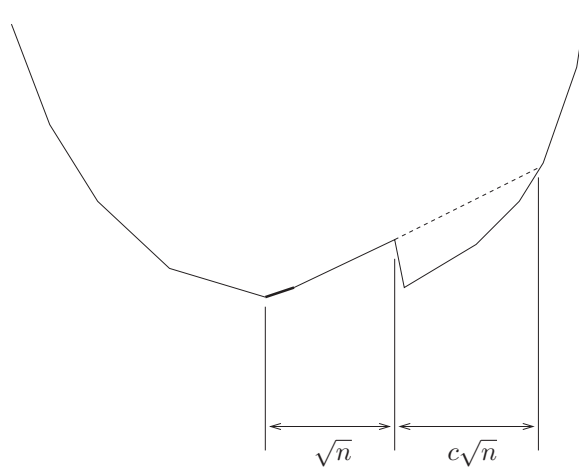
Fig. 12. Reconstruction lower bound.

bad must be $< c\varepsilon_{\mathcal{D}} n$ (for some constant c). We use Yao's minimax principle and proceed exactly as in the proof of Lemma 2.19. We prove a lower bound for a deterministic algorithm that gives the correct answer on a distribution of inputs with probability $> 2/3$. Consider a polygon that has distance εn (for some $\varepsilon > 0$) from convexity that has stretches of violating edges as shown in Figure 12. Essentially, the polygon is constructed by pasting together many stretches as given in the figure, and ensuring that the stretches do not violate each other. The downward pointing part that comes after the stretch of $\sqrt{n}$ is just one edge. It is obvious that the bold edge e is bad. The problem here is to find some edge in the $c\sqrt{n}$ range.

The distribution is formed by choosing the permutation σ at random (in the same model as that in the proof of Theorem 2.19). We are given a query for the edge e. Using an argument similar to the one for the testing lower bound, we can show that at least $\delta\sqrt{n}$ (for some constant δ) lookups are required to detect a violation. □

3. A CONVEXITY FILTER FOR 3D TERRAINS

The dataset is an n-face triangulated terrain $\mathcal{D}$ represented in standard DCEL fashion. We assume that the xy-projection of any face of $\mathcal{D}$ is a triangle with both edge lengths and angles bounded above and below by constants (bounded aspect ratio condition). The reconstructed terrain $\mathcal{D}^c$ is convex in the sense of being the boundary of the upper hull of its vertex set. There are various equally reasonable definitions of the parameter $\varepsilon_{\mathcal{D}}$. For simplicity, we define $\varepsilon_{\mathcal{D}} n$ as the minimum number of faces of $\mathcal{D}$ that need to be removed in order to make the terrain convex. Note that this definition does not require us to "patch the holes." Choosing to do so, however, would only increase the distance by a constant factor, which, for the purpose of our filter, is immaterial. The edge table allows us to sample random edges. From this, it is elementary to implement a uniform sampler for triangular faces as well. (To sample vertices would be more difficult but, fortunately, we do not need that feature.)

The filter processes the terrain during the first query in sublinear time and then uses the resulting data structure to answer subsequent queries. The (sublinear) cost of the processing is charged entirely to the first query. The idea is to break up the terrain into connected patches of suitable size by removing a small set $\mathcal{F}$ of separating faces. The fence $\mathcal{F}$ decomposes the terrain into connected patches of suitable sizes. A critical feature of $\mathcal{F}$ is to be of sublinear size. To achieve this, we use a sublinear version

---

OFFLINE-RECONSTRUCTION

Initialization: $T \leftarrow \emptyset$
  for each face f of $\mathcal{D}$:
    if f is in convex position with T
      then $T \leftarrow T \cup \{f\}$
      else find face $g \in T$ not in
          convex position with f
           $T \leftarrow T \setminus \{g\}$
    output $\mathcal{D}^c \leftarrow T$

---

of the classical planar separator theorem. The weakening is required to make the computation sublinear. The final processing step is to convexify $\mathcal{F}$. This is a delicate operation which cannot be performed in isolation with the rest of the terrain: this is a perfect illustration of why early decisions are crucial in online filtering.

We define a suitable range space (of unbounded VC dimension!) whose sampling gives us enough global information about the whole terrain to guide the reconstruction of $\mathcal{F}$. The convexified $\mathcal{F}$ fences off the patches in such a way that it is possible from then on to answer any subsequent query by treating its associated patch in isolation from the rest of the terrain. But, before we can get to online reconstruction, we need to define two key procedures: one is an offline algorithm for convexifying the terrain within twice the minimum distance; the other estimates the distance $\varepsilon_{\mathcal{D}}$ in sublinear time.

Any filter must explore both global and local properties. The difficulty lies in gathering enough information in sublinear time. Any approach must involve a combination of sampling and local search. The filter essentially works as follows: first, it estimates $\varepsilon_{\mathcal{D}}$ using the sublinear time procedure. If the distance is very small, then the offline algorithm is used for convexification. Otherwise, it constructs a fence $\mathcal{F}$ and convexifies it. It is critical that this convexification be done by taking the global structure into account—this is achieved by choosing a large enough sample of faces of $\mathcal{D}$ (which then is used to define the range space mentioned in the previous paragraph) and convexifing $\mathcal{F}$ so that it is in convex position with most of the sample. This creates a "skeleton" that captures the global properties of $\mathcal{D}$ and also splits $\mathcal{D}$ into a set of small connected patches. Next, each patch is reconstructed independently, ensuring that it stays in convex position with the convexified $\mathcal{F}$. Since a patch is a small connected portion of $\mathcal{D}$, it can be visited exhaustively by local search (thereby, the filter gains information about the local properties of $\mathcal{D}$). In the following sections, we discuss the various components that constitute the filter. Finally, we put the pieces together and describe the filter itself.

3.1. Offline Reconstruction

We describe a 2-approximation offline convexification algorithm, that is, one that, given $\mathcal{D}$ as input, returns a terrain $\mathcal{D}^c$ that is convex and is at distance at most $2\varepsilon_{\mathcal{D}}n$ from it. Note that $\mathcal{D}^c$ can have holes. The convexification proceeds incrementally. Beginning with the empty terrain T, we consider each face of $\mathcal{D}$ one by one and add it to T if it is in convex position[4] with every face currently in $\mathcal{D}^c$. To do this in quasilinear time, we maintain T in a dynamic data structure which supports insertion, deletion, and queries in amortized polylogarithmic time.

---

[4] Two triangles are said to be in convex position if both of them are faces of their convex hull.

Denote by T_v (respectively T_p) the set of vertices (respectively face-supporting planes) in T. A terrain face f is in convex position with T if and only if (i) its three vertices lie below each plane in T_p; and (ii) the points of T_v all lie below the plane supporting f. By duality, both tests can be reduced to *dynamic halfspace emptiness* in 3D: maintain a set of points under insertion and deletion, and for any query plane find whether whether all points lie on one side, and if they do not, report one point on each side. Chan [2006] has given a halfspace range reporting algorithm which allows us to do that in $O(\log^6 n)$ amortized time for each query/insert/delete.

Whenever the procedure finds a face f that is not in convex position with T, then a face $g \in T$ that is not in convex position with f is removed. Consider a minimal subset U of $\varepsilon_{\mathcal{D}} n$ faces that need to be removed to make $\mathcal{D}$ convex. One of f or g has to be present in U. This ensures that the total number of faces removes is at most $2\varepsilon_{\mathcal{D}} n$.

THEOREM 3.1. *Offline convex reconstruction of an n-face terrain can be performed with an approximation factor of 2 in $\widetilde{O}(n)$ time.*

3.2. Estimating the Distance to Convexity

Consider the violation graph G whose nodes are the faces of $\mathcal{D}$ and whose edges join any two faces not in convex position. Removing all the faces that correspond to a vertex cover of this graph makes $\mathcal{D}$ convex. The minimum vertex cover is of size $\varepsilon_{\mathcal{D}} n$, and any maximal matching M in G is of size $|M| \leq \varepsilon_{\mathcal{D}} n \leq 2|M|$. The offline reconstruction algorithm essentially finds such a maximal matching in G. Fix any constants $0 < \alpha < \beta < 1$ such that $\alpha \geq \frac{1}{2}(3\beta - 1)$, $\alpha \geq (2\beta - 1)$, and let S be a random sample formed by picking each vertex of G independently with probability $p = n^{\alpha - \beta} \log n$. The offline reconstruction algorithm can be used to build a maximal matching M_S for the subgraph G_S of G induced by S. The sample S can be easily specified in $O(|S|)$ time, so that computing M_S takes $\widetilde{O}(pn)$ time, which is $\widetilde{O}(n^{1+\alpha-\beta})$. As we show the following, in knowing M_S allows us to distinguish between the two cases: $\varepsilon_{\mathcal{D}} \geq n^{-\alpha}$ and $\varepsilon_{\mathcal{D}} \leq n^{-\beta}$.

Case 1 $\varepsilon_{\mathcal{D}} \geq n^{-\alpha}$. Fix some maximal matching M of G. The size of M is $\geq n^{1-\alpha}/2$. If ξ is the number of edges of M in G_S, then $\mathbf{E}\,\xi = p^2|M|$. Since M is a matching, ξ can be expressed as the sum of independent random variables. By Chernoff's bound [Alon and Spencer 2000], $\text{Prob}\,[\,\xi < \frac{1}{2}p^2|M|\,] < e^{-\Omega(p^2|M|)} = e^{-\Omega(p^2 n^{1-\alpha})} = e^{-\Omega(n^{1+\alpha-2\beta}\log^2 n)}$. Since $\alpha \geq 2\beta - 1$, with high probability $\xi \geq \frac{1}{2}p^2|M|$. This implies that G_S contains a perfect matching of size at least $\frac{1}{2}p^2|M| \geq \frac{1}{4}p^2 n^{1-\alpha}$. Its minimum vertex cover is at least that size; therefore, $|M_S| \geq \frac{1}{8}p^2 n^{1-\alpha}$.

Case 2 $\varepsilon_{\mathcal{D}} \leq n^{-\beta}$. Now, the size of M is $\leq n^{1-\beta}$. If χ denotes the number of vertices of M in S, then $\mathbf{E}\,\chi = 2p|M|$ and, by Chernoff's bound, $\text{Prob}\,[\,\chi \geq 2p|M| + y\,] < e^{-y^2/|M|}$, for any $y > 0$. Setting $y = \frac{1}{8}p^2 n^{1-\alpha} - 2p|M| > \frac{1}{9}p^2 n^{1-\alpha}$, we find that $\text{Prob}\,[\,\chi \geq \frac{1}{8}p^2 n^{1-\alpha}\,] < e^{-\Omega(p^4 n^{1+\beta-2\alpha})} = e^{-\Omega(n^{1+2\alpha-3\beta}\log^4 n)}$. Since the vertices of M provide a vertex cover for G, it follows that, with high probability, $|M_S| < \frac{1}{8}p^2 n^{1-\alpha}$.

THEOREM 3.2. *Given any small $\delta > 0$ and any constants $0 < \alpha < \beta < 1$ such that $\alpha \geq \frac{1}{2}(3\beta - 1)$ and $\alpha \geq (2\beta - 1)$, we can compute a 0/1 bit $b(\mathcal{D})$ in $\widetilde{O}(n^{1+\alpha-\beta})$ time, such that $b(\mathcal{D}) = 0$ if $\varepsilon_{\mathcal{D}} \geq n^{-\alpha}$, $b(\mathcal{D}) = 1$ if $\varepsilon_{\mathcal{D}} \leq n^{-\beta}$, and $b(\mathcal{D})$ takes on any value otherwise.*

3.3. Fencing off the Terrain

Here we describe an algorithm that finds a sublinear set of faces (the fence) of $\mathcal{D}$ whose removal breaks $\mathcal{D}$ into connected components (patches) also of sublinear size. Refer to Figure 13. The thick black line represents the fence, which breaks the terrain into small patches. As we discussed earlier, this procedure will be run in the first query. Let
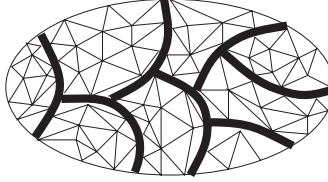
Fig. 13.   The thick black line, the fence, is itself a collection of $o(n)$ triangles.

G be the planar triangulation formed by the projection of $\mathcal{D}$ onto the xy-plane. Note the because of the bounded aspect ratio condition, any triangle in G has bounded sides and angles. The main step in constructing the fence is to design a sublinear algorithm to find balanced planar separators in such graphs. A *balanced planar separator* for a planar graph G with n vertices is a set of vertices whose removal separates G into connected components, each having size $< cn$ (where c is some constant less than 1). Lipton and Tarjan [1979] gave the first linear time algorithm to find a balanced planar separator of size $O(\sqrt{n})$. Henceforth, by planar seperator, we always refer to balanced separators. Our aim is to find a set of *faces* in G to remove (note that upto constant factors, this gives a vertex separator of the same size). We are able to beat the linear time bound because we are provided with a geometric embedding of the graph. We also assume the bounded aspect ratio condition.

The algorithm first randomly selects a set of $c\sqrt{n}\log n$ faces, for some sufficiently large constant c. This sample (call it R) is then used to guide the separator. A random starting vertex v in G is chosen, and then R is used to find geometric paths (paths in the plane) from v to the boundary of G that pass through at most $O(\sqrt{n})$ faces. These paths can then be shown to generate a planar separator of $O(\sqrt{n})$ size. We begin by stating the main lemma of this section (this lemma is of independent interest).

LEMMA 3.3. *For any vertex $v \in G$ (where G is a bounded aspect ratio planar graph with an embedding), there exists a geometric path from v to the boundary of G that passes through $O(\sqrt{n})$ faces. Furthermore, this path can be shown to be x and y-monotone, consisting only of vertical or horizontal line segments. This path can be constructed in $\widetilde{O}(\sqrt{n})$ time.*

We will need to prove some smaller claims before we can attack this lemma. Consider the area defined by a horizontal line segment, and two rays pointing in the positive y-direction. This is called a *vertical slab*. We will only consider line segments that have $\Theta(1)$ length. Now, we assign a charge to each triangle t (face) of G, and spread the charge out evenly in the area of t. The total amount of charge throughout G is n. The charge contained in a slab S can be determined by the following fact - for a triangle t, if an f-fraction of it (area-wise) lies inside S, then t contributes f amount of charge.

CLAIM 3.4.   *A line segment ℓ of constant size can only intersect a constant number of triangles.*

PROOF.   First consider two triangles $\triangle ABC$ and $\triangle BCD$ and suppose that ℓ intersects AB, BC, CD. Let ℓ intersect AB at E and BC at F. Refer to Figure 14. Wlog, assume that $BF > FC$. Take $\triangle BEF$. If $BE > BF/2$, by the bounded angle condition, we get the EF is at least a constant. On other hand, if $BE \leq BF/2$, then by triangle inequality, we get that $EF \geq BF/2$ and is therefore a constant. The portion of ℓ inside $\triangle ABC$ and $\triangle BCD$ has length $\Theta(1)$. Now, take all triangles that intersect ℓ (let us not consider the triangles that contain the endpoints of ℓ; there are only two of them). Take the set of triangles that all share one endpoint and put them in the order in which they intersect ℓ - there can only be a constant number of them, because the angles are bounded. Note
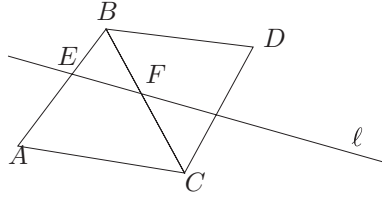
Fig. 14.   Wedge.

that the last two triangles in this sequence have the structure of $\triangle ABC$ and $\triangle BCD$, where ℓ intersects AB, BC, CD. This shows that there can only be constant number of such sets of triangles, completing the proof of the claim.   □

CLAIM 3.5.   *If a slab intersects k triangles, then it contains $\Omega(k)$ amount of charge.*

PROOF. Any triangle that is completely inside the slab S contributes one unit charge (all the charge it contains). The main problem is to deal with triangles that intersect the boundary of S. Let us denote by ℓ the line segment (assume its horizontal) defining S, and let r_1 and r_2 be the rays. Abusing notation, ℓ also denotes the length of this segment.

By Claim 3.4, only a constant number of triangles can intersect ℓ. We shall now only consider triangles that intersect r_1 or r_2 – if a triangle does not intersect any edge of the boundary of S, then it contributes one unit of charge to S. First consider a triangle t with no vertex inside S. Two edges of t intersect both r_1 and r_2. Because the slab has constant width, wlog, the portion of r_2 between these edges is of constant length. This implies that t contributes $\Omega(1)$ units of charge to S.

Take a triangle t which has a vertex v inside S and whose edges intersect both r_1 and r_2 (call this a triangle of Type 1). Consider the triangle with vertex v and the intersection points of the edge opposite to v with r_1 and r_2. This triangle has constant area and again t contributes $\Omega(1)$ units of charge. Now take a triangle t with a vertex v inside S which only intersects some r_i (i is either 1 or 2). Furthermore, assume that the perpendicular distance of v from r_i is $\geq \ell/2$. This is a triangle of Type 2. Such a triangle can be easily seen to contribute $\Omega(1)$ units of charge to S. Finally, we take the last case: triangle t has vertex v inside S, its edges only intereesect some r_i, and the distance of v to r_i is $< \ell/2$. But, there must be some triangle having v as a vertex which is of type 1 or 2. Since the number of triangles incident to one vertex is $\Theta(1)$, this implies that only a constant fraction of k triangles can be of the final kind. This completes the proof of the claim.   □

We now complete the proof of Lemma 3.3.

PROOF. We first describe the algorithm used to find such a path from a starting vertex v. A random sample of faces R, of size $c\sqrt{n}\log n$ (where c is a sufficiently large constant) is chosen. It is easy to show that if some slab intersects $> c\log n$ triangles in R, then the slab intersects $> c_1\sqrt{n}$ triangles in G. On the other hand, if the slab intersects $\leq c\log n$ triangles in R, then it intersects $< c_2\sqrt{n}$ triangles in G (c_1, c_2 are some fixed constants, and these hold with high probability).

We draw a constant length horizontal segment ℓ from v in the positive x-direction. By Claim 3.4, ℓ intersects at most a constant number of triangles. Suppose the vertical slab defined by ℓ intersects $\leq c\log n$ triangles of R. Then, we draw a vertical line (going in the positive y-direction) from ℓ to the boundary (thereby completing the path). If the vertical slab intersects $> c\log n$ triangles of R, then we move in the horizontal direction by drawing another constant length horizontal segment from the right endpoint of ℓ. If the horizontal segments eventually hit the boundary of G, then the path is complete.

The path finally obtained has size $O(\sqrt{n})$. Note that the vertical part of the path can intersect at most $O(\sqrt{n})$ triangles. Each constant length horizontal segment (except for probably the rightmost one) defines a vertical slab intersecting $\Omega(\sqrt{n})$ triangles. By Claim 3.5 the amount of charge in this slab is also $\Omega(\sqrt{n})$. All these slabs are disjoint and the total amount of charge overall is n. There can only be $O(\sqrt{n})$ such horizontal segments, and the total number of triangles intersected by these is $O(\sqrt{n})$. □

This brings us to the final theorem of this section, where we show how a sublinear-sized fence that leaves patches of sublinear size can be constructed in sublinear time.

THEOREM 3.6. *There exists a* $\widetilde{O}(\sqrt{n})$ *algorithm that finds a balanced planar separator of size* $O(\sqrt{n})$ *in any triangulated bounded aspect ratio planar graph G which is provided as a DCEL (G is given by straight line embedding) with the planar coordinates of the vertices. For any* $0 < a < 1$*, this can be used to find a fence of size* $O(n^{1-a/2})$ *in* $\widetilde{O}(n^{1-a/2})$ *time such that the patches have size* $O(n^a)$*.*

PROOF. First, we describe how to get a planar separator. Choose a vertex v at random. With probability $\geq 3/5$, the x-coordinate of v is the middle $(3/5)n$ positions in the sorted order of vertices based on x-coordinate. Similarly, this holds for y-coordinates. Therefore, with constant probability, v is the middle $(3/5)n$ positions for both x-sorted and y-sorted lists of vertices. Without loss of generality, there are at least $n/10$ vertices with *both x* and y-coordinates larger than those of v and at least $n/10$ vertices with both x and y-coordinates less than those of v. By choosing $O(\log n)$ vertices uniformly at random, we can ensure (with high probability) that we find at least one vertex that satisfies this property.
Lemma 3.3 tells us that there is a geometric path intersecting $O(\sqrt{n})$ triangles that starts from v and ends at the boundary of G. Also, this path (when directed from v) only increases in the x-direction and decreases in the y-direction. Similarly, there is a path starting from v that only decreases in the x-direction and increases in the y-direction. These paths can be found in $\widetilde{O}(\sqrt{n})$ time. Together, these paths form a separator of G. This is also balanced, since no component can have size larger than $9n/10$.
Recursive application of this procedure yields a fence of size $O(n^{1-a/2})$ wth patches of size $O(n^a)$. The total running time is $O(n^{1-a/2} \log n)$. □

3.3.1. Finding Separators in General Terrains. As an aside, we describe an algorithm that finds a fence for a general planar graph $\mathcal{G}$ (not nessarily embeddable with bounded aspect ratio). Pick a random sample of $r = n^a$ edges in G, for fixed $0 < a < 1$, and build its (say, x-oriented) trapezoidal map $\mathcal{M}_r$. As is well known, with high probability, each trapezoid intersects $O((n/r) \log n)$ triangles. Consider the dual $\mathcal{H}_r$ graph of $\mathcal{M}_r$, where each node is a trapezoid and two nodes are joined if the corresponding (closed) trapezoids intersect. The graph is planar and so, by iterated application of the planar separator theorem [Lipton and Tarjan 1979], for any fixed $0 < b < 1$, we can find, in $O(r \log r)$ time, a set V of $O(r^{1-b/2})$ nodes whose removal leaves $\mathcal{H}_r$ with no connected component of size exceeding r^b. The fence $\mathcal{F}$ is the set of all the terrain's faces whose projections intersect the trapezoids associated with the nodes of V. With high probability, the fence consists of $O(r^{1-b/2}(n/r) \log n)) = O(n^{1-ab/2} \log n)$ triangles and its removal from the terrain leaves connected patches, each one consisting of $O(n^{1+ab-a} \log n)$ triangles. Note that, because it involves triangles (and not trapezoids), the removal may create much greater fragmentation than is caused within $\mathcal{H}_r$ by the removal of V. Finding the fence takes time $O(n^a \log n + n^{1-ab/2} \log n)$ time, Renaming ab by b, we have proven the following.

LEMMA 3.7. *Let G be any planar graph provided with a geometric embedding. For any* $0 < b < a < 1$*, in* $O((n^a + n^{1-b/2}) \log n)$ *time, it is possible to find a fence* $\mathcal{F}$ *consisting*
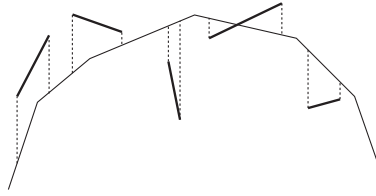
Fig. 15.   The fence is reconstructed by lifting its faces to the upper boundary of Σ^c.

of $O(n^{1-b/2} \log n)$ triangles, whose removal from the terrain leaves connected patches consisting of $O(n^{1+b-a} \log n)$ triangles each.

3.4. Reconstructing the Fence

Our aim is to first reconstruct the fence in sublinear time, and then use this structure to reconstruct the various patches. Since the fence has sublinear size, we can hope to perform the fence reconstruction quickly. Given a reconstructed fence, we will (later) argue that the patches can be reconstructed in sublinear time. But how to convexify the fence? A naive approach would be to simply apply the offline algorithm for the fence triangles. Since the offline algorithm is essentially linear (in the fence size), the total running time for this would be sublinear. But that would fail miserably, since it does not allow the global shape of the terrain to influence the reconstruction. We need a different approach.

We choose a random sample Σ of the faces of $\mathcal{D}$—the size of Σ shall be set later. Let Σ^c be the offline convexification of the terrain Σ provided by Theorem 3.1, and let Σ^f the intersection of the halfspaces bounded above by the planes supporting the faces of Σ^c (the dual convex hull). The reconstructed fence $\mathcal{F}^c$ is obtained by lifting the triangles of $\mathcal{F}$ vertically and "wrapping" them over the surface of Σ^f (refer to Figure 15). Note that such a lifting could replace one fence face by many faces, but based on the bounded aspect ratio condition, we have a bound (proven later in this section) for the total size of $\mathcal{F}^c$. (The bound holds only when Σ is sufficiently large, but our choice of Σ will ensure that.)

LEMMA 3.8.   *The size of $\mathcal{F}^c$ is $\widetilde{O}(n^{1-a/4})$.*

We now explain why $\mathcal{F}^c$ captures the global structure of $\mathcal{D}$. Henceforth, the term "face" refers to a triangle in 3-dimensions (we introduce this because we later use "triangle" to refer to an object in the 2-dimensional plane). We define a range space $(X, \mathcal{R})$, which, although of unbounded VC dimension, has enough sampling power to guide the convexification of the fence. Regarding both $\mathcal{D}$ and $\mathcal{F}$ as sets of faces, we define the ground set $X = \mathcal{D} \setminus \mathcal{F}$. Given two sets S, T of faces, let $\kappa(S, T)$ be the set of faces in T that are not in convex position with at least one face of S. Considering all possible sets Γ of $|\mathcal{F}^c|$ faces, we define $\mathcal{R} = \{ \kappa(\Gamma, X) : |\Gamma| = |\mathcal{F}^c| \}$.

Let us represent a face f by 12 reals - 9 for the vertices of the face, and 3 for the point in dual space which corresponds to the plane containing f. The face f can be seen as a point $p_f \in \mathbb{R}^{12}$. (Note that the latter 3 reals are completely determined by the former 9 reals. This is a redundant description and actually not necessary for what follows, but it will be helpful.) Consider faces $f \in \Gamma$ and $x \in X$. The convex position of f with respect to x is completely determined by - the position of the vertices of f with respect to the plane containing x, and (in dual space) the position of f, which is a point, with respect to the three planes corresponding to the vertices of x. Let us move to $\mathbb{R}^{12}$. There is an arrangement of 4 hyperplanes in this space, such that the convex position x with respect to f is completely determined by the position of p_f in this arrangement.
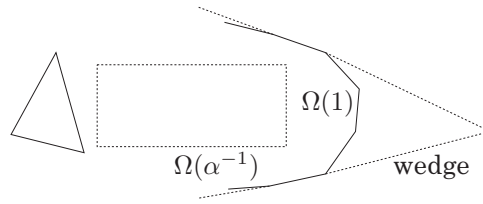
Fig. 16.   Wedge.

By taking the necessary planes for all $x \in X$, we get an arrangement of $O(n)$ planes which completely determine the convex position of f with respect to every face in X.

We can view Γ as a point in $p_\Gamma \in \mathbb{R}^{12|\mathcal{F}^c|}$. Using the construction given above for every face in Γ, it is easy to see why the convex position status of each face in Γ with respect to X is completely specified by the location of p_Γ in a certain $12|\mathcal{F}^c|$-dimensional arrangement of $O(n|\mathcal{F}^c|)$ hyperplanes - we have a set of $O(n)$ hyperplanes for every face in Γ. It follows that the primal shatter function φ grows as $\varphi(m) = O(m|\mathcal{F}^c|)^{12|\mathcal{F}^c|}$. With high probability, if Σ is chosen to be a random sample of X of size $O(r^2|\mathcal{F}^c|\log|\mathcal{F}^c|) = \widetilde{O}(r^2 n^{1-a/4})$, it is a $(1/r)$-approximation for $(X, \mathcal{R})$, for any $r > 0$.

By Theorem 3.1, the number of triangles in Σ that were modified during the convexification is at most $2\varepsilon_\Sigma|\Sigma|$. Using an argument similar to the proof of Theorem 3.2, we can show that, with high probability, this number is $O(\varepsilon_\mathcal{D}|\Sigma|)$. This implies that $|\kappa(\mathcal{F}^c, \Sigma)| = O(\varepsilon_\mathcal{D}|\Sigma|)$. Since

$$\left| \frac{|\kappa(\mathcal{F}^c, \Sigma)|}{|\Sigma|} - \frac{|\kappa(\mathcal{F}^c, X)|}{|X|} \right| \leq \frac{1}{r},$$

we could easily bound the "damage" caused by the convexification of the fence as follows:

LEMMA 3.9. $\kappa(\mathcal{F}^c, X) \leq nr^{-1} + O(\varepsilon_\mathcal{D} n)$.

We now prove Lemma 3.8. For ease of notation, we shall assume that $\mathcal{F}$ has $O(n^u)$ faces and Σ has size $\widetilde{O}(n^v)$. We will be concerned only with xy-projections of $\mathcal{D}$ and Σ^f. In the following proof, *triangle* refers to the xy-projection of a face of $\mathcal{D}$, while *facet* refers to the xy-projection of a face of Σ^f. *Edges* refer to the edges of triangles. Note that all facets are convex, disjoint from each other (except for their boundaries) and contain a triangle. By the bounded aspect ratio assumption, the radius lengths of the incircle and circumcircle of any triangle are bounded from above and below (by some constant). Let v' be some value less than v.

Definition 3.10.   For $\alpha < n^{-(1-v')}$, an α-*thin facet* is a facet with two edges e_1, e_2 such that the minimum distance between e_1 and e_2 is less than α and the angle between e_1 and e_2 is also less than α. The edges e_1 and e_2 are called *sharp* edges. The *min-thinness* of a facet f is the minimum α such that f is α-thin.

The sharp edges of an α-thin facet form a *wedge* that contains the facet (Figure 16).

CLAIM 3.11.   *With high probability, there are at most $O(\alpha n)$ α-thin facets.*

PROOF. Consider some α-thin facet f which contains triangle t. The distance between t and the sharp edges is at least $\Omega(\alpha^{-1})$ (since t must be inside the wedge created by these sharp edges, and has at least constant in-radius). There exists a rectangle of $\Omega(\alpha^{-1})$ width and $\Omega(1)$ height that is present completely inside f but does not intersect t (Figure 16). Therefore, we can also show that there exist at least $\Omega(\alpha^{-1})$ edges of $\mathcal{D}$ that have at least a constant fraction of their length inside f (let these edges be *bad* for f). Note that the offline convexification must have removed these bad edges. Let
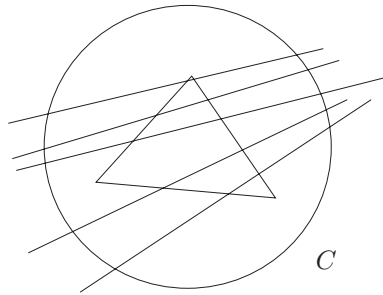
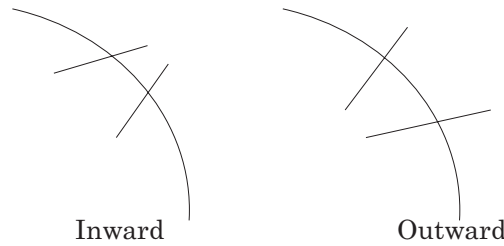Fig. 17. Facets intersecting with C.

Inward   Outward

Fig. 18. Inward and outward.

the number of α-thin facets be M. Since all facets are disjoint, an edge can be bad for only a constant number of α-thin facets. The total number of bad edges is $\Omega(\alpha^{-1}M)$. With high probability (by taking a Chernoff bound), at least $(c\alpha^{-1}Mn^{v-1}\log n)$ of these edges are chosen in Σ (for some constant c). But this quantity has to be $O(n^v \log n)$, since that is the number of edges removed by convexification. This implies that the number of α-thin facets is $O(\alpha n)$. $\quad\square$

CLAIM 3.12. *For any $v' < v$, the total complexity of the lifted fence is $O(n^{1+u-v'} + n^{v'} \log n)$.*

PROOF. The complexity of lifting a fence face is simply the complexity of the corresponding triangle when laid over the xy-projection of Σ^f. A triangle is said to *generate* all the faces created by overlaying. The total complexity of all triangles generating $O(n^{1-v'})$ faces is $O(n^{1+u-v'})$ (since there are at most $O(n^u)$ fence triangles). Consider some triangle t generating $k > n^{1-v'}$ faces. Each of these faces is created by the intersection of t with a facet.

We will first show that many of these facets are k^{-1}-thin. Since triangles do not intersect, no facet can be completely contained inside t. At least $\Omega(k)$ facets intersect some edge e of t. We take a circle C of constant (but large enough) radius such that C contains e and the minimum distance between e and C is $\Omega(1)$. The intuition for the following proof is quite simple—since C is a constant sized circle and many facets intersect it, many facets have to be thin (Figure 17). Since the area of a facet is $\Omega(1)$, there can be at most a constant number of facets contained completely inside C. Therefore, at least $\Omega(k)$ facets intersect both e and C. A facet can intersect C in two ways - inward and outward (Figure 18). Consider a facet f (containing triangle t') that intersects C only in the inward direction. Either more than half of t' is contained in C or the arc length of some intersection between C and f is $\Omega(1)$. Only a constant number of facets can have only inward intersection. Therefore, $\Omega(k)$ facets intersect C outwards, and (by a Markov argument) $\Omega(k)$ of these intersections have arc length $< k^{-1}$. Consider

---

3D-TERRAIN-FILTER

```
if first query
then if b(𝒟) = 1
      then convexify 𝒟 using
        OFFLINE-RECONSTRUCTION
      else build fence ℱ and convexify
        into ℱᶜ and go to (1)
else
 (1) identify patch containing query face f
     if needed, convexify extended patch
     with OFFLINE-RECONSTRUCTION
```

---

any such facet f' - the two edges intersecting C has a minimum distance of less than k^{-1}. Since f' intersects e, the angle between the edges must be $O(k^{-1})$, and f' is $O(k^{-1})$-thin. This shows that for any triangle t generating k faces, $\Omega(k)$ of these faces are $O(k^{-1})$-thin facets. We will say that t is a *witness* for these $O(k^{-1})$-thin facets, since the intersection of C with these facets shows their $O(k^{-1})$-thinness. For any facet f of min-thinness α, note that at most $(\alpha n^{1-v'})^{-1}$ triangles are witnesses for *any* thinness (since the minimum distance between sharp edges is $< n^{-(1-v')}$ and the angle is α.)

We sum up the constrisions (in terms of complexity) of all triangles generating more that $n^{1-v'}$ faces. Let this sum be S. By the arguments given above, βS comes from thin facets that are witnessed (for some fixed constant $\beta < 1$). Some facets are counted more than once in βS because many triangles can witness one facet. Let us consider all witnessed facets with min-thinness in the range $[\alpha/2, \alpha]$. There are at most $O(\alpha n)$ such facets and each is counted in S at most $2(\alpha n^{1-v'})^{-1}$ times. Therefore, the total contribution of this in S is $O(n^{v'})$. We apply this argument for the values $\alpha = n^{-(1-v')}, 2^{-1}n^{-(1-v')}, 2^{-2}n^{-(1-v')}, \ldots, (cn)^{-1}$ (for some sufficiently large constant c) and get that $S = O(n^{v'} \log n)$. □

Noting that $u = 1-a/2$ and ensuring $v > 1-a/4$, we can set $v' = (1+u)/2$. By the previous claim, the complexity of the lifted fence can be made $\widetilde{O}(n^{1-a/4})$, proving Lemma 3.8.

3.5. Online Reconstruction

We now have all the necessary tools required to make the filter. As we mentioned earlier, the first query is the occasion of some preliminary processing whose cost is entirely charged to the query itself. By Theorem 3.2, we estimate the distance to convexity in $\widetilde{O}(n^{1+\alpha-\beta})$ time. If $b(\mathcal{D}) = 1$, then we convexify the terrain in $\widetilde{O}(n)$ time by appealing to Theorem 3.1. Since $\varepsilon_\mathcal{D} < n^{-\alpha}$, the running time is $\widetilde{O}(\varepsilon_\mathcal{D}^{-\alpha^{-1}})$.

Assume now that $b(\mathcal{D}) = 0$, which implies that $\varepsilon_\mathcal{D} > n^{-\beta}$. By Lemma 3.9, setting $r = n^\beta$ shows that $\kappa(\mathcal{F}^c, X) \leq O(\varepsilon_\mathcal{D} n)$. A crucial aspect of the reconstructed fence is that the convexification of any patch can be done in isolation, as long as we include the fence triangles bounding the patch in question. This follows from this transitivity lemma, which we prove later. (We use the subscript xy to denote the projection onto the xy-plane.)

LEMMA 3.13.   *Let f, g be two faces of a (possibly discontinuous) terrain and let F, G be two sets of faces in convex position such that: (i) removing the region F_{xy} disconnects f_{xy} from g_{xy}; same is true of G_{xy}. If f (respectively g) is in convex position with F (respectively G), then f and g are in convex position with each other.*

Given a query f, unless $b(\mathcal{D}) = 1$, the filter finds the patch corresponding to f. If it is the first access to the patch, then it proceeds to reconstruct the entire patch together with all its bordering fence triangles (what we call the *extended patch*). Otherwise, it simply outputs the corresponding face in the reconstructed patch. By Theorem 3.6, computing the fence takes $\widetilde{O}(n^{1-a/2})$ time. By our setting of $r = n^\beta$, to find Σ requires $\widetilde{O}(r^2 n^{1-a/4}) = \widetilde{O}(n^{1+2\beta-a/4})$ time, and convexification can be done in time $\widetilde{O}(n^{1+2\beta-a/4})$. Reconstructing the fence adds nothing to the asymptotic complexity. Any query that involves convexifying the corresponding patch takes $\widetilde{O}(n^a)$. Putting everything together, we see that in the worst case the time for answering any query is

$$\widetilde{O}(n^{1+\alpha-\beta} + \varepsilon_{\mathcal{D}}^{-\alpha^{-1}} + n^{1-a/2} + n^{1+2\beta-a/4} + n^a).$$

The constraints $0 < a < 1$, $0 < \alpha < \beta < 1$, and $\alpha \geq \frac{1}{2}(3\beta - 1)$ are all satisfied if we set α to be an arbitrarily small positive constant and $a = 12/13$ and $\beta = 1/13$. It is immediate that the amortized query time is $O(n^\alpha)$, proving the following theorem.

THEOREM 3.14. *Any n-face 3D bounded aspect ratio terrain $\mathcal{D}$ has a convexity filter with a worst case query time of $O(n^{12/13+\alpha} + \varepsilon_{\mathcal{D}}^{-O(\alpha^{-1})})$ and an amortized time of $O(n^\alpha)$, for an arbitrarily small $\alpha > 0$.*

We now prove Lemma 3.13. We will prove the following claim, from which Lemma 3.13 will be obvious.

CLAIM 3.15. *Let f, g be two faces of a possibly discontinuous terrain and S be a set of faces in convex position. If removing S_{xy} disconnects f_{xy} from g_{xy} and f and g are in convex position with S, then f and g are in convex position with each other.*

PROOF. Let $\mathcal{B}$ be the (possibly unbounded) convex body formed by the faces of S. For simplicity, assume that S is minimal - we cannot remove any face from S and still maintain the lemma assumptions. Take the region in the xy-plane disconnected by the removal of S_{xy}. Project the boundary of this region onto $\mathcal{B}$ and call this curve C^S - note that the curve C^S_{xy} separates f_{xy} from g_{xy}. By the minimality assumption, the curve C^S is simple. Without loss of generality, let f_{xy} be contained in the inner region defined by C^S_{xy}. Let H^f be the plane containing f and C^f be the intersection curve of H^f and $\mathcal{B}$. Note that C^S lies completely to one side of H^f. Suppose there is a vertex of g that lies in the halfspace (defined by H^f) that does not contain C^S. Note that g lie inside $\mathcal{B}$, since it is in convex position with S. Therefore, it must be the case that the xy-projection of this vertex lies inside C^f, which lies inside C^S. This is contradicts the fact that C^S_{xy} separates f_{xy} from g_{xy}. The face g lies completely to one side of H^f.

Defining H^g and C^g (which may not be closed), we can apply a similar argument to show that f lies completely on one side of H^g. $\quad\square$

3.6. Lower Bound

We show that any 3D convexity filter that works for arbitrary terrains has a worst case query time of $\Omega(\varepsilon_{\mathcal{D}}^{-1})$ time, thus revealing a fundamental complexity gap between the two and three-dimensional cases. Recall that the 2D filter made essential use of a certain transitivity feature of convexity violation: if e, f, g are edges in clockwise order and (e, g) is not in convex position, then at least one of (e, f) or (f, g) is not either. In designing our filter, we used a 3D variant of this by letting the fence play the role of f. But, unlike in 2D, the fence cannot be a constant size object. Why this implies a lower
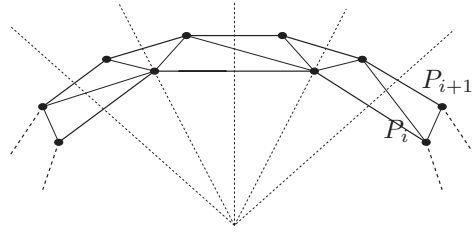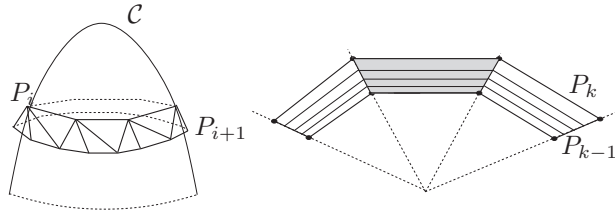
Fig. 19.   The concentric rings of the *xy*-projection.

Fig. 20.   A hard terrain to reconstruct.

bound is explained in this section. We stress that the terrain constructed here will not
have bounded aspect ratio.

We appeal to Yao's *minimax lemma* to deal with the fact that our algorithms are
randomized. We will start with $\varepsilon_\mathcal{D} = \Theta(\log n/n)$. After describing the construction for
this value, we will show how to handle any value of $\varepsilon_\mathcal{D} = \Omega(\log n/n)$. Assume that
the filter changes at most $c\varepsilon_\mathcal{D} n$ faces, for some fixed $c > 1$. We define a distribution of
inputs and show that, for any deterministic algorithm that performs reconstruction,
some query takes $\Omega(n/\log n)$ expected time over that distribution.

We start with a fixed $\mathcal{D}$ and build the distribution around it. Fix some parameter
$m > 0$. The *xy*-projection of $\mathcal{D}$ consists of $\Theta(\log m)$ concentric regular polygons
$P_0, P_1, \ldots, P_{k-1}$ centered at the origin (Figure 19): (i) the innermost polygon P_0 has
a constant number of vertices; (ii) P_i has $2^{i-1}|P_1|$ vertices and every other edge is
parallel to an edge of P_{i-1}; (iii) P_{k-1} has $m/(c_1 \log m)$ vertices, for fixed $c_1 > 0$. The radii
of the P_i's are chosen so that the boundaries are fairly close to each other but disjoint.
Next, we lift these polygons vertically so that their edges are all horizontal tangents to
the paraboloid $\mathcal{C} : Z = -(X^2 + Y^2)$ at their midpoints (Figure 20). Each band between
consecutive polygons is triangulated appropriately and the construction is lifted to $\mathcal{C}$
to form a convex terrain with (lifted) P_0 as its highest face. Finally, we add an extra
polygon P_k that is a slightly scaled-up version of P_{k-1}. The band between P_{k-1} and P_k
consists of $m/(c_1 \log m)$ trapezoids, each one of which is now divided up into a stack of
$c_1 \log m$ parallel subtrapezoids. After lifting, each subtrapezoid finds itself tangent to
$\mathcal{C}$. Setting $m = \Theta(n)$, triangulating all faces produces n faces.

The terrain $\mathcal{D}$ is convex: we introduce convexity violations by choosing one *stack* $\mathcal{S}$ of
subtrapezoids, and tilting them ever so slightly so that: (i) each subtrapezoid violates
one common triangle of P_1; (ii) the stack $\mathcal{S}$ violates $O(1)$ triangles per (P_i, P_{i+1}) band.
This tilting is done so that the common edge between this stack of subtrapezoids and
the trapezoid of P_{k-1} does not move (the tilting is done with this edge hinged). Once
this tilting is done, there will have to be slight modifications performed on this stack
and the stacks adjacent to $\mathcal{S}$ to ensure that all stacks are in convex position with each
other. Note that $\varepsilon_D n < c' \log n$ (for some constant c'). Suppose we decide to keep all the
stacks of subtrapezoids. There are only $O(1)$ triangles in each (P_i, P_{i+1}) band which

violate convexity with $\mathcal{S}$. Since there are $O(\log n)$ bands, the total number of faces which are not in convex position with the stack are $c' \log n$. All the remaining faces are in convex position with each other. This constant c' is independent of c_1 - in other words, we can make c_1 arbitrarily larger than c'.

The filter guarantees to change at most $cc' \log n$ faces. Set $c_1 \log m > cc' \log n > c\varepsilon_{\mathcal{D}} n$. In this way, a query to the common violating triangle of P_1 cannot return the triangle unchanged. Indeed, if it did, then the entire stack $\mathcal{S}$ of $c_1 \log m$ triangles would later have to be modified, which would prove the filter faulty. Modifying the violating triangle of P_1 appropriately requires knowing where the stack $\mathcal{S}$ is placed around the (P_{k-1}, P_k) band, which takes $\Omega(|P_k|)$ expected time.

The extension to higher values of ε is quite straightforward. Essentially, the number of concentric rings is (upto constant factors) equal to εn. Fix some parameter m. We choose concentric regular polygons $P_0, P_1, \ldots, P_{k-1}$ (where $k = \Theta(\varepsilon m)$) such that P_0 has a constant number of vertices, P_i has either $|P_{i-1}|$ or $2|P_{i-1}|$ vertices, and P_{k-1} has $(c_1 \varepsilon_{\mathcal{D}})^{-1}$ vertices (for some suffcently large constant c_1). The outermost polygon P_k is as before a slightly scaled up version of P_{k-1} and the band (P_{k-1}, P_k) consists of stacks of subtrapezoids (as before) with stack size of $c_1 \varepsilon_{\mathcal{D}} m$. One of these stacks is tilted to ensure that it violates one common triangle of P_1 but violates the convexity of at most $O(1)$ subtrapezoids in each ring. The parameter $m = \Theta(n)$ is chosen to ensure that the total number of faces is n. The distance to convexity $\varepsilon_{\mathcal{D}}$ is $\Theta(\varepsilon)$. Using the same argument we can force a query for a common violating triangle in P_1 to make a modification. For this, the tilted stack of subtrapezoids must be detected, which will take $\Omega(P_k) = \Omega(\varepsilon_{\mathcal{D}}^{-1})$ time.

THEOREM 3.16. *Any convexity filter for an arbitrary terrain $\mathcal{D}$ of n faces has a worst case query time of $\Omega(\varepsilon_{\mathcal{D}}^{-1})$ for any n such that $(\log n)/n \leq \varepsilon_{\mathcal{D}}$.*

REFERENCES

AGARWAL, P. K. AND DESIKAN, P. K. 1997. An efficient algorithm for terrain simplification. In *Proceedings of the 8th Annual Symposium on Discrete Algorithms (SODA)*. 139–147.

AGARWAL, P. K., HAR-PELED, S., MUSTAFA, N., AND WANG, Y. 2002. Near-linear time approximation algorithms for curve simplification. In *Proceedings of the 10th Annual Europeam Symposium on Algorithms (ESA)*. 29–41.

AGARWAL, P. K. AND SURI, S. 1998. Surface approximation and geometric partitions. *SIAM J. Comput. 27*, 1016–1035.

AILON, N., CHAZELLE, B., COMANDUR, S., AND LIU, D. 2004a. Estimating the distance to a monotone function. In *Proceedings of the 8th International Workshop on Randomization and Approximation Techniques in Computer Science (RANDOM)*. 229–236.

AILON, N., CHAZELLE, B., COMANDUR, S., AND LIU, D. 2004b. Property preserving data reconstruction. In *Proceedings of the 15th Annual International Symposium on Algorithms and Computation (ISAAC)*. 16–27.

ALON, N., DAR, S., PARNAS, M., AND RON, D. 2000. Testing of clustering. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science (FOCS)*. 240–250.

ALON, N. AND SPENCER, J. 2000. *The Probabilistic Method* 2nd Ed. Wiley.

AMENTA, N., CHOI, S., DEY, T. K., AND LEEKHA, N. 2000. A simple algorithm for homeomorphic surface reconstruction. In *Proceedings of the 16th Annual Symposium on Computational Geometry (SoCG)*. 213–222.

AUSTIN, T., AND TAO, T. 2010. Testability and repair of hereditary hypergraph properties. *Random Structures Algo.* To appear.

BHATTACHARYYA, A., GRIGORESCU, E., JHA, M., JUNG, K., AND RASKHODNIKOVA, S. 2010. Lower bounds for local monotonicity reconstruction from transitive-closure spanners. In *Proceedings of the 14th Workshop on Randomization and Computation (RANDOM)*. 448–461.

CHAKRABORTY, S., FISCHER, E., AND MATSLIAH, A. 2011. Query complexity lower bounds for reconstruction of codes. In *Proceedings of the 2nd Symposium on Innovations in Computer Science (ICS)*. To appear.

CHAN, T. 2006. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *Proceedings of the 17th Annual Symposium on Discrete Algorithms (SODA)*. 1196–1202.

CHAZELLE, B., LIU, D., AND MAGEN, A. 2003. Sublinear geometric algorithms. In *Proceedings of 35th Annual Symposium on Theory of Computing (STOC)*. 531–540.

CHVATAL, V. AND KLINCSEK, G. 1980. Finding largest convex subsets. *Congress. Numer. 29*, 453–460.

CZUMAJ, A., ERGUN, F., FORTNOW, L., MAGEN, A., NEWMAN, I., RUBINFELD, R., AND SOHLER, C. 2003. Sublinear-time approximation of Euclidean minimum spanning tree. In *Proceedings of 14th Annual Symposium on Discrete Algorithms (SODA)*. 813–822.

CZUMAJ, A. AND SOHLER, C. 2001. Property testing with geometric queries. In *Proceedings of 9th Annual European Symposium on Algorithms (ESA)*. 266–277.

CZUMAJ, A. AND SOHLER, C. 2004. Estimating the weight of metric minimum spanning trees in sublinear-time. In *Proceedings of 36th Annual Symposium on Theory of Computing (STOC)*. 175–183.

CZUMAJ, A., SOHLER, C., AND ZIEGLER, M. 2000. Property testing in computational geometry. In *Proceedings of 8th Annual European Symposium on Algorithms (ESA)*. 155–166.

ERGUN, F., KANNAN, S., KUMAR, S. R., RUBINFELD, R., AND VISWANATHAN, M. 1998. Spot-checkers. In *Proceedings of 30th Annual Symposium on Theory of Computing (STOC)*. 259–268.

INDYK, P. 1999a. Sublinear-time algorithms for metric space problems. In *Proceedings of 31st Annual Symposium on Theory of Computing (STOC)*. 428–434.

INDYK, P. 1999b. A sublinear-time approximation scheme for clustering in metric spaces. In *Proceedings of 40th Annual Symposium on Foundations of Computer Science (FOCS)*. 154–159.

KALE, S., PERES, Y., AND SESHADHRI, C. 2008. Noise tolerance of expanders and sublinear expander reconstruction. In *Proceedings of the 49th Foundations of Computer Science (FOCS)*. 719–728.

LIPTON, R. J. AND TARJAN, R. E. 1979. A separator theorem for planar graphs. *SIAM J. Appl. Math. 36*, 177–189.

MEHLHORN, K., NÄHER, S., SEEL, M., SEIDEL, R., SCHILZ, T., SCHIRRA, S., AND UHRIG, C. 1996. Checking geometric programs or verification of geometric structures. In *Proceedings of 12th Annual Symposium on Computational Geometry (SOCG)*. 159–165.

MISHRA, N., OBLINGER, D., AND PITT, L. 2001. Sublinear time approximate clustering. In *Proceedings of 12th Annual Symposium on Discrete Algorithms (SODA)*. 439–447.

SAKS, M. E. AND SESHADHRI, C. 2010. Estimating the longest increasing sequence in polylogarithmic time. In *Proceedings of the 51st Annual IEEE Symposium on Foundations of Computer Science (FOCS)*.