Available online at www.sciencedirect.com

ScienceDirect

Computational Geometry 39 (2008) 191–208

Computational Geometry
Theory and Applications

www.elsevier.com/locate/comgeo

Adaptive sampling for geometric problems over data streams

John Hershberger [a], Subhash Suri [b,*,1]

[a] *Mentor Graphics Corp., 8005 SW Boeckman Road, Wilsonville, OR 97070, USA*
[b] *Computer Science Department, University of California, Santa Barbara, CA 93106, USA*

Received 14 June 2006; received in revised form 20 October 2006; accepted 24 October 2006

Available online 12 December 2006

Communicated by P. Agarwal

Abstract

Geometric coordinates are an integral part of many data streams. Examples include sensor locations in environmental monitoring, vehicle locations in traffic monitoring or battlefield simulations, scientific measurements of earth or atmospheric phenomena, etc. This paper focuses on the problem of summarizing such geometric data streams using limited storage so that many natural geometric queries can be answered faithfully. Some examples of such queries are: report the smallest convex region in which a chemical leak has been sensed, or track the diameter of the dataset, or track the extent of the dataset in any given direction. One can also pose queries over multiple streams: for instance, track the minimum distance between the convex hulls of two data streams, report when datasets A and B are no longer linearly separable, or report when points of data stream A become completely surrounded by points of data stream B, etc. These queries are easily extended to more than two streams.

In this paper, we propose an adaptive sampling scheme that gives provably optimal error bounds for extremal problems of this nature. All our results follow from a single technique for computing the approximate convex hull of a point stream in a single pass. Our main result is this: given a stream of two-dimensional points and an integer r, we can maintain an adaptive sample of at most $2r + 1$ points such that the distance between the true convex hull and the convex hull of the sample points is $O(D/r^2)$, where D is the diameter of the sample set. The amortized time for processing each point in the stream is $O(\log r)$. Using the sample convex hull, all the queries mentioned above can be answered approximately in either $O(\log r)$ or $O(r)$ time.

Keywords: Stream processing; Single-pass algorithms; Convex hulls; Geometric approximations

1. Introduction

A data stream is a potentially unbounded sequence of data elements that must be processed in a single pass, using very little time per element. The result of processing the stream depends on all the elements, even though the processing algorithm may not be able to store them all. Data streams have emerged as an important paradigm for processing data that arrive continuously and must be processed immediately. For instance, telecom service providers continuously monitor packet flows through their networks to look for usage patterns or signs of attack, or to optimize

* Corresponding author.
E-mail addresses: john_hershberger@mentor.com (J. Hershberger), suri@cs.ucsb.edu (S. Suri).

[1] Subhash Suri's research was supported in part by the National Science Foundation grant IIS-0121562.

0925-7721/$ – see front matter
doi:10.1016/j.comgeo.2006.10.004

their routing tables. Financial markets, banks, web servers, and news organizations also generate rapid and continuous data streams. Many data streams are "geometric" in the sense that the location of the source is best thought of as a point in some coordinate space. For instance, cell phone users can be mapped to their geographical locations; retail store or credit card transactions include the location of the point-of-sale. The growing adoption of GPS-based devices also is making it possible to localize and track users of laptops or PDAs, drivers of cars, trucks, or emergency vehicles.

Another source of geometric data streams is the emerging area of sensor networks. Tiny but smart sensors can be deployed in very large quantities over geographically distributed regions and used to monitor physical plants, hazardous environments, natural habitats, etc. These sensors can be networked using wireless technology. A growing number of academic and commercial sensor network projects are underway, with a broad range of applications: smart highways, which monitor and reroute traffic in response to congestion; smart buildings, which detect and automatically respond to changes in climate or hazard conditions; bio-sensor networks, which detect the presence of biologically hazardous agents, or track the spread of a plant disease in a forest or a bacterial/viral disease in ocean life, etc. One example of a large-scale sensor-based monitoring system is the project *EarthScope* [9], which intends to "blanket" the earth with digital seismic sensors to observe the structure of the North American continent.

Broadly speaking, geometric coordinates are an integral part of many data streams. Sensor networks in particular provide a compelling application domain for streams because (1) the data are generated continuously and need to be processed immediately; (2) sensors have very limited storage and, therefore, need to summarize their observations concisely; (3) communication is expensive (largely because of the battery drain of radio transmission), and so it is better to transmit and receive summaries than raw data. But the motivation to study geometric data streams is not limited to sensors. Even traditional spatial or geographical databases benefit from data stream algorithms, because as these databases grow in size, response time constraints force the use of single-pass algorithms. Many important geographical or spatial databases, such as the Sloan Digital Sky Survey [14], already have several terabytes of data and are still growing.

1.1. Our contribution

In this paper, we propose an adaptive sampling scheme for summarizing a streaming set of two-dimensional points. We assume that the data stream consists of an ordered sequence of points $p_1, p_2, \ldots, p_n, \ldots$, where p_1 is the first (i.e., oldest) point in the stream, and p_n is the most recent point observed. Each point p_i is a tuple (x_i, y_i) in some x–y coordinate space. We make no assumption about the order of points—it can be determined by an adversary. Our storage is limited to r points, for some integer r, which is assumed to be significantly smaller than the number of points in the stream. We are interested in *single pass* algorithms, so any point not stored explicitly is lost.

The central object in our study is the convex hull. The convex hull of a set of points P is the smallest convex polygon containing the set P. The number of points on the convex hull can vary from three (a triangle) to $|P|$, when all the points lie on the hull.[2] Thus, if our storage r is smaller than the number of points in the stream, then we are forced to settle for an approximation of the convex hull. We use the *distance* between the true and the approximate convex hulls as the measure of quality. Our approximate convex hull always lies inside the true hull, and we measure error as the maximum distance from the true convex hull to the approximate hull.

Our main result is the following theorem: given a stream of two-dimensional points and an integer r, we can maintain an adaptive sample of at most $2r + 1$ points such that the distance between the true convex hull of all the points and the convex hull of the sample points is $O(D/r^2)$, where D is the diameter of the sample set. All previously published results for approximating the convex hull of a stream have error $\Theta(D/r)$ [7,12]. Our algorithm gives an order of magnitude improvement—for a given error bound, we can reduce the number of sample points from $O(r^2)$ to $O(r)$. Our approximation error is provably optimal—a simple example shows that any convex hull approximation with $O(r)$ points must suffer $\Omega(D/r^2)$ error in the worst case (see Section 5.4). Finally, the per-point processing cost of our sampling scheme is amortized $O(\log r)$ in the worst case.

Data streams often arise in the context of monitoring applications, and identifying and tracking *extreme* values of the signal (e.g., outliers or anomalies) is often a critical goal. For signals that correspond to points in the plane, the convex hull represents the extreme set. Not only does the convex hull convey the *geometric shape* of the point stream,

[2] Bounds are also known for the complexity of the convex hull of random points. For instance, if n points are drawn uniformly at random from a circular disk, then the expected number of points on the convex hull is $\Theta(n^{1/3})$.

but many of the classical extremal queries in computational geometry can also be performed easily and efficiently using the convex hull. These include the *spatial extent* queries: diameter, width, extent in any other direction, farthest neighbors, etc.[3]

Other queries concern spatial separation or enclosure: track the minimum distance between the convex hulls of two data streams; report when data streams A and B are no longer linearly separable; track the overlap between the spatial extents of streams A and B; report when points of data stream A become completely surrounded by points of data stream B, etc. These queries are easily extended to more than two streams. All these queries can be answered easily and efficiently from the convex hulls of the streams.

1.2. Prior work

There has been a significant amount of previous work on computing approximations to data streams. A very brief sample of these results includes one-pass algorithms for estimating the frequency moments [2], quantile summaries and order statistic computation [15,21]. In addition, there is a lot of work on data clustering, decision tree construction, use of wavelets, maintaining samples and statistics over sliding windows, as well as work on high-level architecture for stream database systems. A good summary and review of many of these results can be found in the excellent surveys [3,24].

Besides the data clustering work mentioned earlier, Indyk [18] has considered computing the cost of a minimum spanning tree or a minimum weight matching in a stream of points. Indyk also gives a c-approximation of the diameter of a d-dimensional point stream using $O(dn^{1/(c^2-1)})$ space and the same amount of processing time per point, for any $c > \sqrt{2}$. In two dimensions, Feigenbaum, Kannan, and Zhang [13] improve and extend the diameter approximation. They give two approximation schemes, both using $O(r)$ space: the first scheme requires $O(r)$ processing time per point, and gives a $(1 + O(1/r^2))$-approximation; the second scheme requires $O(\log r)$ processing time per point but gives only a $(1 + O(1/r))$-approximation. The sampling scheme of [13] can also maintain the approximate convex hull of the point stream with error $O(D/r)$. Matias, Vitter, and Young [22] use an approximate van Emde Boas structure [10,11] to maintain an approximate convex hull in $O(\log\log r)$ time per point; the approximate convex hull has error $O(D/r)$. Cormode and Muthukrishnan [7] propose radial histograms for approximating several geometric quantities, including diameter, farthest neighbors, as well as the convex hull. Their convex hull approximation also has error of $O(D/r)$, where r is the sample size and D is the diameter.

Closely related to our research is the recent work on core sets. The idea of a core set was first introduced by Agarwal, Har-Peled, and Varadarajan [1], who called it an *ε-kernel*. Agarwal et al. [1] show that several *extent* problems for point sets can be well-approximated by relatively small subsets, and these subsets can be maintained under insertions and deletions, as well as under the data stream model. In particular, the result of [1] when specialized to two-dimensional point sets states that a subset of size $O(r)$ can be maintained using a data structure of size $O(r \log^2 n)$ such that the extent (projection) of the subset in any direction is a $(1/r^2)$-approximation of the extent of the entire set. The data structure requires $O(r)$ time to update after each new point arrival. Independently of our work, and concurrently with it, Chan [5] has also obtained stream algorithms for approximating geometric quantities such as the diameter and width of a point set. He nearly matches our space and approximation bounds for the diameter, and achieves a better approximation for the width. In particular, Chan's general-purpose algorithm, specialized to the plane, uses $O(r \log r)$ space to achieve a relative error of $O(1/r^2)$ for both diameter and width.

The technique of core sets is quite powerful, and applies more broadly than the specialized algorithm of our paper. This limitation, however, also has its virtues: our algorithm is quite elementary, and has better processing and storage bounds than those achieved through core sets. In particular, both core set algorithms [1,5] build on the approximation technique of Dudley [8]. Our approach, described in Sections 4 and 5, is more local and better suited to low-cost updates than Dudley's technique. Our algorithm has also been implemented, and simulation results show that it performs extremely well in practice.

Finally, in classical approximation theory, several papers have addressed the problem of approximating a convex body [8,16,20,23]. Richardson [27] presents a scheme to approximate a planar convex set by an r-sided polygon such

[3] The *diameter* is the maximum distance between any two points in the stream; the *width* is the minimum distance between two parallel lines that enclose the stream between them.

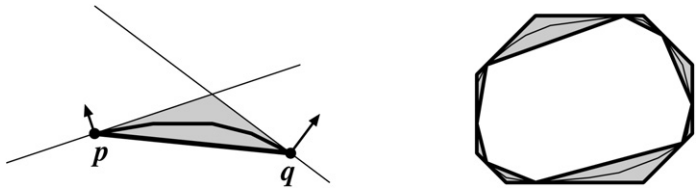

Fig. 1. Uncertainty triangles.

that the *Hausdorff distance* between the two is bounded by $O(D/r^2)$. These results apply to static objects only. Our adaptive sampling scheme not only matches the static-case approximation error bound of Richardson, but also works for streaming point sets.

1.3. Organization

Our paper is organized in eight sections. In Section 2, we introduce the notion of uncertainty triangles, which are central to our error analysis. In Section 3, we describe the simple uniform sampling scheme. In Section 4, we describe our adaptive sampling for static (i.e., off-line) point sets. Then, in Section 5, we show how to implement adaptive sampling in the stream model. In Section 6, we discuss how to use the convex hull approximation to process various queries. Section 7 discusses experimental results obtained using an implementation of our algorithm. Finally, in Section 8, we conclude with a few closing remarks and open problems.

2. Uncertainty triangles

We assume an input stream of two-dimensional points denoted $p_1, p_2, \ldots, p_n, \ldots$, where p_n is the most recent point. The order of points in the stream is completely arbitrary, and we operate in the single-pass algorithm model—any point not explicitly stored is forgotten. The *convex hull* of the points seen so far is the smallest convex polygon containing those points. The corners of this polygon are called *vertices* of the convex hull; the line segments joining adjacent vertices are called *edges* of the convex hull.

Every vertex of a convex hull is *extreme* in some direction: among all the points in the set, it is the farthest in that direction. The vertices appear along the convex hull boundary in the same order as the angular order of their extremal directions.[4]

If our storage is limited to r, then we are forced to drop all but r points from the convex hull, and content ourselves with an approximation to the true hull. Our algorithm will keep a sample set of $O(r)$ points. The convex hull of these sample points is the approximate hull. If the samples are maintained as extrema in a fixed set of directions, then each sample point is also a vertex of the true convex hull. Let us first analyze the approximate error under the assumption that every sample point is extreme in some direction. In the case of streaming data (Section 5), this assumption may not hold, and the error analysis is more subtle.

Suppose p and q are two consecutive sample points along the boundary of the approximate convex hull. Then, in the true convex hull, there is a convex chain from p to q, which has been collapsed to the segment $\overline{pq}$ in our approximation. Therefore, we define the approximation error for the edge $\overline{pq}$ as the maximum distance from $\overline{pq}$ to a vertex on the path joining p to q in the true hull.

We cannot measure this error exactly because the convex chain from p to q has been discarded—the approximate hull only kept p and q. Instead, we bound the error using *uncertainty triangles*, defined as follows. Suppose that p and q are extrema of the original set of points in directions θ_p and θ_q. The *supporting lines* of p and q are the lines through p and q that are perpendicular to θ_p and θ_q, respectively. The triangle bounded by $\overline{pq}$ and the supporting lines of p and q is the uncertainty triangle for $\overline{pq}$; all the vertices of the true convex hull on the path between p and q lie in this triangle. The error for edge $\overline{pq}$ is no larger than the height of the triangle. See Fig. 1, left side. Every

[4] If three or more points are extreme in some direction θ, only two of them are hull vertices, namely those that are extreme in a non-zero range of directions. The other extrema in direction θ lie in the interior of the convex hull edge with the two vertices as its endpoints.

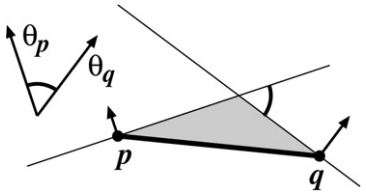

Fig. 2. The sum of the angles of the uncertainty triangle at p and q equals $\theta(\overline{pq})$.

edge of the approximate hull has an uncertainty triangle, and so the true hull is sandwiched in the ring of uncertainty triangles. See Fig. 1, right side.

We can bound the height of the uncertainty triangle for an edge $\overline{pq}$ based on the length of $\overline{pq}$ and the angles of the supporting lines of p and q. We can assume that the extreme directions for p and q, namely, θ_p and θ_q, point toward the same side of $\overline{pq}$, and hence the intersection of the supporting lines projects perpendicularly onto $\overline{pq}$. Therefore the height of the uncertainty triangle is at most $\ell(\overline{pq})$ times the tangent of the smaller of the angles between $\overline{pq}$ and the supporting lines, where $\ell(x)$ denotes the length of the line segment x. Observe (Fig. 2) that the sum of these two angles equals the angle between the directions θ_p and θ_q. If we define $\theta(\overline{pq})$ to be $|\theta_p - \theta_q|$, then the height of the uncertainty triangle at $\overline{pq}$ is at most $\ell(\overline{pq}) \cdot \tan(\theta(\overline{pq})/2)$, which is very close to

$$\frac{\ell(\overline{pq}) \cdot \theta(\overline{pq})}{2}. \tag{1}$$

In the error bounds for streaming points, we replace the supporting lines at p and q by lines parallel to the supporting lines and a short distance outside the hull. Each line defines a half-plane, and the true hull lies in the intersection of all the half-planes. See Section 5 for details.

3. The uniformly sampled hull

Since both the edge lengths and the extreme directions of a sampled convex hull are determined by the sample points, the approximation error of the hull depends crucially on the criteria used to select its vertices. One simple way to choose the sample points is to select extrema in r fixed, evenly spaced directions, for some integer parameter r. If we define $\theta_0 = 2\pi/r$, these directions are $j\theta_0$, for $j = 0, \ldots, r-1$. This is a very natural idea that has been used frequently in computational geometry. In the context of the diameter of a streaming set points, this is used by Feigenbaum, Kannan, and Zhang [13]; Cormode-Muthukrishnan's radial hull [7] can also be viewed as a two-level variation of this idea.

See Fig. 3 for an example of the uniformly sampled hull. If we use r samples, then the worst-case error of the uniformly sampled hull is approximately $\frac{1}{2}D\theta_0 = \pi D/r$, where D is the *diameter* of the point set—an upper bound on the length of any convex hull edge. Interestingly, although the uncertainty triangles' error is $\Theta(D/r)$, one can still use the uniformly sampled hull to approximate the diameter within error $O(D/r^2)$. Intuitively, this happens because the worst-case error is achieved only for sampled hull edges whose length is around D; shorter edges have smaller uncertainty triangles. See Fig. 4. The following fact is easily shown; for the sake of completeness, we include its proof, which is essentially borrowed from [13].

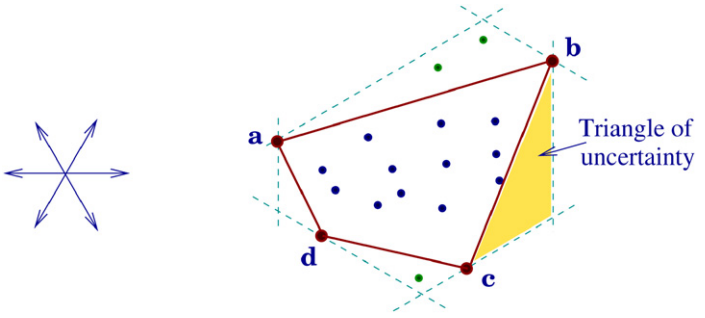

Fig. 3. Uniformly sampled hull, with $r = 6$. The sample hull's vertices are a, b, c, d.

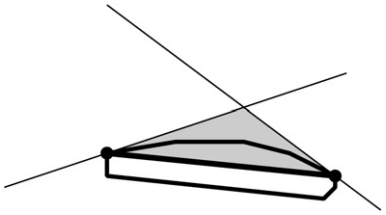

Fig. 4. Large uncertainty triangles occur only for skinny point sets.

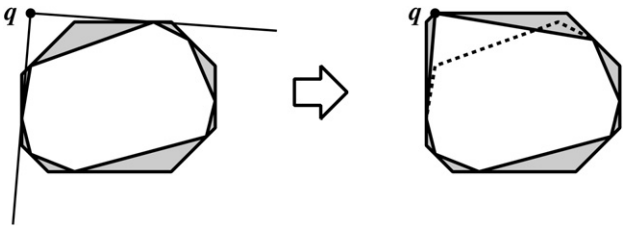

Fig. 5. The new point q replaces extrema in directions found by considering the tangents from q to the hull.

Lemma 3.1. *Let D be the diameter of a point set S, and let $\widetilde{D}$ be the diameter of the extrema of S in directions $j\theta_0$, for $j = 0, \ldots, r-1$. Then $D = \widetilde{D}(1 + \mathrm{O}(1/r^2))$.*

Proof. Let $a, b \in S$ be the two points that determine D, and let θ be the direction of the vector $a - b$. Let $\bar{j}\theta_0$ be the sample direction closest to θ. The angle between θ and $\bar{j}\theta_0$ is at most $\theta_0/2$. The extrema in directions $\bar{j}\theta_0$ and $\pi + \bar{j}\theta_0$ must be at least as far apart as the projections of a and b onto direction $\bar{j}\theta_0$, which is at least $D\cos(\theta_0/2) = D(1 - \mathrm{O}(\theta_0^2)) = D(1 - \mathrm{O}(1/r^2))$. That is, $D \geqslant \widetilde{D} \geqslant D(1 - \mathrm{O}(1/r^2))$, which is equivalent to the statement to be proved. □

3.1. Per-point processing cost

A straightforward implementation of the uniform sampling strategy, as proposed in [13], takes $\Theta(r)$ time to process each point. One maintains a separate extremum for each direction $j\theta_0$. When a new stream point arrives, compute its dot product with each of the r extreme direction vectors, and replace the current extremum if the new point has a larger dot product. At any time, the approximate diameter is the maximum distance between the extrema in each pair of opposite sample directions. (Assume that r is even.)

We can reduce the cost of the uniform sampling approach to $\mathrm{O}(\log r)$ time per point, using standard techniques from computational geometry. Instead of storing the extrema indexed by direction, we store the extrema in a searchable, concatenable list structure, implemented as a balanced binary tree [6], a skip list [26], or (concretely) as a C++ STL `set` [19]. The list is sorted by the directions in which the vertices are extreme. Each vertex stores the integer range $[l, h]$ such that it is extreme in directions $j\theta_0$ for all $l \leqslant j \leqslant h$. A searchable list of convex hull vertices supports many geometric queries in logarithmic time [4,25]. In particular, binary search on the list of $\mathrm{O}(r)$ hull vertices takes $\mathrm{O}(\log r)$ time to detect whether a query point q is inside the hull, and, if not, to find the tangents from q to the hull (the tangents are the lines through q and at least one hull vertex that have all the other hull vertices on the same side).

We can use list operations to update the hull when a new data stream point q arrives. First we check if q is inside the current hull. If so, we ignore it. If not, we find the tangents from q to the current hull. By looking at the slopes of the tangent lines and the direction indices of the vertices that the tangent lines touch, we can determine whether q is extreme in any of the sample directions. If not, we ignore q. Otherwise, the hull vertices that are no longer extrema must be deleted from the list, and q must be inserted in their place. (See Fig. 5.) If the list is implemented as an STL `set`, this takes $\mathrm{O}((k+1)\log r)$ time, where k is the number of vertices deleted, but other implementations can reduce this to $\mathrm{O}(\log r)$. In any case, each point can be deleted from the convex hull only once, so the amortized time to process each point is $\mathrm{O}(\log r)$. We summarize this observation in the following lemma.

Lemma 3.2. *The convex hull of r uniformly sampled extrema can be maintained for a stream of points in* $O(\log r)$ *amortized time per point. The uncertainly triangles of the uniformly sampled hull have height* $O(D/r)$.

The uniformly sampled hull can be a poor approximation of the true convex hull around long edges. For instance, if the point stream has a long skinny shape, then its *width* can be arbitrarily smaller than its diameter. Similarly, for tracking linear separation or minimum distance between two hulls, the absolute error of $\Theta(D/r)$ can yield *unbounded relative error*.

3.2. Towards adaptive sampling

We now come to our main technical contribution: reducing the convex hull approximation error from $O(D/r)$ to $O(D/r^2)$. Our scheme builds on the uniformly sampled hull—we always maintain the extrema in the r fixed uniform directions. But we then add extrema in at most $r+1$ additional directions, which are chosen adaptively based on the current hull. Of course, a straightforward choice of r new samples will typically reduce the error only by a factor of 2. Our key contribution is to find r new directions that reduce the error *quadratically*.

A key challenge of adaptive sampling is that the sampling directions change with time—old directions may be replaced by new extremum directions, invalidating previous work and forcing us to reinstate the structural invariants with respect to new directions. This has implications both for processing cost (we need to show that changes in directions do not require too much new processing), and for error analysis (we need to show that all the past data that were lost before a direction θ was chosen to be a sampling direction do not adversely affect the error bound).

4. Adaptive sampling scheme for a static point set

Simply put, the goal of adaptive sampling is to reduce each convex hull edge length to $O(D/r)$, because Eq. (1) bounds the error by $O(D/r^2)$ for such edges. However, in flat sections of the true convex hull, even very fine sampling may not shorten an edge. In this case we also take into account the angular range associated with the edge, which indirectly bounds the height of the uncertainty triangle for the edge. We present the ideas procedurally in this section, but we are not concerned with algorithmic efficiency. Instead, we simply want to describe how to build a static structure for a fixed set of points. We show how to maintain the structure for a stream of points in Section 5.

For a static set, we first find the extrema in the uniformly sampled directions $j\theta_0$, for $j = 0, \ldots, r-1$, then add up to $r+1$ more extrema in additional directions, chosen adaptively. To guide the sampling, we define a *sample weight* for each edge of the convex hull. Given an edge e of the current hull, let $\theta(e)$ be the minimum angle between the directions in which the two endpoints of e have been sampled as extrema. The sample weight of an edge e grows with the length of e and decreases with $\theta(e)$. Each edge of the uniformly sampled hull has a weight, some large and some small. We repeatedly *refine* large-weight edges (choose new sample directions between their endpoints' directions) to reduce all the edge weights to below a fixed threshold.

After the initial uniform sampling, $\theta(e) = \theta_0 = 2\pi/r$ for all hull edges e. Let P be the perimeter of the uniformly sampled hull. For a given edge e, let $\tilde{\ell}(e)$ be the *total length of the other two sides of the uncertainty triangle above* e. Note that $\tilde{\ell}(e)$ is greater than the length of e, but not by much—if $\ell(e)$ is the length of e, $\tilde{\ell}(e) \leqslant \ell(e)/\cos\theta_0 = \ell(e)(1 + O(1/r^2))$. We define the *sample weight* of an edge e as

$$w(e) = r\frac{\tilde{\ell}(e)}{P} - \log_2\bigl(\theta_0/\theta(e)\bigr).$$

Recall that $2D \leqslant P \leqslant \sum_e \ell(e) < \pi D$, so the first term of $w(e)$ is the approximate multiple by which $\ell(e)$ differs from the ideal $O(D/r)$. The second term decreases the weight as the maximum height of the uncertainty triangle above e decreases. The dependence is logarithmic, rather than linear, for convenience in later calculations; however, we could also have used a second term of $-\theta_0/\theta(e)$ instead.

This weight function uses both internal and external length approximations. P is the sum of $\ell(e)$ for the edges of the uniformly sampled hull, whereas $\tilde{\ell}(e)$ is an upper bound on the length of the arc of the true convex hull that e approximates. Here are the reasons for these choices:

- Instead of P we could have used the actual perimeter $\sum_e \ell(e)$ for the current approximation; however, in this case the perimeter would change as edges were refined, and refinement of one edge would affect the weights of other

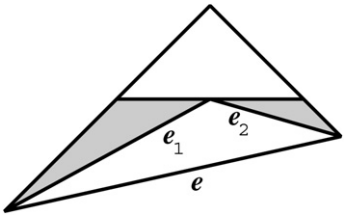

Fig. 6. The triangle inequality shows that $\tilde{\ell}(e_1) + \tilde{\ell}(e_2) \leqslant \tilde{\ell}(e)$.

edges. By fixing P based on the uniformly sampled hull, we ensure that each edge can be refined independently of the others.

- If we used the actual length of an edge $\ell(e)$ in the weight formula, the total length of the replacement edges would exceed that of the original edge. Using $\tilde{\ell}(e)$ instead of $\ell(e)$ makes the total decrease, and simplifies the proof of Lemma 4.1.

After the initial uniform sampling, the total sample weight, summed over all edges of the hull, is at most $r + O(1/r)$. To sample adaptively, we repeatedly choose an edge e with $w(e) > 1$ and refine it. That is, we introduce a new sampling direction that bisects the angular range defined by e's endpoints. We determine the extreme point p in this new direction. If p is distinct from both endpoints of e, then we replace e by the two edges e_1 and e_2 defined by p and e's endpoints. Note that $\tilde{\ell}(e_1) + \tilde{\ell}(e_2) \leqslant \tilde{\ell}(e)$. See Fig. 6. If p is the same as an endpoint of e, then we simply halve the angular range associated with e; that is, $\theta(e)$ is cut in half. Because refinement bisects an edge's angular interval, the second term in $w(e)$ just counts the number of refinement steps that have been applied to obtain a particular edge e.

The following lemma helps us bound the number of refinement steps that are applied.

Lemma 4.1. *Each refinement step decreases the total of all the positive sample weights by at least* 1.

Proof. Refinement is applied to an edge e only if $w(e) > 1$. The initial weight $w(e) = \tilde{\ell}(e) \cdot (r/P) - m$, where $m \geqslant 0$ is the number of refinements applied to obtain e. The weights of the replacement edges e_1 and e_2 are $w(e_1) = \tilde{\ell}(e_1) \cdot (r/P) - m - 1$ and $w(e_2) = \tilde{\ell}(e_2) \cdot (r/P) - m - 1$. The sum of these replacement weights is at least $m + 2$ less than the original weight, and the larger of the replacement weights is at least 1 less than the original weight. We bound the sum of the positive replacement weights.

$$\begin{aligned}
&\max\bigl(w(e_1), 0\bigr) + \max\bigl(w(e_2), 0\bigr) \\
&\quad = \max\bigl(\tilde{\ell}(e_1) \cdot (r/P) - m - 1, 0\bigr) + \max\bigl(\tilde{\ell}(e_2) \cdot (r/P) - m - 1, 0\bigr) \\
&\quad \leqslant \max\bigl(\tilde{\ell}(e) \cdot (r/P) - m - 1, 0\bigr) + \max(-m - 1, 0) \\
&\quad \leqslant w(e) - 1.
\end{aligned}$$

This concludes the proof. □

Because the sum of the sample weights before refinement is at most $r + o(1)$, the preceding lemma implies the following bound on the number of samples:

Lemma 4.2. *Adaptive sampling adds at most* $r + 1$ *new extrema to the sample set.*

Intuitively, our adaptive sampling process ensures that no edge is longer than $O(D/r)$ unless its uncertainty triangle is very flat. This leads to the following important approximation bound.

Lemma 4.3. *After adaptive sampling, every edge of the hull has an uncertainty triangle with height* $O(D/r^2)$*, where* D *is the diameter of the point set.*

Proof. After sampling, $w(e) \leqslant 1$ for all edges. That is,

$$\tilde{\ell}(e)\frac{r}{P} \leqslant 1 + \log_2\bigl(\theta_0/\theta(e)\bigr).$$

The height of e's uncertainty triangle is at most $\tilde{\ell}(e) \times \sin(\theta(e)/2)$. If $\tilde{\ell}(e) = kP/r$ for some k (not necessarily an integer), then

$$\theta(e)/2 < \theta_0/2^k.$$

In this case, the error (height) is at most

$$\frac{kP}{r}\sin(\theta_0/2^k) = \mathrm{O}\left(\frac{P}{r^2}\cdot\frac{k}{2^k}\right) = \mathrm{O}\left(\frac{D}{r^2}\cdot\frac{k}{2^k}\right) = \mathrm{O}(D/r^2).$$

The maximum value of $k/2^k$ is achieved for k between 1 and 2. That is, the worst-case error of $\mathrm{O}(D/r^2)$ is obtained for edges shorter than $2P/r$ and $\theta(e) \geqslant \theta_0/2$. □

This is a significant improvement over uniform sampling—using the same number of sample points, we get an order of magnitude improvement in approximation quality. Or, alternatively, for the same error bound, we can reduce the number of sample points from $\mathrm{O}(r^2)$ to $\mathrm{O}(r)$.

5. Adaptive sampling for streaming points

The challenge of performing adaptive sampling on a data stream is that the adaptive sampling directions change over time. We need to show that if a point was ignored in the past (because it was not extreme in the old set of sampling directions), then that point does not materially affect the error bounds in the future, as the sampling directions change. A secondary challenge is the per-point processing cost—as old sampling directions retire and are replaced by new sampling directions, we need to show that the structural invariants can be restored without too much computational overhead. Our scheme will meet both these goals.

When a new point arrives in the stream, we determine whether it modifies the current hull or not. If the new point is inside the current hull, then, of course, we simply drop it. Otherwise, we add it to the convex hull. This addition may increase the perimeter of the uniformly sampled hull. The change in the convex hull's perimeter has two implications for adaptive sampling:

1. The increase in the perimeter may reduce the weight of some edges below the refinement threshold, so an edge that is currently refined may no longer need refinement. In this case, one or more of the current extrema may become unnecessary; if this occurs, we discard these extrema from the hull.
2. The edges incident to the new vertex may have weights above the refinement threshold, and thus need refinement. This means that we reduce their angular ranges (without adding any more extreme points).

We maintain the invariant that the sample weight $w(e)$ is at most 1 for each edge of the current hull at all times. We provide more details in the remainder of this section.

5.1. The refinement tree

The adaptively sampled hull is a refinement of the uniformly sampled hull. The refinement of each edge of the uniformly sampled hull is represented by a binary tree. Each node of the tree corresponds to an angular range, and the children of the node, if any, correspond to the subranges obtained by bisecting their parent's range. A node stores a convex hull edge and its uncertainty triangle; the endpoints of the edge are extrema of the sampled hull in the directions that are the ends of the angle range. If an edge $\overline{pq}$ is refined, its node in the tree is an internal node, and we also associate with it the extreme vertex t in the direction that bisects the node's angle range. The node's children correspond to the two refined edges $\overline{pt}$ and $\overline{tq}$. If $t \notin \{p, q\}$, then both children are nontrivial *edge nodes*. If t equals one of the endpoints, say p, then one child is a *vertex node* for p (conceptually, a zero-length edge that is not refined further) and the other is an edge node for $\overline{pq}$ with a reduced angular range and uncertainty triangle. Note that the tree

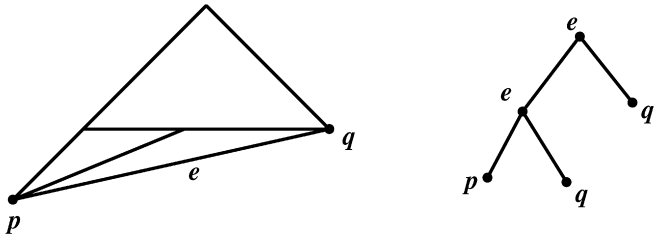

Fig. 7. Repeated bisection of an edge e's angular range creates a long path in the refinement tree.

may contain long paths whose nodes store the same hull edge. These paths correspond to refinements that reduce the angular range of an edge without finding an extremum outside the uncertainty triangle. See Fig. 7. By Lemma 4.1, the total size of all the refinement trees for all the uniformly sampled hull edges is $O(r)$.

When a new vertex is added to the current hull, restoring the edge weight invariant requires us to visit all those refinement tree nodes that need to be unrefined. In order to bound the number of such nodes, we enforce a *height limit* on the refinement tree. We can limit the tree height to k, for any $k \geqslant \log_2 r$. When $k = \log_2 r$, we have $\theta(e) = \theta_0/r$ for every hull edge e with sample weight greater than 1, and so the approximation error for e is $O(D/r^2)$. At the other extreme, when $k = 0$, this reduces to uniform sampling. Thus the tree height parameter can be used to control the degree of adaptive sampling.

Remark. By exploiting the infinite-precision arithmetic of the Real RAM model, one can obtain a theoretical algorithm that compresses to a single link each long path in the refinement tree that repeatedly bisects the angular range of a single edge. This reduces the number of tree nodes visited to be proportional to the number of vertices deleted. However, there is no free lunch: the binary tree structure of the long paths is encoded in the bit patterns representing the slope of the edge whose angular range is being reduced, and a practical computation model must still perform an amount of work proportional to the path length.

5.2. The algorithm

We can now describe the main steps of our adaptive convex hull algorithm. When a new data stream point q arrives, we perform the following steps:

Algorithm AdaptiveHull

1. If q is inside the ring of uncertainty triangles of the adaptive hull, discard q and return.
2. Insert q into the uniformly sampled hull, as in Section 3, and recompute P, the perimeter of the uniformly sampled hull.
3. If q is extreme in two consecutive directions $j\theta_0$ and $(j+1)\theta_0$ (for any j), and the uniformly sampled hull edge connecting those extrema was previously nontrivial, delete the refinement tree associated with that edge.
4. Unrefine all non-leaf refinement tree nodes whose sample weight is now less than 1.
5. Remove all refinement tree nodes invalidated by q and rebuild the (at most two) refinement trees that q affects.

Step 1. We store the edge leaves of each refinement tree in a searchable list, so binary search operations on the adaptively sampled hull can be performed by combining a top-level search on the uniformly sampled hull with a bottom-level search on the refinement of a uniformly sampled hull edge.

Step 2–3. Step 2 takes $O(\log r)$ time to insert q in the uniformly sampled hull, and Step 3 takes $O(1)$ time per deletion.

Step 4. If q changes the uniformly sampled hull, then its perimeter P increases, and some refined edges may need to be unrefined, even in portions of the convex hull not directly affected by q. Each internal node of a refinement tree has a sample weight $w(e) = \tilde{\ell}(e) \cdot (r/P) - d(e)$, for some integer $d(e)$. The weight $w(e) > 1$, but for some threshold value of P it will drop below 1, at which time the two children of the node should be deleted and the node should

become a leaf. This causes two edges of the adaptively sampled hull to be replaced by a single edge. When we create an internal node, we enter it in a priority queue, indexed by its threshold. We unrefine each edge when P increases to above its threshold.

Step 5. When a new point q is inserted, it may invalidate some nodes of a refinement tree. Each node has an edge (a, b) and an angular range (θ_a, θ_b) associated with it. Vertex a is extreme in direction θ_a and vertex b is extreme in direction θ_b. (If the node is a vertex node, then $a = b$.) The node is invalid if q is farther in direction θ_a than a or farther in direction θ_b than b. All invalid nodes must be removed from the tree and replaced by new nodes whose edges have at least one end at q.

The invalid nodes are easy to find: the edge of the adaptive hull closest to q is invalid if any edge is. The invalid edges form a contiguous sequence of the hull boundary. These edges correspond to leaves of the refinement tree, and all the invalid nodes are ancestors of these leaves. They can be found in $O(km)$ time, where m is the number of invalid hull edges. Deleting all the invalid nodes breaks the tree into $O(k)$ subtrees.

To rebuild the refinement tree, we essentially need to re-run the static algorithm of Section 4, starting from the lowest common ancestor of the invalid nodes, using q and the edge endpoints stored in the roots of the subtrees that remain after deleting invalid nodes. This takes $O(k)$ time. It is possible that some unrefinement may be needed at the end of this process—a new node using q may have weight less than 1 and not require edge refinement, even though the previous (invalid) node did. This is easy to handle, as in Step 4. When we are done rebuilding the refinement tree, we have essentially computed the static adaptively sampled hull on the vertices of the previous adaptive hull plus q.

5.3. Complexity and error analysis

The running time of the update algorithm is $O(\log r)$ plus $O(1 + \operatorname{Pri} Q(r))$ per refinement tree node affected. Here $\operatorname{Pri} Q(r)$ is the time needed for a single operation on the priority queue of size r that contains internal nodes indexed by their unrefinement thresholds. A standard priority queue implementation gives $\operatorname{Pri} Q(r) = \Theta(\log r)$, but we will show how to reduce this below.

The update running time is $O(\log r + r \cdot \operatorname{Pri} Q(r))$ in the worst case, but can also be bounded by $O(\log r + k(m + 1) \operatorname{Pri} Q(r))$, where k is the height limit on the refinement trees and m is the number of vertices of the adaptive hull that are deleted. Since each vertex can be deleted only once, this is $O(\log r + k \cdot \operatorname{Pri} Q(r))$ amortized time. Note also that the worst-case k factor applies only if the affected edges are very long. If the affected edges have length proportional to D/r, on the average, the amortized running time is $O(\log r + \operatorname{Pri} Q(r))$ per update, which is $O(\log r)$ using a standard priority queue.

Using an idea suggested by Yossi Matias (personal communication), we can reduce $\operatorname{Pri} Q(r)$ to $O(1)$. Instead of using

$$\operatorname{Thresh}(e) = \frac{r \cdot \tilde{\ell}(e)}{1 + d(e)}$$

as the threshold value of P at which an edge e will be unrefined (here $\tilde{\ell}(e)$ and $d(e)$ are parameters specific to the edge e), we use the power-of-two value $2^{\lfloor \log_2 \operatorname{Thresh}(e) \rfloor}$. This means that e may be unrefined slightly too early, but the approximation quality is asymptotically unchanged; it is not hard to verify that the error is still $O(D/(r2^k))$. We can represent the priority queue by an array of $\lceil \log_2 r \rceil$ buckets, corresponding to the $\lceil \log_2 r \rceil$ smallest powers of two greater than the current value of P. By using this representation and standard RAM operations, we get $\operatorname{Pri} Q(r) = O(1)$, and the worst-case amortized running time of the update algorithm is $O(\log r + k) = O(\log r)$.

To prove that the error bound of Lemma 4.3 extends to the streaming case, we need to introduce some notation. Because our sample directions are obtained by bisecting angular intervals, each sample direction θ can be expressed as a multiple of $\theta_0/2^i$ for some i. For a sample direction θ, we denote by $\operatorname{index}(\theta)$ the smallest integer i such that θ is a multiple of $\theta_0/2^i$. Then $\operatorname{index}(\theta)$ is one more than the depth in the refinement tree of the node where θ is used as a bisecting direction.

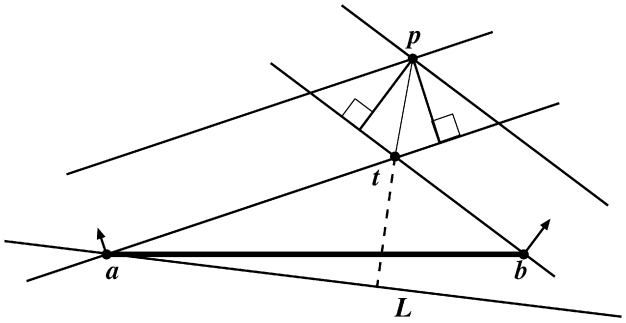

Fig. 8. Bounding the distance from L to p.

For each sample direction θ, we define a line $L(\theta)$ perpendicular to θ. If $\text{index}(\theta) = 0$, that is, θ is a multiple of θ_0, then $L(\theta)$ is the supporting line passing through the sample point p associated with direction θ. Otherwise $L(\theta)$ is obtained by translating the supporting line in direction θ by

$$d_{\text{index}(\theta)} = (8\pi P/r^2) \sum_{j=1}^{\text{index}(\theta)} j/2^j.$$

Note that the distance from the sample point to $L(\theta)$ depends on $\text{index}(\theta)$, the number of samples r, and the perimeter of the uniformly sampled hull. Each line $L(\theta)$ bounds a half-plane that contains the sample hull.

We will maintain the following invariant:

Invariant. For each currently active sample direction θ, the current true hull lies in the half-plane bounded by $L(\theta)$.

Because $L(\theta)$ is a supporting line for θ a multiple of θ_0, the invariant implies that the adaptive hull is no worse than a uniformly sampled hull with r samples. However, we can use the following lemma to prove a much better bound.

Lemma 5.1. *Let $e = \overline{ab}$ be an edge of the adaptive hull, and let i be the larger of the indices for θ_a and θ_b. Let L be any line that passes through an endpoint of e and has its normal in the range (θ_a, θ_b). Then no point of the current data stream lies more than distance d_{i+1} from L in the direction away from the sample hull.*

Proof. Because $w(e) \leqslant 1$, we have $\tilde{\ell}(e) \leqslant \frac{P}{r}(1 + \log_2(\theta_0/\theta(e)))$. Since refinement directions are chosen by hierarchical bisection, $\theta(e) = |\theta_a - \theta_b| = \theta_0/2^i$. Thus $\tilde{\ell}(e) \leqslant \frac{P}{r}(1 + i)$.

By the invariant, all data points lie in the wedge determined by lines perpendicular to θ_a and θ_b and at distance d_i from a and b, respectively. The distance of the intersection of these two lines (call it p) from L is an upper bound on the maximum distance of any data point from L (on the side away from the sample hull). We bound this distance using the triangle inequality by the sum of the distance from L to the apex t of e's uncertainty triangle, plus the distance from t to p. See Fig. 8.

Without loss of generality suppose L passes through a. The angle between L and the uncertainty triangle edge through a is at most $\theta(e)$, and $\ell(\overline{at}) \leqslant \tilde{\ell}(e)$, and so the distance from L to t is at most

$$\ell(\overline{at}) \sin\theta(e) \leqslant \tilde{\ell}(e)\theta(e) \leqslant \frac{2\pi P}{r^2}\left(\frac{i+1}{2^i}\right).$$

We bound $\ell(\overline{pt})$ by dropping perpendiculars from p to the supporting lines of a and b. These perpendiculars form an angle of $\theta(e)$, and so $\ell(\overline{pt}) \leqslant d_i/\cos(\theta(e)/2)$. Recall that $1/\cos\phi = 1 + \phi^2/2 + \mathrm{O}(\phi^4)$, and that $\sum_{j\geqslant 1}(j/2^j) = 2$. We bound the sum of the two segment lengths by

$$\begin{aligned}
&\frac{2\pi P}{r^2}\left(\frac{i+1}{2^i}\right) + d_i + d_i\left(\left(\theta(e)\right)^2/8 + \mathrm{O}\left(\left(\theta(e)\right)^4\right)\right) \\
&\quad = d_i + \frac{2\pi P}{r^2}\left(\frac{i+1}{2^i}\right) + d_i\left(\frac{\pi^2}{r^2 2^{2i+1}} + \mathrm{O}\left(\frac{1}{r^4 2^{4i}}\right)\right)
\end{aligned}$$

$$\leqslant d_i + \frac{4\pi P}{r^2}\left(\frac{i+1}{2^{i+1}}\right) + \frac{16\pi P}{r^2}\left(\frac{\pi^2}{r^2 2^{2i+1}} + \mathrm{O}(r^{-4})\right)$$
$$\leqslant d_i + (8\pi P/r^2)\left(\frac{i+1}{2^{i+1}}\right) = d_{i+1},$$

where the last inequality holds for all $r > 2\pi$. □

An error bound matching that of Lemma 4.3 is an immediate consequence of this lemma.

Corollary 5.2. *At each instant during adaptive sampling of a stream of points, the true hull lies within distance* $\mathrm{O}(D/r^2)$ *of the adaptive hull, where* D *is the diameter of the point set.*

Proof. For each edge $e = \overline{ab}$ of the adaptive hull, we apply Lemma 5.1 to the line L supporting e. It follows that no stream point lies more than d_{i+1} from L in the direction away from the hull, where $i = \max(\mathrm{index}(\theta_a), \mathrm{index}(\theta_b))$. Since $d_{i+1} < d_\infty = (8\pi P/r^2)\sum_{j\geqslant 1}(j/2^j) = 16\pi P/r^2$, and $P = \Theta(D)$, we have established the claim. □

We now prove that the invariant holds.

Lemma 5.3. *The invariant stated above is true at every instant for the adaptive hull of streaming points.*

Proof. The proof is by induction. The base case is established when the first point arrives. The single point is extreme in all directions, and there is no error in the sampled hull, and so the invariant is trivially true.

We now assume that the invariant holds prior to the arrival of a new point q, then argue that the steps of Algorithm AdaptiveHull maintain the invariant.

Step 1. If q is discarded, it is not extreme in any of the current sample directions. Therefore it is contained in the half-plane of $L(\theta)$ for each sample direction θ.

Step 2. If P grows in this step, then d_i is non-decreasing for each i, since d_i depends linearly on P. For every sample direction θ, the line $L(\theta)$ either stays fixed or moves away from the hull. All points that were previously contained in its half-plane continue to be contained.

We consider performing Steps 3, 4, and 5 in a different but equivalent order:

4. Unrefine all non-leaf edges on the current hull whose sample weight is now less than 1.
5a. For each sample direction in which q is extreme, update the vertex at the associated refinement tree node. (This shortens and lengthens some edges.)
5b, 3. Unrefine all non-leaf edges whose sample weight is less than 1. (In particular, the edges removed by the original Step 3 have zero length, and hence are unrefined out of existence.)
5c. Refine all leaves with sample weight greater than 1. (There are at most two, corresponding to edges from q to vertices of the old hull.)

Now we argue that the invariant is preserved by these changes.

Steps 4, 5b, 3. Unrefinement just removes elements from the set of sample directions; the invariant continues to hold for those that remain.

Step 5a. For each sample direction θ, replacing the old extremum by q moves $L(\theta)$ outward, and so the invariant continues to hold.

Steps 5c. Consider an edge $\overline{aq}$ that is refined. Prior to q's arrival, a belonged to an edge $e = \overline{ab}$. Point q is extreme in direction θ_b and not in θ_a. Let i be $\max(\mathrm{index}(\theta_a), \mathrm{index}(\theta_b))$. Each refinement of $\overline{aq}$ selects either a or q as the extremum in some direction θ, with $\theta_a < \theta < \theta_b$ and $\mathrm{index}(\theta) > i$. The line $L(\theta)$ through the extremum and

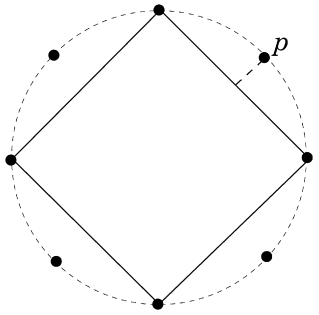

Fig. 9. The distance from p to the convex hull is $\Omega(D/r^2)$.

perpendicular to θ either satisfies the hypothesis of Lemma 5.1 for the segment $\overline{ab}$ (if the extremum is a) or is farther from the point set than such a line (if the extremum is q). By the lemma, no point of the stream lies farther than d_{i+1} in direction θ from $L(\theta)$. Since $\text{index}(\theta) \geqslant i+1$, the invariant holds for the refinement directions.

This completes the proof that the invariant is preserved when a point is added to the adaptive hull. □

Corollary 5.2, which follows from the invariant, shows that the error bound for the static case also holds for streams. To minimize running time and maximize accuracy, we choose $k = \log_2 r$. The error bound and the choice of k lead to our main theorem:

Theorem 5.4. *Given a stream of data points and an integer r, we can maintain an adaptive sample of at most $2r+1$ points such that the distance from the true convex hull to the hull of the sample is $O(D/r^2)$, where D is the current diameter of the sample set. The amortized time to update the sample after a new point is added is $O(\log r)$ in the worst case.*

The amortized update time bound can be made into a worst-case bound, at the expense of a more complex algorithm, which we now sketch. The operations of AdaptiveHull can be classified into five groups: binary search, tree surgery, node additions, node deletions, and unrefinements. The first three take $O(\log r)$ time in the worst case; the last two may take more. To make the overall update algorithm run in $O(\log r)$ time, we perform the first three kinds of operations immediately, and create a queue of node deletions and unrefinements to be carried out later. (This requires Step 5 to use binary search and tree surgery to remove subtrees, not individual nodes.) At each invocation of the enhanced algorithm, we spend $O(\log r)$ time processing operations from the deletion/unrefinement queue. This works because not-yet-reclaimed storage and over-refined tree nodes do not impair the approximation quality or search performance of the data structure.

5.4. A lower bound

A simple example involving points evenly spaced along a circle shows that our error bound is the best possible.

Theorem 5.5. *If a planar set of points is represented by a sample of size r, the worst-case distance between a vertex of the true convex hull and the convex hull of the sample is $\Omega(D/r^2)$, where D is the diameter of the set.*

Proof. If a set S of $2r$ points is evenly spaced along a circle, and r points are chosen as a sample, then any point not chosen lies at distance $\Omega(D/r^2)$ from the convex hull of the sample. See Fig. 9. □

6. Processing extremal queries

The convex hull is a convenient data structure for point streams that can be used to answer a variety of extremal geometric queries. In the following, we briefly describe how to track or maintain some useful properties. In each case the cost of a query on the approximate convex hull is $O(\log r)$ or $O(r)$, even though the cost of building and

maintaining the approximate hull on a data stream is $O(\log r)$ per data stream point, for a total of $O(n \log r)$ time after n points have been processed (Theorem 5.4).

Diameter. The diameter of a point set is the maximum distance between any two points in the set. The diameter of the adaptively sampled hull estimates the diameter of the point stream at all times within additive error $O(D/r^2)$. There are standard algorithms in computational geometry for computing the diameter of a convex polygon in linear time [25], which is $O(r)$ for the adaptively sampled hull.

Width or Directional Extent. The width of a point set is the minimum distance between two parallel lines that enclose the point set between them. (Thus, the width can be viewed as the opposite of the diameter; the latter *maximizes* this quantity.) The width of a convex polygon can also be determined in linear time ($O(r)$ for the adaptively sampled hull). More generally, given a direction θ, the *extent* of the point set in direction θ can be defined as the minimum separation of two lines perpendicular to direction θ that enclose the convex hull. Given a direction θ, we can determine the directional extent in time $O(\log r)$.

The additive error in the width or directional extent is $O(D/r^2)$. This can be an arbitrarily poor approximation when the width or extent is much smaller than D. Since the original publication of this paper, Chan [5] has proposed another scheme that can maintain a $(1+\varepsilon)$ approximation of the width using $O(\frac{1}{\sqrt{\varepsilon}} \log \frac{1}{\varepsilon})$ space.

Linear Separation. Given approximate convex hulls for two point streams, A and B, we can track the approximate minimum distance between them, or determine when they are no longer approximately linearly separable. We build a secondary data structure on the convex hulls of A and B that allows minimum distance computation as well as insertion of new points in $O(\log r)$ time. (Whenever a new point appears on one of the convex hulls, we find its distance to the other convex hull by binary search in $O(\log r)$ time.) When the two hulls intersect, this distance becomes zero, and we can produce a certificate of non-separation, also in $O(\log r)$ time.

Containment. We can track or detect when all points of a stream are surrounded by points of another stream. With approximate convex hulls, we can track the containment, within error $O(D/r^2)$, in time $O(\log r)$ per point. Whenever a new point is added to the contained hull, we check whether it also lies in the containing hull or not, a test which can be done in $O(\log r)$ worst-case time.

Spatial Overlap. If the spatial extents of the two point streams are partially overlapping, we can use their approximate convex hulls to quantify the overlap. Given two convex polygons, there are linear time algorithms (i.e., $O(r)$) to compute their intersection [25].

It is easy to extend these queries to multiple streams, where we may want to track pairwise separation or other attributes. Many other natural geometric quantities, such as the farthest neighbor of any point, smallest circle containing all the points, etc., can also be computed from the approximate convex hull.

7. Experimental results

In this section, we describe our experimental results. We implemented both uniform sampling and adaptive sampling, and compared their performance over several synthetic data streams. The adaptive hull described in Sections 4 and 5 uses a variable number of sample directions (between r and $2r+1$). For a fair comparison, we modified our algorithm slightly to use a fixed number of sample directions ($=2r$). That is, the modified adaptive algorithm refines the maximum-weight edges until the number of sample directions is $2r$, even if that means refining some edges with weight $w(e) \leqslant 1$. We run uniform sampling with parameter $2r$, and the modified adaptive sampling with parameter r; they both maintain samples of size $2r$. (This modification to achieve equal-size approximations was simpler to implement than the alternative, which would be to maintain uniformly sampled hulls of all sizes between r and $2r+1$, then compare the adaptive hull against whichever uniform hull matches it in size.)

We compared the algorithms on data points drawn uniformly at random from three families of shapes: a disk, a square, and an ellipse with aspect ratio r. In each case we measured the uncertainty triangle height and the distance from the approximate hull to a data point outside it (both average and worst-case bounds). We also counted the number

Table 1
Experimental results, with $r = 32$ for the uniformly sampled hull, $r = 16$ for the adaptive and partially adaptive hulls, and $\theta_0 = 2\pi/r = \pi/8$. "Partial" refers to the partially adaptive scheme

	Uncertainty triangles				Max distance from hull		% Points outside hull	
	Max height		Average height					
	Uniform	Adaptive	Uniform	Adaptive	Uniform	Adaptive	Uniform	Adaptive
Disk	64	107	47	48	43	54	0.77	0.84
Square, rotated by								
0	30	22	8	5	11	4	0.16	0.07
$\theta_0/4$	489	84	195	10	13	6	0.35	0.12
$\theta_0/3$	439	90	176	21	13	4	0.35	0.09
$\theta_0/2$	30	27	11	7	11	11	0.17	0.11
Ellipse, rotated by								
0	174	38	35	8	77	19	19.54	2.44
$\theta_0/4$	417	38	47	9	146	19	36.00	2.50
$\theta_0/3$	387	44	45	10	141	21	33.96	2.42
$\theta_0/2$	174	23	35	8	77	11	19.54	1.94
Changing ellipse rotated by	Partial	Adaptive	Partial	Adaptive	Partial	Adaptive	Partial	Adaptive
0	238	50	76	14	100	22	13.14	1.78
$\theta_0/4$	724	57	119	13	201	28	52.57	2.43
$\theta_0/3$	844	64	136	13	215	31	58.44	2.26
$\theta_0/2$	958	53	152	14	229	27	65.34	2.92

of points (from 10^5 total) that fell outside the approximate hull. In all cases we used $r = 32$ for the uniformly sampled hull and $r = 16$ for the adaptive hull. Our adaptive hull was never much worse than the uniformly sampled hull, and usually much better.

The first line of Table 1 shows results for points chosen from a disk. The uniformly sampled hull is ideal for this distribution, and indeed our algorithm is 25% worse in worst-case error and 10% worse in number of points outside the approximate hull.

The second section of the table shows results for points in a square. Because the uniform sample directions match the boundaries of an axis-aligned square, the quality of the uniformly sampled hull is artificially enhanced. We considered slight rotations of the square to make the uniform sample directions less tuned to the data set. The results show that the uncertainty triangles for the uniformly sampled hull were five to ten times larger than those for the adaptive hull, even though the geometry of the square meant that the uncertainty triangles were relatively empty. Nevertheless, the uniformly sampled hull had two to four times as many data points outside as the adaptive hull.

The third section of the table shows results for points in an ellipse with aspect ratio 16. As in the square case, we tried several slight rotations of the ellipse to remove the uniform sample directions' natural alignment with the ellipse axes. In all of the metrics we measured, the adaptive hull is four to fourteen times better than the uniformly sampled hull. In the worst case, a remarkable 36% of the data points fall outside the uniformly sampled hull, versus 2.5% for the adaptive hull. Fig. 10 shows the uncertainty triangles for this case.

The fourth section of the table considers adaptive sampling on a changing distribution. Our continuously adaptive algorithm works much better on such data than a nonadaptive hull, or, worse yet, one adapted to the wrong distribution. The data stream consisted of 10^5 points from a near-vertically oriented ellipse, followed by 10^5 points from a near-horizontally oriented ellipse that completely contained the first ellipse. The adaptive algorithm completely changes its sample directions in the second half of the stream. In this case, we compared ourselves with a "partially adaptive" algorithm. This scheme is inspired by (a particularly bad example of) machine learning: we used the first half of the stream as a "training set" for an adaptive hull, then fixed the directions chosen for the training set while processing the second half of the stream. As one might expect, a nonadaptive hull whose extreme directions have been chosen based on the wrong distribution performs very poorly—in this case, roughly as poorly as a uniformly sampled hull with $r = 16$.

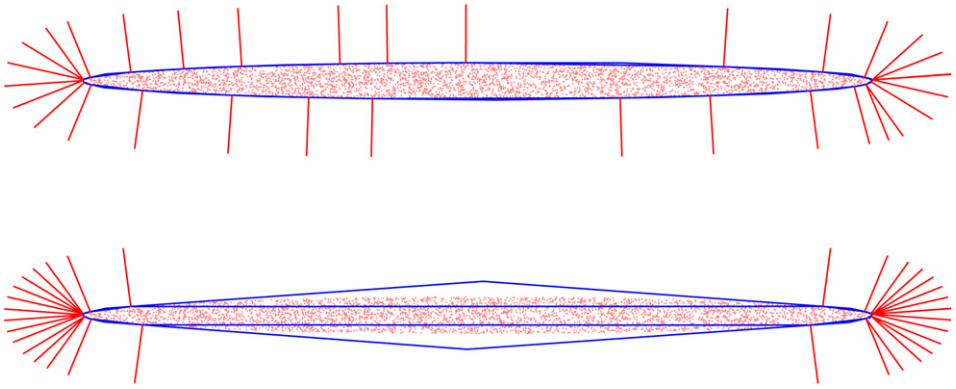

Fig. 10. The adaptive (above) and uniform (below) sample hulls for the case "ellipse rotated by $\theta_0/4$". (The figure has been rotated back for convenience of presentation.) The radial line segments show the sample directions, and the uncertainty triangles are drawn solid on top of the data points.

8. Discussion and extensions

We presented a provably optimal summary scheme for maintaining the convex hull of a two-dimensional point stream. Our adaptive sampling uses at most $2r + 1$ points, and maintains a hull that lies within distance $O(D/r^2)$ of the true convex hull, where D is the diameter of the sample set. This is asymptotically tight in the worst case. Our simulations show that the adaptive hull works better than the uniform hull in most cases; in extreme cases, such as the rotated ellipse or changing ellipse, the improvements in approximation quality are dramatic.

Many current and emerging technologies generate geometric streams. We believe it will become increasingly important to design algorithms and data structures to summarize such streams. Research in geometric streams is still in its early stages and much work remains to be done. Possible directions to explore include the following:

In some cases, a more refined "shape of the stream" may be desired than what the convex hull offers. For instance, if the points formed an "L" shape, then the convex hull approximation hides the cavity, returning a triangle instead. Is it possible to maintain a more detailed shape, which highlights cavities in the stream?

Also, suppose that the points naturally form multiple clusters, instead of a single cluster. If we have some *a priori* knowledge of the extent and separation of clusters, then we can easily maintain a separate convex hull for each cluster: partition the plane into disjoint regions such that points of one cluster fall within one region; then maintain separate approximate hulls for points in each region. Is it possible to achieve a similar result without knowing the spatial extents of different clusters? (Since the original conference publication of this paper, we have developed a streaming algorithm that combines clustering with approximate convex hulls [17], partially answering the questions raised above.)

Acknowledgements

Thanks to Nisheeth Shrivastava for implementing the algorithm and performing the experiments. Thanks also to Yossi Matias for suggesting the use of an approximate priority queue to speed up our algorithm's worst-case running time.

References

[1] P.K. Agarwal, S. Har-Peled, K.R. Varadarajan, Approximating extent measures of points, J. ACM 51 (4) (2004) 606–635.
[2] N. Alon, Y. Matias, M. Szegedy, The space complexity of approximating the frequency moments, J. Comput. Syst. Sci. 58 (1) (1999) 137–147.
[3] B. Babcock, S. Babu, M. Datar, R. Motwani, J. Widom, Models and issues in data stream systems, in: Proc. 21st ACM Sympos. Principles Database Syst., 2002.
[4] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, second ed., Springer-Verlag, Berlin, Germany, 2000.
[5] T. Chan, Faster core-set constructions and data stream algorithms in fixed dimensions, in: Proc. 20th Annu. ACM Sympos. Comput. Geom., 2004, pp. 152–159.
[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Introduction to Algorithms, MIT Press, Cambridge, MA, 1990.

[7] G. Cormode, S. Muthukrishnan, Radial histogram for spatial streams, Technical Report 2003-11, DIMACS, 2003.
[8] R.M. Dudley, Metric entropy of some classes of sets with differentiable boundaries, J. Approx. Theory 10 (1974) 227–236.
[9] http://www.earthscope.org.
[10] P. van Emde Boas, Preserving order in a forest in less than logarithmic time and linear space, Inf. Process. Lett. 6 (3) (1977) 80–82.
[11] P. van Emde Boas, R. Kaas, E. Zijlstra, Design and implementation of an efficient priority queue, Math. Systems Theory 10 (1977) 99–127.
[12] J. Feigenbaum, S. Kannan, M. Strauss, M. Viswanathan, An approximate L^1-difference algorithm for massive data streams, in: IEEE Sympos. Found. Comput. Sci., 1999, pp. 501–511.
[13] J. Feigenbaum, S. Kannan, J. Zhang, Computing diameter in the streaming and sliding-window models, 2002. Manuscript.
[14] J. Gray, A. Szalay, A. Thakar, P.Z. Zunszt, T. Malik, J. Raddick, C. Stoughton, J. van den Berg, The SDSS SkyServer: Public access to the Sloan digital sky server data, Technical Report MSR-TR-2001-104, Microsoft Research, 2001.
[15] M. Greenwald, S. Khanna, Space-efficient online computation of quantile summaries, in: Proc. ACM SIGMOD Intl. Conf. on Management of Data, 2001.
[16] P.M. Gruber, Approximation of convex bodies, in: P.M. Gruber, J.M. Wills (Eds.), Convexity and its Applications, Birkhäuser, Basel, Switzerland, 1983, pp. 131–162.
[17] J. Hershberger, N. Shrivastava, S. Suri, Summarizing spatial data streams using ClusterHulls, in: Proc. 8th Workshop on Algorithm Engineering and Experiments (ALENEX), 2006, pp. 26–40.
[18] P. Indyk, Stream-based geometric algorithms, in: Proc. ACM/DIMACS Workshop on Management and Processing of Data Streams, 2003.
[19] N.M. Josuttis, The C++ Standard Library: A Tutorial and Reference, Addison-Wesley, Reading, MA, 1999.
[20] P.S. Kenderov, Polygonal approximation of plane convex compacts, J. Approx. Theory 38 (1983) 221–239.
[21] G.S. Manku, S. Rajagopalan, B.G. Lindsay, Random sampling techniques for space efficient online computation of order statistics of large datasets, in: Proc. ACM SIGMOD Intl. Conf. on Management of Data, 1999.
[22] Y. Matias, J.S. Vitter, N. Young, Approximate data structures with applications, in: Proc. 5th ACM-SIAM Sympos. Discrete Algorithms, 1994, pp. 187–194.
[23] D.E. McClure, R.A. Vitale, Polygonal approximation of plane convex bodies, J. Math. Anal. Appl. 51 (1975) 326–358.
[24] S. Muthukrishnan, Data streams: Algorithms and applications, Technical report, Computer Science Department, Rutgers University, 2003.
[25] F.P. Preparata, M.I. Shamos, Computational Geometry: An Introduction, Springer-Verlag, New York, 1985.
[26] W. Pugh, Skip lists: a probabilistic alternative to balanced trees, Commun. ACM 33 (6) (1990) 668–676.
[27] T.J. Richardson, Approximation of planar convex sets from hyperplane probes, Discrete Comput. Geom. 18 (2) (1997) 151–177.