# Practical Methods for Shape Fitting and Kinetic Data Structures using Coresets*

Hai Yu† Pankaj K. Agarwal† Raghunath Poreddy‡ Kasturi R. Varadarajan‡

**Abstract**

The notion of $\varepsilon$-kernel was introduced by Agarwal *et al.* [5] to set up a unified framework for computing various extent measures of a point set $P$ approximately. Roughly speaking, a subset $Q \subseteq P$ is an $\varepsilon$-kernel of $P$ if for every slab $W$ containing $Q$, the expanded slab $(1+\varepsilon)W$ contains $P$. They illustrated the significance of $\varepsilon$-kernel by showing that it yields approximation algorithms for a wide range of geometric optimization problems.

We present a simpler and more practical algorithm for computing the $\varepsilon$-kernel of a set $P$ of points in $\mathbb{R}^d$. We demonstrate the practicality of our algorithm by showing its empirical performance on various inputs. We then describe an incremental algorithm for fitting various shapes and use the ideas of our algorithm for computing $\varepsilon$-kernels to analyze the performance of this algorithm. We illustrate the versatility and practicality of this technique by implementing approximation algorithms for minimum enclosing cylinder, minimum-volume bounding box, and minimum-width annulus. Finally, we show that $\varepsilon$-kernels can be effectively used to expedite the algorithms for maintaining extents of moving points.

## 1 Introduction

Motivated by numerous applications, considerable work has been done on computing various descriptors of the extent of a set $P$ of $n$ points in $\mathbb{R}^d$. These measures, called *extent measures*, either compute certain statistics of $P$ itself or compute certain statistics of a (possibly nonconvex) geometric shape (e.g., sphere, box, cylinder, etc.) enclosing $P$. Examples of the former include computing the $k$th largest distance between pairs of points in $P$, and the examples of the latter include computing the smallest radius of a sphere (or cylinder), the minimum volume (or surface area) of a box, and the smallest width of a slab (i.e., the region enclosed by two parallel hyperplanes) or a spherical or cylindrical shell that contain $P$. Although traditionally $P$ is assumed to be stationary (insertion/deletion of points at discete times have been considered), recently there has been work on maintaining extent measures of a set of moving points [4]. In the latter, the goal is to maintain the extent measure as the points move, e.g., using the kinetic data structure framework [12].

The exact algorithms for computing extent measures are generally expensive, e.g., for $d = 3$, the best known algorithm for computing the smallest simplex containing $P$ requires $O(n^4)$ time [30], and the smallest enclosing cylindrical shell requires $O(n^5)$ time [3]. Consequently, attention has shifted to developing approximation algorithms, and several approximation algorithms for specific problems exist, see [1, 3, 11, 15, 30] and the references therein. Although these algorithms are tailored to specific problems,

*A preliminary version of the paper appeared in *Proceedings of the 20th Annual ACM Symposium on Computational Geometry*, 2004, pp. 263–272. Research by the first two authors is supported by NSF under grants CCR-00-86013, EIA-98-70724, EIA-01-31905, and CCR-02-04118, and by a grant from the U.S.–Israel Binational Science Foundation. Research by the fourth author is supported by NSF CAREER award CCR-0237431.

†Department of Computer Science, Duke University, Durham, NC 27708-0129, USA. {*fishhai,pankaj*}*@cs.duke.edu*

‡Department of Computer Science, University of Iowa, Iowa City, IA 52242-1419, USA. {*rporeddy,kvaradar*}*@cs.uiowa.edu*

they rely on similar techniques. A natural open question is thus whether a unified framework exists for computing extent measures approximately. Addressing this issue, Agarwal *et al.* [5] introduced the notion of $\varepsilon$-*kernel* for a point set $P$. Roughly speaking, a subset $Q \subseteq P$ is an $\varepsilon$-kernel of $P$ if for every slab $W$ containing $Q$, the expanded slab $(1+\varepsilon)W$ contains $P$. They presented an $O(n + 1/\varepsilon^{d-1})$-time algorithm for computing an $\varepsilon$-kernel of $P$ of size $O(1/\varepsilon^{d-1})$ and an $O(n+1/\varepsilon^{3(d-1)/2})$-time algorithm for computing an $\varepsilon$-kernel of size $O(1/\varepsilon^{(d-1)/2})$. They also introduced the notion of $\varepsilon$-kernel for a set $\mathcal{F}$ of functions as follows. The *extent* of $\mathcal{F}$ at a point $x \in \mathbb{R}^{d-1}$ is

$$\mathfrak{E}_{\mathcal{F}}(x) = \max_{f\in\mathcal{F}} f(x) - \min_{f\in\mathcal{F}} f(x), \tag{1}$$

and a subset $\mathcal{G} \subseteq \mathcal{F}$ is an $\varepsilon$-*kernel* of $\mathcal{F}$ if $\mathfrak{E}_{\mathcal{G}}(x) \geq (1-\varepsilon)\mathfrak{E}_{\mathcal{F}}(x)$ for all $x \in \mathbb{R}^{d-1}$. Using their result on computing the $\varepsilon$-kernel of point sets and the linearization technique, they compute in $O(n + 1/\varepsilon^{O(1)})$ time an $\varepsilon$-kernel of $\mathcal{F}$ of size $O(1/\varepsilon^{r\sigma})$ if each $f_i \in \mathcal{F}$ is of the form $g_i^{1/r}$, where $g_i$ is a polynomial, $r$ is a positive integer, $\sigma = \min\{d-1, k/2\}$, and $k$ is the dimension of linearization for $g_i$'s (see the original paper for the definition of $k$). They illustrated the power of their technique by showing that it yields approximation algorithms for a wide range of geometric optimization problems, including algorithms for minimum-width spherical and cylindrical shells containing $P$; kinetic data structures for maintaining approximate extent measures of moving points, which process $1/\varepsilon^{O(1)}$ events; and maintaining extent measures in a streaming model in polylog$(n)$ time. No algorithms with similar running time were previously known for these problems.

Many subsequent papers have used a similar approach for other geometric optimization problems, including clustering and other extent-measure problems [6, 8, 13, 14, 22, 23, 24, 25, 26]. These approaches compute a subset $Q \subseteq P$ of small size and solve the underlying optimization problem for $Q$. The term *coreset* is now commonly used to refer to such a subset.

Although the algorithm by Agarwal *et al.* [5] is conceptually simple, it is not practical. It works in two stages: the first stage computes an $(\varepsilon/2)$-kernel $Q'$ of size $O(1/\varepsilon^{d-1})$, and the second stage computes an $(\varepsilon/3)$-kernel $Q$ of $Q'$ of size $O(1/\varepsilon^{(d-1)/2})$. The second stage invokes $O(1/\varepsilon^{(d-1)/2})$ times Gärtner's randomized algorithm [18] (or a deterministic counterpart) that computes the face of the convex hull $\mathrm{conv}(Q')$ closest to a given point lying outside $\mathrm{conv}(Q')$. The deterministic counterparts are rather complicated and even Gärtner's randomized algorithm is not that simple and efficient. We present a simpler and more practical algorithm for computing an $\varepsilon$-kernel of a set of points that needs only an approximate nearest-neighbor-searching data structure. The worst-case running time of the simplest version of this algorithm is the same as that of Agarwal *et al.* [5].[1] Using the duality transform and the linearization technique, as described in Agarwal *et al.* [5], we can then compute $\varepsilon$-kernels for a set of linear functions, polynomials, or their roots.

We have implemented our $\varepsilon$-kernel algorithm, using the ANN library for answering approximate nearest-neighbor queries [28], and tested on a variety of inputs in dimension up to 8. The empirical results show that our algorithm works extremely well in low dimensions ($\leq 4$) both in terms of the size of the kernel and the running time. For example, to achieve an error below 0.05, the size of the $\varepsilon$-kernel is usually few hundreds and the running time is just a few seconds. The size and running time increase exponentially with the dimension.

In Section 3 we study the extent measures that arise in shape fitting. As in [5], we can simply plug our $\varepsilon$-kernel algorithm to compute various shape-fitting based extent measures in time $O(n + 1/\varepsilon^{O(1)})$. However, we present a very simple incremental algorithm for these problems. We analyze the number of iterations of the algorithm for the case of minimum-width slab, by exploiting the argument for the correctness of our algorithm for computing $\varepsilon$-kernels. The analysis extends to other shape-fitting problems, including

[1] Chan [16] has independently observed a similar simplification, and developed an algorithm for computing an $\varepsilon$-kernel of size $O(1/\varepsilon^{(d-1)/2})$ whose running time is $O(n + 1/\varepsilon^{d-3/2})$. Our focus is on the empirical study of our simplified algorithm.

minimum-width spherical and cylindrical shell, minimum-radius cylinder and the minimum-volume box enclosing a given set $P$ of points. Although the worst-case bound on number of iterations of this algorithm for some of the cases is weaker than a more tailored algorithm for that specific case (e.g., an incremental algorithm that computes a coreset for the smallest enclosing ball in any fixed dimension in $O(1/\varepsilon)$ iterations is developed by [14], while our algorithm takes $O(1/\varepsilon^{(d-1)/2})$ iterations), our empirical results show that in practice the algorithm converges in very few iterations. Our approach is reminiscent of Clarkson's linear-programming algorithm [17]. We illustrate the versatility of the technique by implementing approximation algorithms for minimum enclosing cylinder, minimum-volume bounding box, and minimum-width annulus. We compare our incremental algorithm for these problems with the natural algorithm of computing a kernel first and then solving the problem on the kernel. Our experiments confirm our expectation that quite often the number of iterations of the incremental algorithm is small and so it outperforms the latter, though in the worst case the incremental algorithm can be somewhat slower than the latter. Note that the incremental algorithm produces an $\varepsilon$-approximate solution in time linear in the number of points. The existence of such a natural and generic algorithm is quite interesting from the theoretical perspective also, particularly because these shape fitting problems have received considerable attention.

In Section 4 we apply our $\varepsilon$-kernel algorithm to maintain an approximation to the minimum orthogonal bounding box and the convex hull of a set of points moving in $\mathbb{R}^2$. We show that our algorithm processes very few events and maintains a good approximation of the extent as the points move. For example, for a set of 100,000 quadratically moving points, we were able to maintain an approximate minimum orthogonal bounding box with error below 0.05 over time by choosing a kernel of size 80. These experimental results affirmed the significance of $\varepsilon$-kernels to kinetic data structures in practice.

## 2 Computing an $\varepsilon$-Kernel

In this section we describe our algorithm for computing the $\varepsilon$-kernel of any set $P$ of points in $\mathbb{R}^d$, analyze its running time, and provide empirical results.

**Geometric preliminaries.** We first define various terms that we will need later, most of which are taken from [5]. Let $\mathbb{S}^{d-1}$ denote the unit sphere centered at the origin in $\mathbb{R}^d$. For any set $P$ of points in $\mathbb{R}^d$ and any direction $u \in \mathbb{S}^{d-1}$, we define the *directional width* of $P$ in direction $u$, denoted by $\omega(u, P)$, to be

$$\omega(u, P) = \max_{p \in P} \langle u, p \rangle - \min_{p \in P} \langle u, p \rangle,$$

where $\langle \cdot, \cdot \rangle$ is the inner product. Let $\varepsilon > 0$ be a parameter. A subset $Q \subseteq P$ is called an *$\varepsilon$-kernel* of $P$ if for each $u \in \mathbb{S}^{d-1}$,

$$\omega(u, Q) \geq (1 - \varepsilon)\omega(u, P).$$

Clearly, $\omega(u, Q) \leq \omega(u, P)$.

We call $P$ *$\alpha$-fat*, for $\alpha \leq 1$, if there exists a point $p \in \mathbb{R}^d$ and a hypercube $\overline{\mathbb{C}}$ centered at the origin so that

$$p + \alpha\overline{\mathbb{C}} \subset \mathrm{conv}(P) \subset p + \overline{\mathbb{C}}.$$

A stronger version of the following lemma was proved in [5].

**Lemma 2.1** *Let $P$ be a set of $n$ points in $\mathbb{R}^d$, and let $\mathbb{C} = [-1, 1]^d$. One can compute in $O(n)$ time an affine transform $\tau$ so that $\tau(P)$ is an $\alpha$-fat point set satisfying $\alpha\mathbb{C} \subset \mathrm{conv}(\tau(P)) \subset \mathbb{C}$, where $\alpha$ is a constant depending on $d$, and so that $Q$ is an $\varepsilon$-kernel of $P$ if and only if $\tau(Q)$ is an $\varepsilon$-kernel of $\tau(P)$.*

**Algorithm.** In view of Lemma 2.1, we can assume that $P$ is an $\alpha$-fat point set so that $\alpha\mathbb{C} \subset \mathrm{conv}(P) \subset \mathbb{C}$, where $\mathbb{C} = [-1,1]^d$. Suppose $\varepsilon \leq 1/4$. Let $\mathbf{S}$ be the sphere of radius $\sqrt{d}+1$ centered at the origin. Set $\lambda = \sqrt{\varepsilon\alpha} \leq 1/2$. It is well known that one can construct a set $\mathcal{I}$ of $O(1/\lambda^{d-1}) = O(1/\varepsilon^{(d-1)/2})$ points on the sphere $\mathbf{S}$ such that for any point $x$ on $\mathbf{S}$, there exists a point $y \in \mathcal{I}$ such that $\|x-y\| \leq \lambda$ (see [29]). We process $P$ into a data structure that can answer $\varepsilon$-approximate nearest-neighbor queries. For a query point $q$, let $\varphi(q)$ be the point of $P$ returned by this data structure. For each point $y \in \mathcal{I}$, we compute $\varphi(y)$ using this data structure. We return the set $Q = \{\varphi(y) \mid y \in \mathcal{I}\}$.

**Theorem 2.2** *The set $Q$ is an $\varepsilon$-kernel of $P$ of size $O(1/\varepsilon^{(d-1)/2})$.*

*Proof:* For simplicity, we first prove the claim under the assumption that $\varphi(y)$ is the *exact* nearest-neighbor of $y$ in $P$.

Fix a direction $u \in \mathbb{S}^{d-1}$. Let $\sigma \in P$ be the point that maximizes $\langle u, p\rangle$ over all $p \in P$, i.e., $\sigma = \arg\max_{p\in P}\langle u, p\rangle$. Suppose the ray emanating from $\sigma$ in direction $u$ hits $\mathbf{S}$ at a point $x$; see Figure 1. We know that there exists a point $y \in \mathcal{I}$ such that $\|x-y\| \leq \lambda$.

If $\varphi(y) = \sigma$, then $\sigma \in Q$ and

$$\max_{p\in P}\langle u, p\rangle - \max_{q\in Q}\langle u, q\rangle = 0.$$

Now suppose $\varphi(y) \neq \sigma$. Let $B$ be the $d$-dimensional ball of radius $||y-\sigma||$ centered at $y$. Clearly, $\varphi(y) \in B$. Let $z$ be the point at which the sphere $\partial B$ is hit by the ray emanating from $y$ in direction $-u$. Let $w$ be the point on $zy$ such that $zy \perp \sigma w$ and $h$ the point on $\sigma x$ such that $yh \perp \sigma x$; see Figure 1.

The hyperplane normal to $u$ and passing through $z$ is tangent to $B$. Since $\varphi(y)$ lies inside $B$, $\langle u, \varphi(y)\rangle \geq \langle u, z\rangle$. Moreover, we have

$$\begin{aligned}
\langle u,\sigma\rangle - \langle u,\varphi(y)\rangle &\leq \langle u,\sigma\rangle - \langle u,z\rangle = \|z-w\| \\
&= \|\sigma - w\|\tan(\angle z\sigma w) \\
&= \|\sigma - w\|\tan(\angle y\sigma x/2) \\
&\leq (1/2)\cdot\|x-y\|\cdot\tan(\angle y\sigma x) \\
&= (1/2)\cdot\|x-y\|\cdot\|y-h\|/\|\sigma-h\| \\
&\leq (1/2)\cdot\|x-y\|^2/\|\sigma-h\| \\
&\leq (1/2)\cdot\|x-y\|^2/(\|\sigma-x\| - \|x-y\|) \\
&\leq (1/2)\cdot\lambda^2/(1-\lambda) \qquad (\|\sigma-x\| \geq 1) \\
&\leq \lambda^2 = \alpha\varepsilon \qquad (\lambda \leq 1/2), \qquad (2)
\end{aligned}$$

where in the fourth inequality, we have used the fact that $\angle y\sigma x < \pi/2$ (since $\|\sigma - x\| \geq 1$ and $\|x-y\| \leq 1/2$), and $\tan(\theta/2) \leq \tan(\theta)/2$ for any $\theta < \pi/2$. Thus, we can write

$$\max_{p\in P}\langle u, p\rangle - \max_{q\in Q}\langle u, q\rangle \leq \langle u,\sigma\rangle - \langle u,\varphi(y)\rangle \leq \alpha\varepsilon.$$

Similarly, we have $\min_{p\in P}\langle u, p\rangle - \min_{q\in Q}\langle u, q\rangle \geq -\alpha\varepsilon$.

The above two inequalities together imply that $\omega(u,Q) \geq \omega(u,P) - 2\alpha\varepsilon$. Since $\alpha\mathbb{C} \subset \mathrm{conv}(P)$, $\omega(u,P) \geq 2\alpha$. Hence $\omega(u,Q) \geq (1-\varepsilon)\omega(u,P)$, for any $u \in \mathbb{S}^{d-1}$. This implies that $Q$ is an $\varepsilon$-kernel of $P$.

We next sketch how to argue $Q$ is a $(2\varepsilon)$-kernel of $P$ if $\varphi(y)$ is a $\delta$-approximate nearest-neighbor of $y$ in $P$, where $\delta = \alpha\varepsilon/2(\sqrt{d}+1) = O(\varepsilon)$. Let $B'$ be a $\delta$-expansion of $B$, i.e., a ball of radius $(1+\delta)\cdot\|y-\sigma\|$

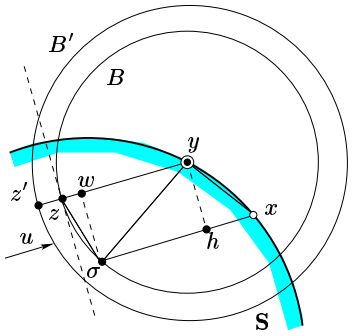



**Figure 1.** Correctness of the algorithm.

centered at $y$, and let $z'$ be the point at which $\partial B'$ is hit by the ray emanating from $y$ in direction $-u$. Since $\varphi(y) \in B'$, it follows from our above discussion that

$$\begin{aligned}\max_{p\in P}\langle u,p\rangle - \max_{q\in Q}\langle u,q\rangle &\leq \langle u,\sigma\rangle - \langle u,z'\rangle \\ &= \langle u,\sigma\rangle - \langle u,z\rangle + \|z-z'\| \\ &\leq \alpha\varepsilon + \delta\cdot\|y-\sigma\| \leq 2\alpha\varepsilon.\end{aligned}$$

The last inequality follows from the fact that $\|y-\sigma\| \leq 2(\sqrt{d}+1)$. Similarly, $\min_{p\in P}\langle u,p\rangle - \min_{q\in Q}\langle u,q\rangle \geq -2\alpha\varepsilon$. Therefore, following a similar argument as given above, we have $\omega(u,Q) \geq (1-2\varepsilon)\omega(u,P)$ as desired. □

**Running time.** Note that a straightforward implementation (the one that answers a nearest-neighbor query by comparing the distances to all the points) of the algorithm runs in $O(n/\varepsilon^{(d-1)/2})$ time. By first computing an $(\varepsilon/2)$-kernel $Q'$ of $P$ of size $O(1/\varepsilon^{d-1})$, as in Agarwal *et al.* [5], and then applying the above algorithm to compute an $(\varepsilon/3)$-kernel $Q$ of $Q'$, we compute an $\varepsilon$-kernel $Q$ of $P$ in $O(n + 1/\varepsilon^{3(d-1)/2})$ time. Although this running time is the same as the one in Agarwal *et al.* [5], our algorithm is more practical than theirs, which, for each $y \in \mathcal{I}$, needs to compute the nearest face on the convex hull of $Q'$ by solving an abstract optimization problem [18]. In practice our algorithm may take advantage of known approximate nearest-neighbor searching data structures, such as BBD-trees [10], and the running time would be bounded by $O(n\log n + \log n/\varepsilon^{(3d-1)/2})$, without computing an $(\varepsilon/2)$-kernel first. In fact, using a data structure described in [16], the running time can be improved to $O(n + 1/\varepsilon^{d-3/2})$.

**Experimental results.** We implemented our $\varepsilon$-kernel algorithm and tested its performance on a variety of inputs. We measured the quality of an $\varepsilon$-kernel $Q$ of $P$ as the maximum relative error in the directional width of $\tau(P)$ and $\tau(Q)$. Since it is hard to compute the maximum error over all directions, we sampled a set $\Delta$ of 1000 directions in $\mathbb{S}^{d-1}$ and computed the maximum relative error with respect to these directions, i.e.,

$$\mathrm{err}(Q,P) = \max_{u\in\Delta}\frac{\omega(u,\tau(P)) - \omega(u,\tau(Q))}{\omega(u,\tau(P))}. \tag{3}$$

We performed two sets of experiments:

(i) We fixed a real value $\varepsilon$, and checked how the kernel size increased, as a function of the input size and dimension, to ensure that $\mathrm{err}(Q,P) \leq \varepsilon$. We fixed $\varepsilon$ to be $0.05$ in our experiments.

(ii) We fixed the input size and checked how $\mathrm{err}(Q,P)$ decreased as the kernel size increased.

We implemented the constant-factor approximation algorithm by Barequet and Har-Peled [11] for computing the minimum-volume bounding box to convert $P$ into an $\alpha$-fat set. In order to generate $\mathcal{I}$, one simple approach is to use the standard coordinate frame on $\mathbb{S}^{d-1}$, choose a parameter $k$, and draw a $\underbrace{k \times \cdots \times k}_{d-1}$ grid on $\mathbb{S}^{d-1}$. There are two weaknesses of this approach — the grid is not regular and the size of $\mathcal{I}$ is of the form $c \cdot k^{d-1}$ (i.e., not arbitrary). Therefore, we implemented a better routine for uniformly sampling an arbitrary number of $k$ points from $\mathbb{S}^{d-1}$. It works by beginning with an arbitrary set of $k$ points on the sphere, regarding each point as an electron, and then using the gradient descent method to minimize the total potential energy of the point set by repeatedly fine-tuning the position of each point. The final configuration tends to be a regular distribution on the sphere. The weakness of this approach is that it is somewhat slow in high dimensions or when $k$ is large. Finally, we use the ANN library [28] for answering approximate nearest-neighbor queries, and we set its relative error to $0.01$.

We used three different types of synthetic inputs in dimensions 2–8 and a few large 3D geometric models [27]:

(i) points uniformly distributed on a sphere (*sphere*);

(ii) points uniformly distributed on a cylindrical surface (*cylinder*);

(iii) clustered point sets (*clustered*), consisting of 20 equal-sized clusters whose centers are uniformly distributed in the unit square and radii uniformly distributed between $[0, 0.2]$;

(iv) 3D geometric models: *bunny* ($\sim 36K$ points), *dragon* ($\sim 438K$ points), and *buddha* ($\sim 544K$ points),

Since the algorithm chooses mostly convex hull vertices as kernels, we used convex sets for synthetic data *sphere* and *cylinder*. Input points lying on a sphere can be regarded as the worst-case example. The relative size of the kernel would be smaller for arbitrary point sets (see, e.g., *clustered*).

All experiments of this paper were conducted on a Dell PowerEdge 650 server with a 3.06GHz Pentium IV processor and 3GB memory, running Linux 2.4.20. Running time is measured in seconds. Tables 1–3 show our results on the first experiment, and Figure 2 shows the results of the second experiment. To ensure that $\varepsilon < 0.05$ in each case of the first experiment, we tried different $\mathcal{I}$ with arithmetically increasing sizes, and chose the first one that produced a kernel whose approximation error (evaluated via (3)) is less than $0.05$. Note that the size of the output kernel is close to but not necessarily equal to $|\mathcal{I}|$. In Table 1, we have decomposed the running time into two components: (i) preprocessing time and (ii) query time. The former includes the time spent in converting $P$ into a fat set and in preprocessing $P$ for approximate nearest-neighbor queries; it increases linearly with the size of the input. The latter includes the time spent in computing $\varphi(x)$ for $x \in \mathcal{I}$, and it depends on the size of $\mathcal{I}$. As can be seen, our algorithm worked extremely well in low dimensions ($\leq 4$) both in terms of the output size and the running time. For example, to achieve an error below 0.05, the size of the $\varepsilon$-kernel is usually few hundreds and the running time is just a few seconds (except for the case of sphere). As Table 3 illustrates, the performance is even better on 3D geometric models. In higher dimensions, both our algorithm and the ANN library ran into the *curse of dimensionality*: the size and running time grow exponentially in $d$. However, we expect that it might still be useful as a first prune in practice, and the running time may be improved by a better tradeoff between the preprocessing and query time (e.g., using approximate Voronoi diagrams [9, 20] for answering approximate nearest-neighbor queries). Our algorithm still performed well on the input *clustered* in dimensions 6 and 8. Figure 2 shows that, as the size of the computed $\varepsilon$-kernel becomes larger, its quality improves drastically while the kernel size is small, e.g., in the range $[0, 200]$, but improves very gradually afterwards. This phenomenon suggests that in practice it is most beneficial to keep the sizes of $\varepsilon$-kernels small.

| *Input Type* | *Input Size* | $d = 2$ | | $d = 4$ | | $d = 6$ | | $d = 8$ | |
|---|---|---|---|---|---|---|---|---|---|
| | | *Prepr* | *Query* | *Prepr* | *Query* | *Prepr* | *Query* | *Prepr* | *Query* |
| *sphere* | 10,000 | 0.02 | 0.01 | 0.03 | 0.02 | 0.05 | 4.40 | 0.07 | 32.44 |
| | 100,000 | 0.43 | 0.01 | 0.61 | 0.09 | 0.85 | 37.56 | 0.95 | 945.69 |
| | 1,000,000 | 8.36 | 0.01 | 10.55 | 1.23 | 14.39 | 150.21 | 17.75 | 3846.47 |
| *cylinder* | 10,000 | 0.02 | 0.01 | 0.03 | 0.01 | 0.05 | 1.54 | 0.06 | 12.57 |
| | 100,000 | 0.47 | 0.01 | 0.61 | 0.07 | 0.82 | 25.83 | 0.96 | 783.48 |
| | 1,000,000 | 9.16 | 0.01 | 10.44 | 0.17 | 14.73 | 63.37 | 17.62 | 3022.17 |
| *clustered* | 10,000 | 0.02 | 0.01 | 0.03 | 0.01 | 0.04 | 0.02 | 0.06 | 0.21 |
| | 100,000 | 0.20 | 0.01 | 0.35 | 0.01 | 0.55 | 0.18 | 0.79 | 1.61 |
| | 1,000,000 | 4.30 | 0.01 | 5.95 | 0.02 | 9.13 | 0.45 | 12.90 | 4.48 |

**Table 1.** Running time for computing $\varepsilon$-kernels of various synthetic data sets in different dimensions, $\varepsilon < 0.05$. *Prepr* denotes the preprocessing time, including converting $P$ into a fat set and building ANN data structures, but not including the time for generating $\mathcal{I}$. *Query* denotes the time for performing approximate nearest-neighbor queries. Running time is measured in seconds.

| *Input Type* | *Input Size* | *Kernel Size* | | | |
|---|---|---|---|---|---|
| | | $d = 2$ | $d = 4$ | $d = 6$ | $d = 8$ |
| *sphere* | 10,000 | 10 | 300 | 4,923 | 6,473 |
| | 100,000 | 10 | 300 | 8,364 | 18,376 |
| | 1,000,000 | 10 | 300 | 9,532 | 20,437 |
| *cylinder* | 10,000 | 6 | 232 | 3,834 | 6,021 |
| | 100,000 | 6 | 293 | 7,842 | 15,413 |
| | 1,000,000 | 6 | 298 | 8,727 | 17,465 |
| *clustered* | 10,000 | 8 | 115 | 400 | 1,160 |
| | 100,000 | 12 | 172 | 670 | 2,226 |
| | 1,000,000 | 12 | 189 | 710 | 2,550 |

**Table 2.** Sizes of $\varepsilon$-kernels for various synthetic data sets in different dimensions, $\varepsilon < 0.05$.

| *Input Type* | *Input Size* | *Running Time* | | *Kernel Size* | *Diameter Error* |
|---|---|---|---|---|---|
| | | *Prepr* | *Query* | | |
| *bunny* | 35,947 | 0.17 | 0.01 | 67 | 0.010 |
| *dragon* | 437,645 | 2.44 | 0.01 | 69 | 0.004 |
| *buddha* | 543,652 | 2.87 | 0.01 | 68 | 0.010 |

**Table 3.** Performance on various 3D geometric models with $\varepsilon < 0.05$. We also computed the relative error in $\mathrm{diam}(Q)$ and $\mathrm{diam}(P)$ as another measure of the quality of $Q$. Running time is measured in seconds.

# 3 Shape Fitting

Let $P$ be a set of $n$ points in $\mathbb{R}^d$. In this section, we describe a very simple incremental algorithm for fitting a simple shape through $P$. We show its versatility and efficiency by applying it to: (i) minimum enclosing cylinder in $\mathbb{R}^3$, (ii) minimum-width annulus in $\mathbb{R}^2$, and (iii) minimum-volume bounding box in $\mathbb{R}^3$. Our algorithm is much faster in most cases compared with those ones based on $\varepsilon$-kernels directly [5].

## 3.1 An incremental algorithm

Let $\Pi$ be an infinite family of simple shapes, e.g., set of all cylinders, set of all boxes, or set of all annuli. Let $\mu : \Pi \to \mathbb{R}$ be a measure function, and let $A_{\mathrm{opt}}(R, \Pi)$ be an algorithm that returns an optimal shape in $\Pi$ enclosing a set $R$ of points, i.e.,

$$A_{\mathrm{opt}}(R, \Pi) = \arg \inf_{\pi \in \Pi, R \subseteq \pi} \mu(\pi).$$

Assume that $\Pi$ is closed under translation and uniform scaling. For a shape $\pi \in \Pi$, let $(1 + \varepsilon)\pi$ denote the scaled version of $\pi$. We here assume that each shape in $\Pi$ has a center with respect to which we perform the

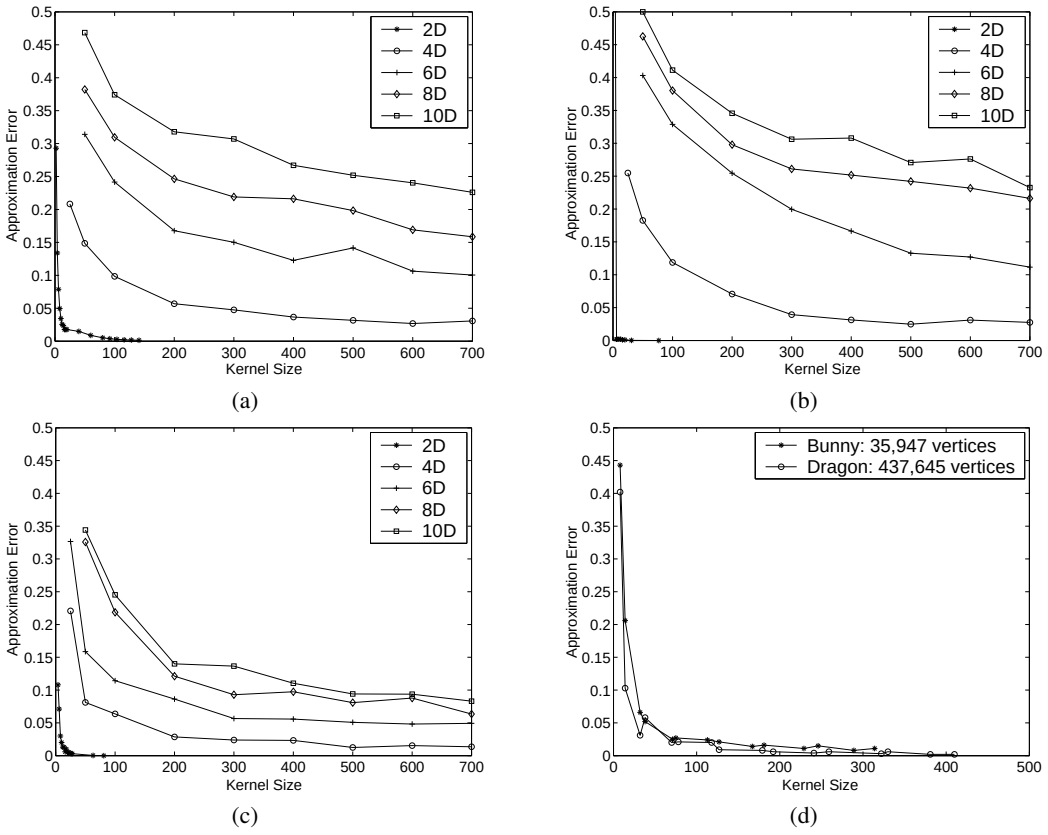



**Figure 2.** Approximation errors under different sizes of computed $\varepsilon$-kernels. (a) *sphere*, (b) *cylinder*, (c) *clustered*, and (d) various geometric models. All synthetic inputs had 100,000 points.

scaling.

APPROX_SHAPE_FITTING $(P, \Pi, \varepsilon)$

$R \subseteq P$: arbitrary set of size $\min\{|P|, c\}$

**while** $(\pi := (1+\varepsilon)A_{\text{opt}}(R, \Pi)) \not\supseteq P$

$p = \arg\max_{q \in P} d(q, \pi)$

$R = R \cup \{p\}$

**end while**

**Figure 3.** An incremental algorithm for shape fitting.

Figure 3 describes the incremental algorithm that computes an $\varepsilon$-approximation of $A_{\text{opt}}(P, \Pi)$ for an input point set $P$, i.e., it returns a shape $\pi \in \Pi$ such that $P \subseteq \pi$ and $\mu(\pi) \leq \mu\left((1+\varepsilon)A_{\text{opt}}(P, \Pi)\right)$. See Figure 4 for an illustration of a few iterations of the algorithm. In the algorithm, $c$ is a constant depending on $\Pi$ that, roughly speaking, is the number of parameters required to specify an element in $\Pi$. For example, if $\Pi$ is a set of annuli in $\mathbb{R}^2$ (resp. boxes, cylinders, slabs in $\mathbb{R}^3$), then $c$ is 4 (resp. 6, 5, 4). The third line in the algorithm means that $p \in P$ is the point that requires $\pi$ to be scaled the most in order to be covered.

As observed in [5], many shape fitting problems can be formulated as computing the minimum extent of an appropriate set of $k$-variate functions $\mathcal{F} = \{f_1, \cdots, f_n\}$ over a domain $\Gamma \subseteq \mathbb{R}^k$. Figure 5 reformulates

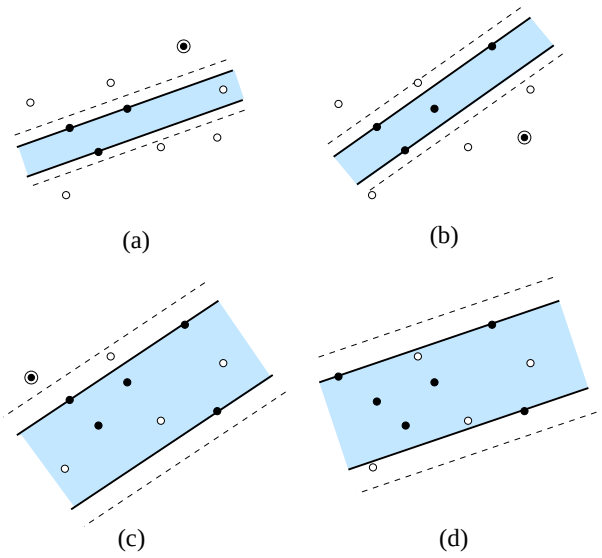



**Figure 4.** A few iterations of the incremental algorithm for computing the minimum-width slab in $\mathbb{R}^2$. Filled circles are points in $R$, and the double circle is the point that is being added to $R$.

the above incremental algorithm for shape fitting into an equivalent incremental algorithm for computing the minimum extent of a family $\mathcal{F}$ of functions. In the algorithm, $A_{\text{opt}}(R)$ is an algorithm that computes the value $x^* = \arg\min_{x \in \Gamma} \mathfrak{E}_R(x)$, where $\mathfrak{E}_R(x)$ is as defined in (1).

There is a slight difference in the two algorithms described in Figures 4 and 5: In the former, only one point is added to the set $R$ in each iteration, while in the latter, two functions are added to $R$ in each iteration. We remark that this difference is not essential.

Note that if the number of iterations is $\kappa$, then the running time of the incremental algorithm is $O(n\kappa)$ plus the time needed to solve $\kappa$ instances of problems of size at most $O(\kappa)$. In the next two subsections, we prove that $\kappa$ can be bounded by a term independent of $n$ in various cases, and perform an experimental study, which shows that $\kappa$ is very small in practice.

APPROX_MINIMUM_EXTENT $(\mathcal{F}, \varepsilon)$

$R \subseteq \mathcal{F}$: arbitrary set of size $\min\{|\mathcal{F}|, c\}$

**while** $(1+\varepsilon)\mathfrak{E}_R((x^* := A_{\text{opt}}(R))) < \mathfrak{E}_{\mathcal{F}}(x^*)$

$f' = \arg\max_{f \in \mathcal{F}} f(x^*)$

$f'' = \arg\min_{f \in \mathcal{F}} f(x^*)$

$R = R \cup \{f', f''\}$

**end while**

**Figure 5.** An incremental algorithm for computing the minimum extent of a family $\mathcal{F}$ of functions.

## 3.2 Analysis

Though our algorithm is not based on $\varepsilon$-kernels directly, its analysis relies on the ideas presented in Section 2 for computing $\varepsilon$-kernels. We first prove that in the case where $\Pi$ is the set of all slabs (i.e., we want to compute the minimum-width slab containing $P$), the incremental algorithm terminates in $O(1/\varepsilon^{(d-1)/2})$

iterations; this bound is tight in the worst case, e.g.,for points unfiromly distributed on $\mathbb{S}^{d-1}$.

**Theorem 3.1** *Let $P$ be a set of $n$ points in $\mathbb{R}^d$, and let $0 < \varepsilon \leq 1/2$ be a parameter. The incremental algorithm computes an $\varepsilon$-approximation of the minimum-width slab containing $P$ in at most $O(1/\varepsilon^{(d-1)/2})$ iterations.*

*Proof:* Clearly, when the algorithm terminates, it returns an $\varepsilon$-approximation of the optimal minimum-width slab. Next we show that the algorithm terminates in $O(1/\varepsilon^{(d-1)/2})$ steps.

Let $\tau$ be the affine transform from Lemma 2.1 so that $\tau(P)$ is an $\alpha$-fat point set in $\mathbb{R}^d$ satisfying $\alpha\mathbb{C} \subset \mathrm{conv}(\tau(P)) \subset \mathbb{C}$, where $\mathbb{C} = [-1, 1]^d$. Note that for any slab $\sigma \subset \mathbb{R}^d$, $\tau(\sigma)$ is also a slab in $\mathbb{R}^d$, and if $p$ is the point farthest away from $\sigma$ in $P$, then $\tau(p)$ is the point farthest away from $\tau(\sigma)$ in $\tau(P)$ .

Let $\mathbf{S} \subseteq \mathbb{R}^d$ be the sphere of radius $\sqrt{d} + 1$ centered at the origin. Let $\delta_1 = \sqrt{\varepsilon\alpha/2} \leq 1/2$ and $\delta_2 = \delta_1/2$. We deploy a set $\mathcal{I}$ of $O(1/\delta_2^{d-1}) = O(1/\varepsilon^{(d-1)/2})$ *sentinel points* on the sphere $\mathbf{S}$ such that for any point $x$ on $\mathbf{S}$, there exists $y \in \mathcal{I}$ such that $\|x - y\| \leq \delta_2$. At the beginning, all sentinel points are unmarked. For any point $q$ on the sphere $\mathbf{S}$, define $\mathcal{N}(q) = \{w \in \mathcal{I} \mid \|w - q\| \leq \delta_2\}$.

At some stage of the algorithm, suppose $R \subseteq P$ is the current subset of points in consideration. The algorithm now computes a slab $\pi^* = A_{\mathrm{opt}}(R, \Pi)$ containing $R$. Suppose that the algorithm does not terminate, i.e., the $\varepsilon$-expansion of $\pi^*$ does not contain $P$. Let $p \in P \setminus \pi^*$ be the point farthest away from $\pi^*$, then $\tau(p) \in \tau(P) \setminus \tau(\pi^*)$ is also the point farthest away from $\tau(\pi^*)$. Let $u$ be one of the two directions normal to $\tau(\pi^*)$ so that $\langle u, \tau(p)\rangle = \max_{q\in P}\langle u, \tau(q)\rangle$, and let $x \in \mathbf{S}$ be the point at which $\mathbf{S}$ is hit by the ray emanating from $\tau(p)$ along $u$ (see Figure 6 (a)). Since $\tau(p)$ is the extreme point in direction $u$, the ball of radius $\|x - \tau(p)\|$ centered at $x$ does not contain any point of $\tau(P)$. Therefore $\tau(p)$ is the nearest neighbor of $x$ in $\tau(P)$. We now mark all the sentinel points in $\mathcal{N}(x)$. Let us remark that marking sentinel points is not part of the algorithm, but just for the analysis.

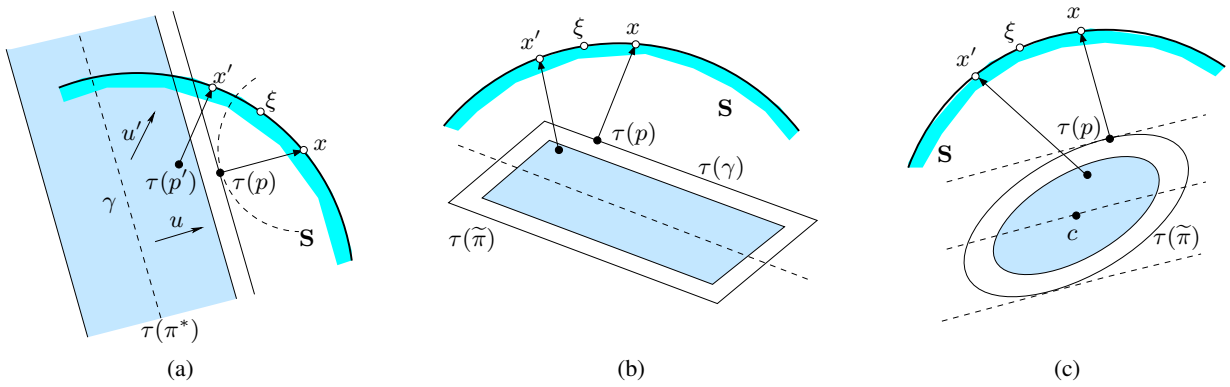



**Figure 6.** Analysis of the algorithm for computing (a) the minimum-width slab, (b) the minimum-volume bounding box, (c) the minimum-radius cylinder (the point $c$ represents the center line of $\tau(\widetilde{\pi})$).

Clearly, $\mathcal{N}(x) \neq \emptyset$ by our choice of $\mathcal{I}$. Moreover, we claim that no points in $\mathcal{N}(x)$ were previously marked. Indeed, otherwise let us pick any marked point $\xi$ from $\mathcal{N}(x)$. Then at some previous step, the algorithm added a point $p' \in P$ to $R$ such that $\tau(p')$ is the farthest point in $\tau(P)$ along some direction $u'$ and $\xi \in \mathcal{N}(x')$, where $x'$ is the point at which the sphere $\mathbf{S}$ is hit by the ray emanating from $\tau(p')$ along $u'$ (see Figure 6 (a)). Thus $\tau(p')$ is the nearest neighbor of $x'$ in $\tau(P)$, and

$$\|x - x'\| \leq \|x - \xi\| + \|x' - \xi\| \leq 2\delta_2 = \delta_1.$$

Now proceeding as in the proof of (2) in Section 2, we obtain

$$\langle u, \tau(p)\rangle - \langle u, \tau(p')\rangle \leq \delta_1^2 = \alpha\varepsilon/2. \tag{4}$$

Let $\gamma$ be the center hyperplane of $\tau(\pi^*)$. Since $\tau(p) \in \tau(P)$ is the point farthest away from $\tau(\pi^*)$, we have

$$d(\tau(p), \gamma) \geq \omega(u, \tau(P))/2 \geq \alpha, \tag{5}$$

where the second inequality follows from the fact that $\alpha\mathbb{C} \subset \text{conv}(\tau(P))$.

It follows from (4) and (5) that $\tau(p')$ lies to the same side of $\gamma$ as $\tau(p)$ (as otherwise we would have $d(\tau(p), \gamma) \leq \langle u, \tau(p)\rangle - \langle u, \tau(p')\rangle \leq \alpha\varepsilon/2 < \alpha$), and

$$d(\tau(p'), \gamma) = d(\tau(p), \gamma) - \big(\langle u, \tau(p)\rangle - \langle u, \tau(p')\rangle\big) \geq d(\tau(p), \gamma) - \alpha\varepsilon/2 \geq (1 - \varepsilon/2)d(\tau(p), \gamma).$$

Therefore, $(1 + \varepsilon)d(\tau(p'), \gamma) \geq d(\tau(p), \gamma)$, i.e., the $\varepsilon$-expansion of $\tau(\pi^*)$ contains $\tau(p)$ and hence $\tau(P)$, which implies the $\varepsilon$-expansion of $\pi^*$ has already contained $P$, a contradiction.

We can then conclude that at least one previously unmarked sentinel point is marked in each iteration of the algorithm, which implies that the algorithm terminates within $|\mathcal{I}| = O(1/\varepsilon^{(d-1)/2})$ iterations. □

Using the same technique, one can prove a bound $O(1/\varepsilon^{(d-1)/2})$ on the number of iterations of the algorithm for other centrally symmetric convex shapes, such as boxes and cylinders.

**Theorem 3.2** *Let $P$ be a set of $n$ points in $\mathbb{R}^d$, and let $0 < \varepsilon \leq 1/2$ be a parameter. The incremental algorithm computes an $\varepsilon$-approximation of the minimum-volume box (resp. the minimum-radius cylinder) containing $P$ in at most $O(1/\varepsilon^{(d-1)/2})$ iterations.*

*Proof:* When the algorithm for computing the minimum-volume bouding box terminates, it returns a bounding box of $P$ whose volume is at most $(1 + \varepsilon)^d < 1 + O(\varepsilon)$ times the optimal. To show that the algorithm terminates in $O(1/\varepsilon^{(d-1)/2})$ iterations, we proceed as in the proof of Theorem 3.1. Let $\tau$, $\mathbf{S}$, $\mathcal{I}$ and $\mathcal{N}(\cdot)$ be defined as before. At some stage of the algorithm, suppose $p$ is the point to be added to $R$ in the current iteration, i.e., $p$ is the point that requires $A_{\text{opt}}(R, \Pi)$ to be scaled the most in order to be covered. Let $\widetilde{\pi}$ be the box least scaled from $A_{\text{opt}}(R, \Pi)$ so that it covers $p$. Then $p$ lies on some facet of $\widetilde{\pi}$, denoted by $\gamma$. Hence $\tau(p)$ also lies on the facet $\tau(\gamma)$ of the zonotope $\tau(\widetilde{\pi})$; see Figure 6 (b).

Let $x \in \mathbf{S}$ be the point at which the ray emanating from $\tau(p)$ and orthogonal to $\tau(\gamma)$ hits the sphere $\mathbf{S}$. We mark all the sentinel points in $\mathcal{N}(x) \subseteq \mathcal{I}$. Note that $\mathcal{N}(x)$ contains at least one previously unmarked sentinel point, as otherwise, by the same argumentation as in the proof of Theorem 3.1, an $\varepsilon$-expansion of $A_{\text{opt}}(R, \Pi)$ would have already contained $P$. The theorem then follows because $|\mathcal{I}| = O(1/\varepsilon^{(d-1)/2})$.

In the case of cylinders, we define $x \in \mathbf{S}$ to be the point at which $\mathbf{S}$ is hit by the ray emanating from $\tau(p)$ and orthogonal to the tangent hyperplane of $\tau(\widetilde{\pi})$ at $\tau(p)$; see Figure 6 (c). The rest of the proof is similar to that given for the case of boxes. □

**Remark.** For all these shapes, the key observation is that when the test in the while loop of the incremental algorithm fails, there is a witness slab that contains $R$ but its $\delta$-expansion does not contain $P$, for $\delta = \Theta(\varepsilon)$.

We now turn to the incremental algorithm described in Figure 5 for approximating the minimum extent of a family $\mathcal{F} = \{f_1, \cdots, f_n\}$ of $k$-variate functions over a domain $\Gamma \subseteq \mathbb{R}^k$. We first address the case of linear functions. Consider the "duality" transform that maps the linear function $f(x) = a_0 + a_1x_1 + \cdots a_kx_k$ to the point $\hat{f} = (a_0, a_1, \ldots, a_k)$, and a point $x = (x_1, \ldots, x_k)$ to the unit vector $\hat{x} = \frac{1}{||(1,x_1,\ldots,x_k)||}(1, x_1, \ldots, x_k)$. A simple calculation shows that when the test in the while loop of the incremental algorithm fails, an $\varepsilon$-expansion of the thinnest slab normal to the unit vector $\widehat{x^*}$ containing $\widehat{R} = \{\hat{f} \mid f \in R\}$ fails to contain the point set $\hat{\mathcal{F}} = \{\hat{f} \mid f \in \mathcal{F}\}$. Thus the proof of Theorem 3.1, applied to $\hat{\mathcal{F}}$, shows that the incremental algorithm in Figure 5 terminates in $O(1/\varepsilon^{k/2})$ steps.

**Theorem 3.3** *Let $\mathcal{F} = \{f_1, \cdots, f_n\}$ be a family of $m$-variate polynomial functions (over the domain $x \in \mathbb{R}^m$) that admits a linearization of dimension $k$, i.e., there exist $k$ $m$-variate polynomials $\phi_1, \cdots, \phi_k$, such that each $f_i \in \mathcal{F}$ can be written in the form of*

$$f(x) = a_{i0} + a_{i1}\phi_1(x) + \cdots + a_{ik}\phi_k(x)$$

*for some parameters $a_{i0}, \cdots, a_{ik}$. Then the incremental algorithm computes an $\varepsilon$-approximation of the minimum extent of $\mathcal{F}$ over any domain $\Gamma \subset \mathbb{R}^m$ in $O(1/\varepsilon^{\min\{k/2,m\}})$ iterations.*

*Proof:* The bound $O(1/\varepsilon^{k/2})$ follows directly from our above discussion and the fact that, by linearization, computing the minimum extent of $\mathcal{F}$ is equivalent to computing the minimum extent of a set of $k$-variate linear functions

$$\left\{ g_i(y_1, \cdots, y_k) = a_{i0} + a_{i1}y_1 + \cdots + a_{ik}y_k \mid 1 \le i \le n \right\}$$

over the domain $(y_1, \cdots, y_k) \in \Gamma = \{(\phi_1(x), \cdots, \phi_k(x)) \mid x \in \mathbb{R}^m\} \subseteq \mathbb{R}^k$.

We next show that the number of iterations can also be bounded by $O(1/\varepsilon^m)$. We need the following result, whose proof can be found in [5]: There exists a decomposition $\mathcal{D}$ of $\mathbb{R}^m$ into $O(1/\varepsilon^m)$ cells, so that for each cell $\Delta \in \mathcal{D}$ and any $x_0 \in \Delta$, the two functions $f'_\Delta, f''_\Delta \in \mathcal{F}$ $\varepsilon$-approximate $\mathcal{F}$ within $\Delta$, i.e.,

$$(1+\varepsilon)\left(f'_\Delta(x) - f''_\Delta(x)\right) \ge \mathfrak{E}_{\mathcal{F}}(x)$$

for all $x \in \Delta$, where $f'_\Delta = \arg\max_{f \in \mathcal{F}} f(x_0)$ and $f''_\Delta = \arg\min_{f \in F} f(x_0)$. Now, recall that a value $x^*$ is computed by $A_{\text{opt}}$ at each step of the incremental algorithm depicted in Figure 5. We say a cell $\Delta \in \mathcal{D}$ is visited by $x^*$ if $x^* \in \Delta$. We claim that each cell in $\mathcal{D}$ (except for at most one cell) can be visited at most once during the execution of the algorithm. This is because, when a cell $\Delta$ is visited for the first time, two functions $f'_\Delta, f''_\Delta$ that $\varepsilon$-approximate $\mathcal{F}$ within $\Delta$ will be added to the set $R$, and hence the algorithm would stop if $\Delta$ is visited for the second time. It follows that the number of iterations of the algorithm can be bounded by the number of cells in $\mathcal{D}$, i.e., $O(1/\varepsilon^m)$. □

Some shape fitting problems cannot be directly formulated as computing the minimum extent of a family of linearizable functions, such as the smallest enclosing spherical or cylindrical shell problem. For these problems, we can use the next theorem to bound the number of iterations. We need the following lemma, which is a refined version of a similar observation in [5].

**Lemma 3.4** *Let $0 < \varepsilon < 1$, and $0 \le a \le A \le B \le b$, and $r \ge 1$ be an integer. If $B^r - A^r \ge (1-\varepsilon^r)(b^r - a^r)$, then $B - A \ge (1-\varepsilon)(b-a)$.*

*Proof:* Let $B^r - A^r = c$ be fixed and let $B$ vary. The function

$$B - A = B - (B^r - c)^{1/r}$$

is monotonically decreasing function with $B$ for $B > 0$, as its derivate with respect to $B$, $1 - 1/(1 - c/B^r)^{1-1/r}$, is negative for $B > 0$. Thus, in order to prove $B - A \ge (1-\varepsilon)(b-a)$, we only need to prove it for the case of $B = b$.

Letting $B = b$ in the inequality $B^r - A^r \geq (1-\varepsilon^r)(b^r - a^r)$, we then have

$$\begin{aligned} A^r &\leq (1-\varepsilon^r)a^r + \varepsilon^r b^r \\ &= ((1-\varepsilon)+\varepsilon)^r a^r + \varepsilon^r b^r - \varepsilon^r a^r \\ &= \sum_{i=0}^{r-1} \binom{r}{i} (1-\varepsilon)^{r-i}\varepsilon^i a^r + \varepsilon^r b^r \\ &\leq \sum_{i=0}^{r-1} \binom{r}{i} (1-\varepsilon)^{r-i}\varepsilon^i a^{r-i} b^i + \varepsilon^r b^r \\ &= ((1-\varepsilon)a + \varepsilon b)^r . \end{aligned}$$

Therefore, $A \leq (1-\varepsilon)a + \varepsilon b$, or equivalently, $B - A \geq (1-\varepsilon)(b-a)$. □

**Theorem 3.5** *Let $\mathcal{F} = \{f_1, \cdots, f_n\}$ be a family of $m$-variate nonnegative functions (over the domain $\mathbb{R}^m$). Suppose there exist two other $m$-variate functions $\varphi(x), \psi(x)$ and an integer $r \geq 1$, so that $\psi(x)$ is positive, and each $g_i(x) = \varphi(x) + \psi(x) f_i^r(x)$ is a polynomial. If $\mathcal{G} = \{g_1, \cdots, g_n\}$ admits a linearization of dimension $k$, then the incremental algorithm computes an $\varepsilon$-approximation of the minimum extent of $\mathcal{F}$, over any domain $\Gamma \subset \mathbb{R}^m$, in at most $O(1/\varepsilon^{r\cdot\min\{k/2,m\}})$ iterations.*

*Proof:* Let $\mathcal{R} \subseteq \mathcal{F}$, and let $\mathcal{K} = \{g_i \mid f_i \in \mathcal{R}\}$. We observe that

$$\arg\min_{x\in\mathbb{R}^m} \mathfrak{E}_{\mathcal{R}}(x) = \arg\min_{x\in\mathbb{R}^m} \mathfrak{E}_{\mathcal{K}}(x); \tag{6}$$

$$\arg\max_i f_i(x) = \arg\max_i g_i(x), \quad \arg\min_i f_i(x) = \arg\min_i g_i(x) \qquad \forall x \in \mathbb{R}^m; \tag{7}$$

$$(1 + (\varepsilon/2)^r)\mathfrak{E}_{\mathcal{K}}(x) \geq \mathfrak{E}_{\mathcal{G}}(x) \Rightarrow (1+\varepsilon)\mathfrak{E}_{\mathcal{R}}(x) \geq \mathfrak{E}_{\mathcal{F}}(x). \tag{8}$$

The first two claims are straightforward. To prove the third one, we observe that if

$$(1 + (\varepsilon/2)^r)(g_a(x) - g_b(x)) \geq \max_i g_i(x) - \min_i g_i(x),$$

then it follows that

$$g_a(x) - g_b(x) \geq (1 - (\varepsilon/2)^r)(\max_i g_i(x) - \min_i g_i(x)),$$

or equivalently

$$f_a^r(x) - f_b^r(x) \geq (1 - (\varepsilon/2)^r)(\max_i f_i^r(x) - \min_i f_i^r(x)).$$

By Lemma 3.4, we then have

$$f_a(x) - f_b(x) \geq (1 - \varepsilon/2)(\max_i f_i(x) - \min_i f_i(x)),$$

which implies

$$(1+\varepsilon)(f_a(x) - f_b(x)) \geq \max_i f_i(x) - \min_i f_i(x).$$

Three observations (6)–(8) imply that the incremental algorithm for computing an $\varepsilon$-approximation of the minimum extent of $\mathcal{F}$ naturally translates into an incremental algorithm for computing an $(\varepsilon/2)^r$-approximation of the minimum extent of $\mathcal{G}$. Moreover, the former has no more iterations than the latter. By Theorem 3.3, we obtain the desired result. □

As shown in [5], the problem of computing the smallest spherical shell containing a set $P$ of $n$ points in $\mathbb{R}^d$ can beformulated as computing $\arg\min_{x\in\mathbb{R}^d} \mathfrak{E}_{\mathcal{F}}(x)$, where $\mathcal{F} = \{\|p - x\| \mid p \in P\}$. Since $\mathcal{F}$ satisfies

the conditions in Theorem 3.5 with $m = d$, $r = 2$, and $k = d$,the incremental algorithm for computing the spherical shell terminates in $O(1/\varepsilon^d)$ iterations. Similarly, using the result in [5], one can argue that the incremental algorithm for minimum-width cylindrical shell terminates in $O(1/\varepsilon^{4(d-1)})$ iterations, as it satisfies the lemma with $m = 2d - 2$, $r = 2$, and $k = O(d^2)$. We thus obtain the following.

**Corollary 3.6** *Let $P$ be a set of $n$ points in $\mathbb{R}^d$, and let $0 < \varepsilon \leq 1/2$ be a parameter. The incremental algorithm computes an $\varepsilon$-approximation of the smallest spherical shell containing $P$ in $O(1/\varepsilon^d)$ iterations, or the smallest cylindrical shell containing $P$ in $O(1/\varepsilon^{4(d-1)})$ iterations.*

**Remark.** A similar incremental algorithm was proposed by Bădoiu *et al.* [14] for computing the smallest enclosing ball of a point set and by Kumar and Yildirim [26] for computing the smallest enclosing ellipsoid. Note that both these problems can be formulated as convex programs, unlike the problems we address here. Thus our proof of convergence is quite different from the proofs in these papers.

### 3.3 Experimental results

In this section we study the empirical performance of our incremental approach for shape fitting by applying it to (i) smallest enclosing cylinder in $\mathbb{R}^3$, (ii) minimum-width annulus in $\mathbb{R}^2$, and (iii) minimum-volume bounding box in $\mathbb{R}^3$.

**Smallest enclosing cylinder.** In the incremental algorithm described in Figure 3, we need to implement the $A_{\text{opt}}$ algorithm for computing the smallest enclosing cylinder. The exact algorithms for computing the smallest enclosing cylinder are not only expensive (the best known exact algorithm requires near-cubic time [2]), but also involve complicated machineries such as parametric search and computing the roots of high-degree polynomials. Therefore we implemented the approximation algorithm by Agarwal *et al.* [2], which computes an $\varepsilon$-approximation of the smallest enclosing cylinder in $O(n/\varepsilon^2)$ time. (Note that the analysis of the incremental algorithm extends readily to the case where $A_{\text{opt}}$ is replaced by an approximation algorithm.) This algorithm requires a procedure for computing smallest enclosing disks, and we have used Gärtner's code [19] for this purpose.

We performed experiments on both synthetic inputs and real geometric models. The synthetic input is generated by uniformly sampling on cylindrical surfaces of fixed radius $r = 1.0$ and various heights $h$. We compare the performance of three algorithms:

(i) the approximation algorithm by Agarwal *et al.* [2]; we refer to this algorithm as APPR;

(ii) compute $\varepsilon$-kernels as described in Section 2 and then apply APPR to kernels; we refer to this algorithm as CORE;

(iii) incremental algorithm with APPR on the small problems for each iteration; we refer to this algorithm as INCR;

We compare the performance of these algorithms in Table 4. We also tried a fourth approach, in which we first computed a $\varepsilon$-kernels and then applied INCR to kernels. We observed that this algorithm improves very little over CORE and performs worse than INCR; the reason is that computing $\varepsilon$-kernels was the most time-consuming step in the second and fourth algorithms. We therefore did not include the performance of this algorithm in the table.

We paid special attention to the number of iterations required by INCR and recorded it in Table 4 (numbers in parenthesis).

As expected, CORE is much faster than APPR, and the output quality of the two algorithms is roughly the same. More interestingly, we observed that INCR always outperforms CORE. Intuitively, the reason

| *Input Type* | *Input Size* | *Running Time* | | | *Output Radius* | | |
|---|---|---|---|---|---|---|---|
| | | APPR | CORE | INCR | APPR | CORE | INCR |
| $h = 2.0$ | 10,000 | 20.4 | 0.3 | 0.1 (6) | 1.024 | 1.012 | 1.009 |
| | 100,000 | 226.8 | 1.1 | 0.3 (6) | 1.021 | 1.020 | 1.013 |
| | 1,000,000 | 2640.1 | 12.2 | 1.6 (5) | 1.010 | 1.011 | 1.013 |
| $h = 20.0$ | 10,000 | 16.8 | 0.3 | 0.1 (7) | 1.029 | 1.078 | 1.072 |
| | 100,000 | 187.2 | 1.1 | 0.3 (7) | 1.086 | 1.056 | 1.021 |
| | 1,000,000 | 2026.5 | 12.5 | 2.4 (8) | 1.066 | 1.039 | 1.068 |
| $h = 200.0$ | 10,000 | 16.5 | 0.3 | 0.1 (7) | 1.052 | 1.094 | 1.050 |
| | 100,000 | 186.0 | 1.0 | 0.2 (4) | 1.030 | 1.092 | 1.018 |
| | 1,000,000 | 2067.2 | 12.0 | 2.9 (10) | 1.072 | 1.039 | 1.037 |
| *bunny* | 35,947 | 68.1 | 0.5 | 0.2 (6) | 0.067 | 0.067 | 0.067 |
| *dragon* | 437,645 | 809.8 | 2.9 | 1.0 (7) | 0.077 | 0.077 | 0.077 |
| *buddha* | 543,652 | 1126.3 | 3.6 | 1.2 (7) | 0.041 | 0.041 | 0.041 |

**Table 4.** A comparison of the performance of various approximation algorithms for computing the smallest enclosing cylinder in $\mathbb{R}^3$. The numbers of iterations performed by the incremental algorithm are in parenthesis. Running time is measured in seconds.

may be explained as follows: in practice, the size of an $\varepsilon$-kernel can be much smaller than its theoretical bound, and INCR can find such a kernel on the fly, instead of using an algorithm that provides worst-case gurantee on the quality of the output. In some ways, the incremental algorithm can be viewed as an output-sensitive algorithm for computing a kernel. As shown in Table 4, the number of iterations of the incremental algorithm never exceeded 10 in all our experiments.

**Minimum-width annulus.** We implemented a simple brute-force $O(n^5)$ algorithm for computing $A_{\text{opt}}$ exactly. We ran our incremental algorithm on point sets uniformly sampled from annuli of fixed inner radius $r = 1.00$ and various widths $w$. The experimental results are shown in Table 5. Note that when the point set is almost circular, which is probably the interesting case in practice, the algorithm has very few iterations. This is what we would expect for the incremental algorithm.

| *Input Type* | *Input Size* | *Running Time* | *Output Width* |
|---|---|---|---|
| $w = 0.05$ | $10^4$ | 0.01 (2) | 0.0501 |
| | $10^5$ | 0.02 (2) | 0.0500 |
| | $10^6$ | 0.18 (2) | 0.0500 |
| $w = 0.50$ | $10^4$ | 0.01 (2) | 0.5014 |
| | $10^5$ | 0.03 (2) | 0.5004 |
| | $10^6$ | 0.26 (2) | 0.5001 |
| $w = 50.0$ | $10^4$ | 0.07 (9) | 50.051 |
| | $10^5$ | 0.12 (9) | 50.018 |
| | $10^6$ | 0.67 (9) | 50.001 |
| $w = 500$ | $10^4$ | 0.32 (13) | 503.64 |
| | $10^5$ | 0.88 (16) | 504.04 |
| | $10^6$ | 2.47 (18) | 502.57 |

**Table 5.** Performance of the incremental algorithm for computing the minimum-width annulus in $\mathbb{R}^2$. The numbers of iterations performed by the algorithm are in parenthesis. Running time is measured in seconds.

**Minimum-volume box.** We used an approximation algorithm (BHBX) by Barequet and Har-Peled [11] for computing $A_{\text{opt}}$, whose implementation was obtained from [21]. We compared our incremental algorithm (INCR) with BHBX on various inputs. The input *box*$(a, b, c)$ is generated by uniformly sampling from the boundary of a 3D box of side lengths $a$, $b$ and $c$. The input *ellipsoid*$(a, b, c)$ is generated by uniformly sampling from the boundary of a 3D ellipsoid of axes lengths $a$, $b$ and $c$. The results are summarized in

Table 6. Note that on inputs that are closer to being spherical, INCR is somewhat slow, whereas it clearly becomes much faster on inputs that are far from spherical.

| *Input Type* | *Input Size* | *Running Time* | | *Output Volume* | |
|---|---|---|---|---|---|
| | | INCR | BHBX | INCR | BHBX |
| sphere | $10^5$ | 45.3 (180) | 53.6 | 7.999 | 7.999 |
| ($r = 1.0$) | $10^6$ | 49.4 (182) | 5180.8 | 8.000 | 8.000 |
| ellipsoid | $10^5$ | 5.4 (53) | 2.4 | 16.20 | 16.18 |
| $(2, 2, 4)$ | $10^6$ | 11.4 (75) | 63.3 | 16.20 | 16.27 |
| ellipsoid | $10^5$ | 0.8 (15) | 0.9 | 327.2 | 328.0 |
| $(2, 4, 40)$ | $10^6$ | 1.4 (17) | 2.6 | 324.8 | 324.7 |
| box | $10^5$ | 0.1 (2) | 0.6 | 16.00 | 16.21 |
| $(2, 2, 4)$ | $10^6$ | 0.1 (2) | 1.8 | 16.04 | 16.07 |
| box | $10^5$ | 0.1 (2) | 0.5 | 320.0 | 320.5 |
| $(2, 4, 40)$ | $10^6$ | 0.1 (2) | 1.3 | 320.0 | 321.4 |
| bunny | 36K | 0.3 (9) | 0.5 | 1.060 | 1.076 |
| dragon | 44K | 0.7 (12) | 0.8 | 2.450 | 2.486 |
| buddha | 54K | 1.4 (20) | 0.9 | 1.311 | 1.316 |

**Table 6.** Performance of the incremental algorithm for computing the minimum-volume box in $\mathbb{R}^3$. The numbers of iterations performed by the algorithm are in parenthesis. Running time is measured in seconds.

# 4 Kinetic Data Structures

In this section, we apply the $\varepsilon$-kernel algorithm to maintain an approximation to the minimum orthogonal bounding box and the convex hull of moving points in $\mathbb{R}^2$, and show empirically how this technique impacts the efficiency of kinetic data structures.

**Maintaining minimum orthogonal bounding box.** We maintain the minimum orthogonal bounding box of a point set by keeping track of the two extremal points along each dimension using kinetic tournament trees [12]. Let $p_x(t)$ (resp. $p_y(t)$) denote the trajectory — a curve in $\mathbb{R}^2$ — of the $x$-coordinate (resp. $y$-coordinate) of the point $p \in P$. Let $\mathcal{F}_x = \{p_x \mid p \in P\}$ and $\mathcal{F}_y = \{p_y \mid p \in P\}$. We compute $\varepsilon$-kernels $\mathcal{G}_x$ and $\mathcal{G}_y$ of $\mathcal{F}_x$ and $\mathcal{F}_y$ respectively, using linearization and duality as described in [5] and the algorithm described in Section 2. We set

$$Q = \{p \mid p_x \in \mathcal{G}_x \text{ or } p_y \in \mathcal{G}_y\},$$

and maintain a bounding box of $Q$ using kinetic data structures.

Our experiments were conducted on two types of synthetic motions: (i) linear motion, and (ii) quadratic motion. For a linear motion $x(t) = v \cdot t + x_0$ along each dimension, $x_0$ is uniformly sampled from $[-1, 1]$, and $v$ is uniformly sampled from $[-4, 4]$, with small perturbations. For a quadratic motion $x(t) = a \cdot t^2 + v \cdot t + x_0$ along each dimension, $(v, x_0) \in \mathbb{R}^2$ is uniformly sampled from $\mathbb{S}^1$, and $a$ follows a normal distribution with mean $0$ and variance $0.005$. These synthetic motions tend to have a large number of external events (i.e., topological changes in the structure maintained) compared to the input size. We also appropriately scaled the parameters so that most of the external events happened during the time interval $[-10, 10]$.

We present our experimental results for the 2D case only; similar results were also observed in higher dimensions. We studied two measures in our experiments: (i) relative error in the size of the bounding box as a function of time, and (ii) number of kinetic events.

The quality of a kernel, which measures the relative error in the size of the bounding box, at a time $t$ is measured by

$$\min\left\{\frac{\mathfrak{E}_{\mathcal{G}_x}(t)}{\mathfrak{E}_{\mathcal{F}_x}(t)}, \frac{\mathfrak{E}_{\mathcal{G}_y}(t)}{\mathfrak{E}_{\mathcal{F}_y}(t)}\right\},$$

where $\mathfrak{E}$ is as defined in (1) of Section 1. We plotted the quality of the computed kernel over time in Figure 7. It was observed that a kernel of size 40 (resp. 80) is sufficient to guarantee the quality above 0.96 over time for a set of 100,000 moving points with linear (resp. quadratic) trajectories. Figure 8 compares how kinetic events were distributed over time for the input and its kernel. Each plotted point in the figure denotes the number of events within $\pm 0.5$ seconds of that time instant. As can be seen, by choosing a kernel of small size, the number of kinetic events was significantly reduced, while the maintained box is a good approximation of the original over time.

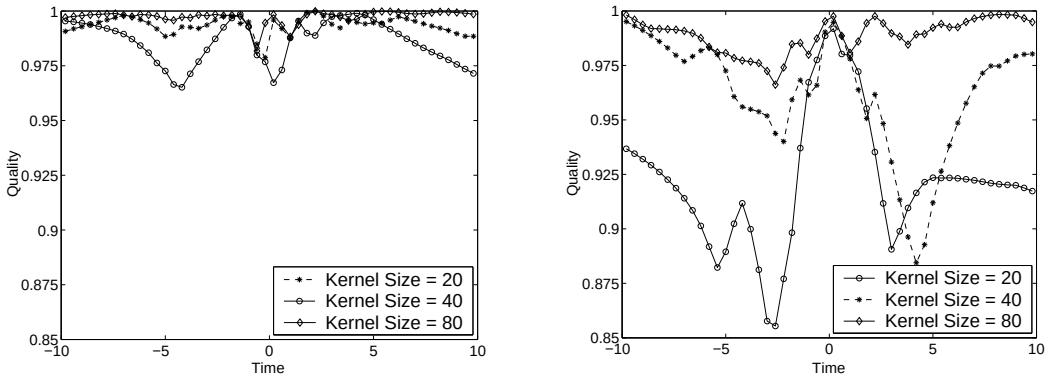



**Figure 7.** Quality of kernels for the minimum orthogonal bounding box of 100,000 moving points over the time interval $[-10, 10]$. (a) Linear motion, (b) quadratic motion.

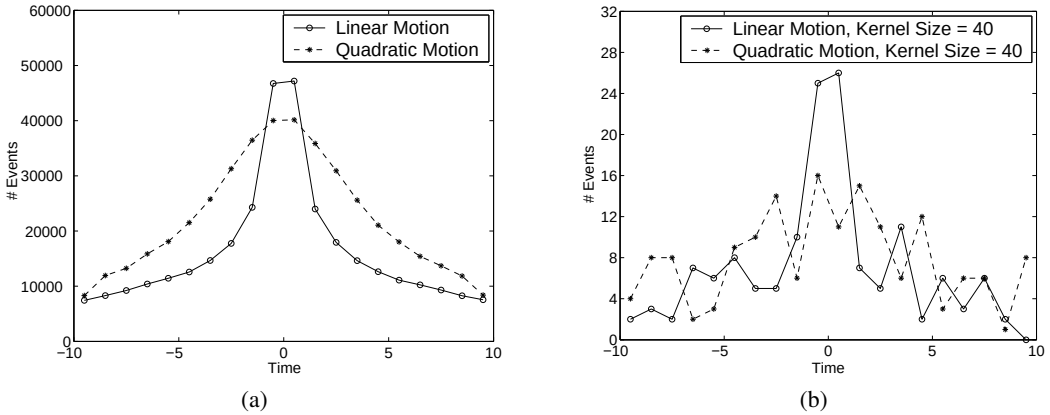



**Figure 8.** Distribution of kinetic events for maintaining the minimum orthogonal bounding box of 100,000 moving points over the time interval $[-10, 10]$. Each plotted point denotes the number of events within $\pm 0.5$ seconds of that time instant. (a) Exact algorithm, (b) using $\varepsilon$-kernels of size 40.

**Maintaining convex hull.** The algorithm of Basch *et al.* [12] for maintaing the convex hull of a point set involves a set of sophisticated certificates and maintenance operations. Instead of using their algorithm, we maintain the convex hull by maintaining a simple triangulation of the point set over time, which is much simpler and relatively efficient in practice. We compute an $\varepsilon$-kernel $Q$ of the moving point set $P$ as described in [5], using the algorthm of Section 2 as a subroutine.

We tested the algorithm for linear motion; quadratic motion would entail solving high-degree polynomial equations. The initial position of a point is randomly chosen on the unit circle, and the velocity is a random unit vector. We also tried various other synthetic motions, e.g., the speeds or the directions of the velocities are clustered. They yielded similar experimental results as reported below.

We measured the quality of a kernel at time $t$ using (3), where $\Delta$ is a set of 200 random directions in $\mathbb{R}^2$. We also computed the ratios between the width and the diameter of the kernel and of the input over time. We plotted our results in Figure 9. It was observed that a kernel of size at most 50 is sufficient to keep the quality above $0.9$ over time for an input of size $10,000$, and the approximation becomes better as the size of the kernel increases. The approximation of the width or the diameter is even better. In Figure 10, we also counted the number of kinetic events for maintaining conv$(P(t))$ and conv$(Q(t))$. Each plotted point in the figure denotes the number of events within $\pm 0.2$ seconds of that time instant. Since we were not using the best known KDS for maintaining the convex hull and the focus was on demonstrating that the KDS for conv$(Q)$ processes much fewer events than that for conv$(P)$, we only counted the number of *external* kinetic events, i.e., the events at which the combinatorial structure of the convex hull changes. A similar ratio will hold if we used the kinetic data structure by Basch *et al.* [12] and counted the number of incremental events as well. We plotted the distribution of external kinetic events in Figure 10 for both the input and its kernel. As can be seen, we were able to process significantly fewer kinetic events while maintaining a good approximation of the convex hull as the points move.

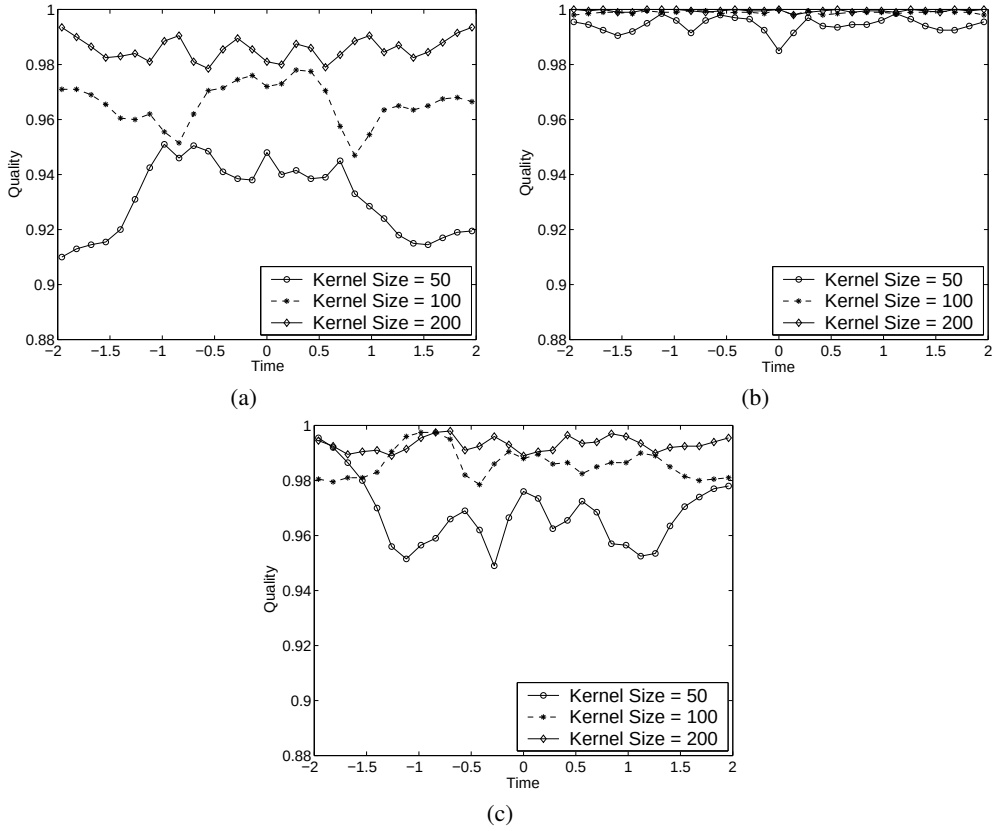
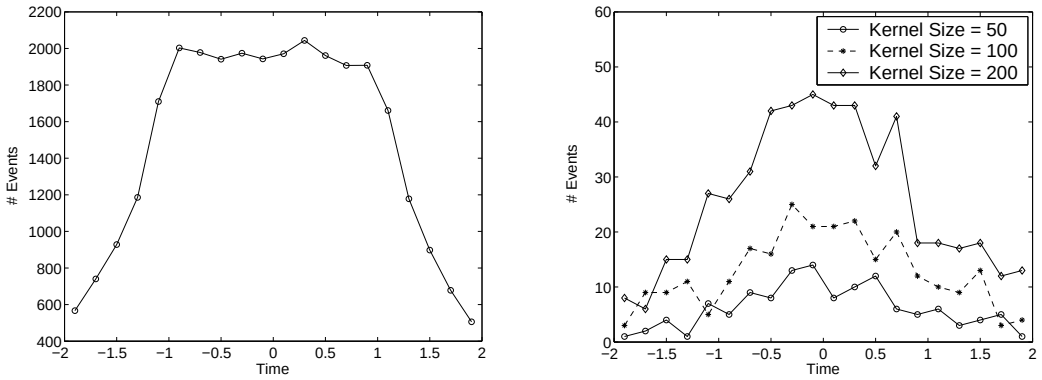



**Figure 9.** Quality of kernels for the convex hull of 10,000 moving points over the time interval $[-2, 2]$. (a) Quality over 200 random directions, (b) quality of width, (c) quality of diameter.




(a) (b)

**Figure 10.** Distribution of external kinetic events for maintaining the convex hull of 10,000 moving points over the time interval $[-2, 2]$. Each plotted point denotes the number of events within $\pm 0.2$ seconds of that time instant. (a) Exact convex hull, (b) using $\varepsilon$-kernels.

# 5 Conclusions

In this paper we presented a simple and practical algorithm for computing the $\varepsilon$-kernel of a set of points in $\mathbb{R}^d$ and studied its empirical performance on various inputs. We then described a generic incremental algorithm for fitting various shapes on a point set. Although we can only prove a weaker bound on the worst-case running time of this algorithm, our empirical study shows that it performs very well in practice. We also applied our $\varepsilon$-kernel algorithm to maintain various extent measures of a set of moving points, which significantly improved the performance of kinetic data structures.

An interesting open problem is whether our incremental algorithm for the smallest enclosing cylinder problem (i.e., fitting a 1-dimensional flat — a line — through the point set) terminates in $O(f(d) \cdot g(d, \varepsilon))$ iterations, where $g(d, \varepsilon)$ is a function that depends polynomially on $d$. Note that the algorithm for the smallest enclosing ball problem (i.e., fitting a 0-dimensional flat — a point — through the point set) terminates in $O(1/\varepsilon)$ iterations, while the algorithm for the minimum-width slab problem (i.e., fitting a $(d-1)$-dimensional flat — a hyperplane — through the point set) requires $\Omega(1/\varepsilon^{(d-1)/2})$ iterations in the worst case.

**Acknowledgments.** The authors are grateful to Sariel Har-Peled for helpful comments.

# References

[1] P. K. Agarwal, B. Aronov, S. Har-Peled, and M. Sharir, Approximation and exact algorithms for minimum-width annuli and shells, *Discrete Comput. Geom.*, 24 (2000), 687–705.

[2] P. K. Agarwal, B. Aronov, and M. Sharir, Line transversals of balls and smallest enclosing cylinders in three dimensions, *Discrete Comput. Geom.*, 21 (1999), 373–388.

[3] P. K. Agarwal, B. Aronov, and M. Sharir, Exact and approximation algorithms for minimum-width cylindrical shells, *Discrete Comput. Geom.*, 26 (2001), 307–320.

[4] P. K. Agarwal, L. J. Guibas, J. Hershberger, and E. Veach, Maintaining the extent of a moving point set, *Discrete Comput. Geom.*, 26 (2001), 353–374.

[5] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, Approximating extent measures of points, *J. ACM*, 51 (2004), 606-635.

[6] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan, Geometric approximation via coresets, in *Combinatorial and Computational Geometry* (J. E. Goodman, J. Pach, and E. Welzl, eds.), Cambridge University Press, Cambridge, in press.

[7] P. K. Agarwal and J. Matoušek, On range searching with semialgebraic sets, *Discrete Comput. Geom.*, 11 (1994), 393–418.

[8] P. K. Agarwal, C. M. Procopiuc, and K. R. Varadarajan, Approximation algorithms for $k$-line center, *Proc. 10th Annu. European Sympos. Algorithms*, 2002, pp. 54–63.

[9] S. Arya, T. Malamatos, and D. Mount, Space-efficient approximate Voronoi diagrams, *Proc. 34th Annu. ACM Sympos. Theory Comput.*, 2002, pp. 721–730.

[10] S. Arya, D. Mount, N. Netanyahu, R. Silverman, and A. Y. Wu, An optimal algorithm for approximate nearest neighbor searching in fixed dimensions, *Proc. 5th ACM-SIAM Sympos. Discrete Algorithms*, 1994, pp. 573–582.

[11] G. Barequet and S. Har-Peled, Efficiently approximating the minimum-volume bounding box of a point set in three dimensions, *J. Algorithms*, 38 (2001), 91–109.

[12] J. Basch, L. J. Guibas, and J. Hershberger, Data structures for mobile data, *J. Algorithms*, 31 (1999), 1–28.

[13] M. Bădoiu and K. Clarkson, Smaller core-sets for balls, *Proc. 14th Annu. ACM-SIAM Sympos. Discrete Algorithms*, 2003, pp. 801–802.

[14] M. Bădoiu, S. Har-Peled, and P. Indyk, Approximate clustering via core-sets, *Proc. 34th Annu. ACM Sympos. Theory Comput.*, 2002, pp. 250–257.

[15] T. M. Chan, Approximating the diameter, width, smallest enclosing cylinder and minimum-width annulus, *Internat. J. Comput. Geom. Appl.*, 12 (2002), 67–85.

[16] T. M. Chan, Faster core-set constructions and data stream algorithms in fixed dimensions, *Proc. 20th Annu. ACM Sympos. Comput. Geom.*, 2004, pp. 152–159.

[17] K. L. Clarkson, Las Vegas algorithms for linear and integer programming when the dimension is small, *J. ACM*, 42 (1995), 488–499.

[18] B. Gärtner, A subexponential algorithm for abstract optimization problems, *SIAM J. Comput.*, 24 (1995), 1018–1035.

[19] B. Gärtner, Smallest enclosing ball — fast and robust in C++, `http://www.inf.ethz.ch/personal/gaertner/miniball.html`

[20] S. Har-Peled, A replacement for Voronoi diagrams of near linear size, *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci.*, 2001, pp. 94–103.

[21] S. Har-Peled, A practical approach for computing the diameter of a point set (software), `http://valis.cs.uiuc.edu/~sariel/research/papers/00/diameter/diam_prog.html`

[22] S. Har-Peled and A. Kushal, Smaller coresets for $k$-median and $k$-means clustering, *Proc. 21st Annu. Sympos. Comput. Geom.*, 2005, pp. 126–134.

[23] S. Har-Peled and S. Mazumdar, Coresets for $k$-means and $k$-median clustering and their applications, *Proc. 36th Annu. ACM Sympos. Theory Comput.*, 2004, pp. 291–300.

[24] S. Har-Peled and Y. Wang, Shape fitting with outliers, *SIAM J. Comput.* 33 (2004), 269–285.

[25] P. Kumar, J. Mitchell, and E. Yildirim, Computing core-sets and approximate smallest enclosing hyperspheres in high dimensions, *Proc. 5th Workshop Algorithm Eng. Exper.*, 2003, pp. 45–55.

[26] P. Kumar and E. Yildirim, Approximating minimum volume enclosing ellipsoids using core sets, to appear in *J. Optimization Theory and Approximations*.

[27] Large geometric models archive, `http://www.cc.gatech.edu/projects/large_models/`

[28] D. Mount and S. Arya, ANN: library for approximate nearest neighbor searching, `http://www.cs.umd.edu/~mount/ANN/`

[29] A. C. Yao, On constructing minimum spanning trees in $k$-dimensional spaces and related problems, *SIAM J. Comput.*, 11 (1982), 721–736.

[30] Y. Zhou and S. Suri, Algorithms for a minimum volume enclosing simplex in three dimensions, *SIAM J. Comput.*, 31 (2002), 1339–1357.