Algorithmica (2002) 33: 150–167
DOI: 10.1007/s00453-001-0096-5

Algorithmica

Efficiently Approximating Polygonal Paths in Three and Higher Dimensions[1]

G. Barequet,[2] D. Z. Chen,[3] O. Daescu,[3] M. T. Goodrich,[2] and J. Snoeyink[4]

Abstract. We present efficient algorithms for solving polygonal-path approximation problems in three and higher dimensions. Given an n-vertex polygonal curve P in $\mathbb{R}^d$, $d \geq 3$, we approximate P by another polygonal curve P' of $m \leq n$ vertices in $\mathbb{R}^d$ such that the vertex sequence of P' is an ordered subsequence of the vertices of P. The goal is either to minimize the size m of P' for a given error tolerance ε (called the *min-#* problem), or to minimize the deviation error ε between P and P' for a given size m of P' (called the *min-ε* problem). Our techniques enable us to develop efficient near-quadratic-time algorithms in three dimensions and subcubic-time algorithms in four dimensions for solving the min-# and min-ε problems. We discuss extensions of our solutions to d-dimensional space, where $d > 4$, and for the L_1 and L_∞ metrics.

Key Words. Curve approximation, Parametric searching.

1. Introduction. In this paper we consider the problem of approximating an arbitrary n-vertex polygonal path P in three-dimensional space by another polygonal path P' whose vertices form an ordered subsequence of the vertices in the original path P. We further discuss this problem in the d-dimensional space, where $d \geq 4$, and extend our solutions for the L_1 and L_∞ metrics.

This problem arises in many applications, such as robotics, image processing, computer graphics, cartography, and data compression. In these applications it is often desirable to approximate a complex graphical or geometric object, which is specified by a polygonal path, by a simpler object that captures the essence of the original object yet achieves a certain degree of data compression [5], [27], [28]. For example, we have been asked to compress three-dimensional path data for optic nerves acquired in medical studies and of river networks in geographic information systems analysis of spawning

[1] Work on this paper by the first and the fourth authors has been supported in part by the U.S. Army Research Office under Grant DAAH04-96-1-0013. Work by the second and third authors has been supported in part by the National Science Foundation under Grant CCR-9623585. Work by the fourth author has been supported also by NSF Grant CCR-96-25289. Work by the fifth author has been supported by NSERC, CIES, and by the Center for Geometric Computing at Johns Hopkins University.

[2] Center for Geometric Computing, Department of Computer Science, Johns Hopkins University, Baltimore, MD 21218, USA. {barequet,goodrich}@cs.jhu.edu. (The first author is currently affiliated with the Faculty of Computer Science at The Technion—Israel Institute of Technology.)

[3] Department of Computer Science and Engineering, University of Notre Dame, Notre Dame, IN 46556, USA. {chen,odaescu}@cse.nd.edu.

[4] Department of Computer Science, University of North Carolina at Chapel Hill, NC 27599–3175, USA. snoeyink@cs.unc.edu.

Received January 10, 1999; revised November 8, 2000. Communicated by C.-K. Yap.
Online publication February 8, 2002.

habitat [24]. Path simplification is also important for devices with low speed, from pen plotters to the World-Wide Web.

In general, the goal of the path-approximation problem is to replace a (very) high-resolution path P with an approximate path P' that achieves significant time and space improvements while suffering only a small approximation error. Two factors come to play in such an approximation: the number of vertices in the approximation and the error between the approximation and the original path. These two factors give rise to two different versions of the path-approximation problem—the min-ε version and the min-# version. In the min-ε version one is given a parameter $m \leq n$ and asked to find an approximating path P' with at most m vertices whose *error*, ε, from the original path is minimized. This version of the problem is motivated by a desire to obtain the best approximation possible that achieves a certain degree of data compression. In the min-# version of the path-approximation problem one is instead given a parameter $\varepsilon > 0$ and asked to find the smallest-vertex path P' that is within *error* ε of the original path. This version of the problem is motivated by a desire to obtain the smallest-complexity path that maintains a certain level of accuracy. In this paper we study both versions of the d-dimensional path-approximation problem (where $d \geq 3$), for the usual metric L_2, and also for L_1 and L_∞.

One issue that immediately arises in the path approximation problem is that one must formalize the "error" between the original path P and the approximation P'. We believe that one of the most natural definitions is the *tolerance-zone* error measure [5], [21], [22], [26]. Define the *ε-tolerance zone* (for $\varepsilon > 0$) of a line segment $\overline{pq}$ to be the region of space (or "*zone*") that is the union of all radius-ε balls centered at points along the segment $\overline{pq}$ (see Figure 1). This definition is, of course, parameterized by the metric used to define radius-ε balls. We consider the usual Euclidean metric, L_2, as well as the L_1 and L_∞ distance metrics. We may then formalize the path-approximation problem as follows:

POLYGONAL PATH APPROXIMATION.
Given a polygonal path $P = (p_1, p_2, \ldots, p_n)$, find a path $P' = (p_{i_1}, p_{i_2}, \ldots, p_{i_m})$, such that:

- $i_j \in \{1, 2, \ldots, n\}$, for $j = 1, 2, \ldots, m$;
- $i_j < i_{j+1}$, for $j = 1, 2, \ldots, m-1$; and
- the subpath $(p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$, for $j = 1, 2, \ldots, m-1$, is completely contained in the ε-tolerance zone for the segment $\overline{p_{i_j} p_{i_{j+1}}}$.

If the subpath $P_{i_j, i_{j+1}} = (p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ is completely contained in the ε-tolerance zone for the segment $\overline{p_{i_j} p_{i_{j+1}}}$, we call $\overline{p_{i_j} p_{i_{j+1}}}$ an *approximating line segment* for $P_{i_j, i_{j+1}}$.

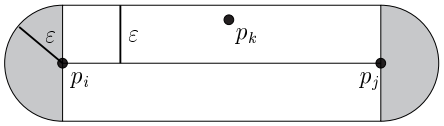

Fig. 1. The ε-tolerance zone of a single segment $\overline{p_i p_j}$.

In the min-ε version of the path-approximation problem we are given $m < n$ and wish to minimize ε; in the min-# version we are given $\varepsilon > 0$ and wish to minimize m.

We restrict our attention to the version of this problem that requires that $p_{i_1} = p_1$ and $p_{i_m} = p_n$. In the general version we only require that $i_1 \leq i_m$, and that the subpaths P_{1,i_1-1} and $P_{i_m+1,n}$ are completely contained in the ε-balls centered at P_{i_1} and P_{i_m}, respectively. We explain later in the Introduction how to reduce the general problem to its restricted version.

1.1. *Related Previous Work.* Imai and Iri [21], [22] and Melkman and O'Rourke [26] study the two-dimensional min-# and min-ε polygonal-path approximation problems. Using the tolerance-zone measure of error, as defined above, they achieve algorithms whose running times are $O(n^2 \log n)$ and $O(n^2 \log^2 n)$, respectively, and use $O(n^2)$ space, for the two problems. Chan and Chin [6] reduce the time complexity of both results by a logarithmic factor. Chen and Daescu [9] further show that the algorithms of [6] can use only $O(n)$ space without increasing their running times.

Varadarajan [31] studies the min-# and min-ε problems for two-dimensional polygonal paths that are monotone, i.e., any line parallel to the y axis intersects such a path in a point. Given a monotone approximation of a monotone path, Varadarajan measures the error as the maximum distance between the intersection points with some line parallel to the y axis. He gives $O(n^{4/3+\delta})$ time and space algorithms for both problems, where $\delta > 0$ is an arbitrarily small constant. However, those algorithms cannot be extended to the three-dimensional version of the problem or to the tolerance-zone error criterion in two dimensions. The recursive simplification heuristic of Douglas and Peucker [10], which is popular in GIS, does extend to curves in three dimensions in $O(n^2)$ worst-case time, but solves neither the min-# nor the min-ε problems.

There has been other less-related, but still significant, work done on other metrics for comparing two-dimensional polygonal paths [22] that do not necessarily use the same set of vertices. Arkin et al. [3], Alt and Godau [2], and Rote [29] describe several metrics for comparing polygonal curves. Guibas et al. [19] study different error measures for the min-# problem, with an eye toward those that could be implemented in $O(n \log n)$ time by greedy algorithms. Fleischer et al. [15] approximate polygonal shapes by inner and outer polygons and show their applications to the problems of polygon containment and of planar motion planning.

Previous work on the three-dimensional path-approximation problem, that uses the tolerance-zone measure of error, has given, to the best of our knowledge, algorithms with supercubic time bounds. Ihm and Naylor [20] give an $O(n^3 \log m)$ algorithm for the min-ε problem. Eu and Toussaint [14] restrict the polygonal chains to be monotonic with respect to one of the three coordinate axes, and define the approximation in terms of an *infinite-beam* measure of error. In this measure of error one requires that each subpath $(p_{i_j}, p_{i_j+1}, \ldots, p_{i_{j+1}})$ be completely contained in the ε-tolerance zone of the *infinite line* containing the segment $\overline{p_{i_j} p_{i_{j+1}}}$, called the infinite beam. When infinite beams are defined in terms of the L_1 or L_∞ metrics, Eu and Toussaint show how to achieve running times of $O(n^2)$ and $O(n^3)$ for the min-# and min-ε problems, respectively, for monotone three-dimensional polygonal paths. Under the Euclidean (L_2) metric, they consider the three-dimensional version of the problem in [31] and give algorithms that run in $O(n^3)$

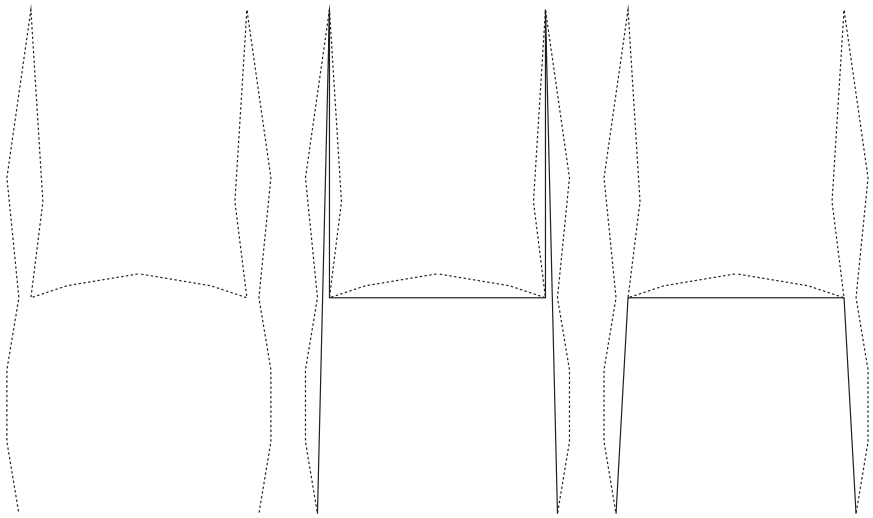

Fig. 2. Approximations using the tolerance-zone and the infinite-beams measures of error. (a) Original path, (b) tolerance zone, and (c) infinite beams.

and $O(n^3 \log n)$ time, respectively, for the min-# and min-ε problems on monotone three-dimensional polygonal paths.

We believe that the tolerance-zone measure of error for segments captures better the intuitive concept of the shape of a d-dimensional polygonal path. We illustrate in Figure 2 the difference between approximating a two-dimensional path with tolerance zones and with infinite beams.

1.2. *Our Results*. In this paper we give the first efficient algorithms for approximating a polygonal path in d dimensions, where $d \geq 3$, for general (nonmonotonic) paths, using the tolerance-zone measure of error for segments. We summarize the running times of our methods in Table 1. The important thing to note is that all of our bounds are subcubic, and most are nearly quadratic. Fairly straightforward bounds of $O(n^3)$ for min-# and $O(n^3 \log n)$ for min-ε follow by adapting previous two-dimensional results [6], [21], [22], [26]. Even so, our approach still follows the general framework of these earlier algorithms. To solve the min-# problem, we first build a graph G on the vertices of the input path P, where each edge represents a valid "shortcut," and we then perform

Table 1. Summary of our min-# and min-ε results.

	Three dimensions		$d \geq 4$ Dimensions	
Metric	Min-#	Min-ε	Min-#	Min-ε
L_2	$O(n^2 \log n)$	$O(n^2 \log^3 n)$	$O(n^{3-2/(\lfloor d/2 \rfloor+1)} \operatorname{polylog} n)$	$O(n^{7/3} \operatorname{polylog} n)^*$
L_1 and L_∞	$O(n^2)$	$O(n^2 \log n)$	$O(n^2)$	$O(n^2 \log n)$

*For $d = 4$ only.

a shortest-path computation in G. To solve the min-ε problem we perform a "binary-search-like" computation using the min-# algorithm as a "probe."

Our algorithms differ from previous approaches in some important ways, however. One of the fundamental differences is that we develop a d-dimensional generalization of the two-dimensional approach of Chan and Chin [6] for constructing the "shortcut" graph G. Rather than constructing this graph directly, we instead construct it as the intersection of two other graphs, $\overrightarrow{G}$ and $\overleftarrow{G}$, which respectively define tolerance zones in terms of "semi-infinite" beams (pointing in opposite directions). The difficult part of this computation is that, fixing an index $1 \leq i < n$, we must determine, for each index j with $i < j \leq n$, if the ray $\overrightarrow{p_i p_j}$ originating at p_i and passing through p_j intersects each radius-ε ball centered at a vertex p_k for $i < k < j$. We reduce this problem to the following data structuring problem:

OFF-LINE BALL-INCLUSION TESTING.
Maintain a collection of different-radius balls subject to a given sequence of the following operations:

- `insert`(B): Add ball B to the collection.
- `inside`(p): Determine if point p is inside the common intersection of the balls.

For $d = 3$, we refer to the problem above as the off-line disk-inclusion testing problem (the balls are disks on a three-dimensional sphere) and provide a data structure for solving this problem in amortized $O(\log n)$ time for inserts and queries, even though each disk insertion can cause $\Theta(n)$ structural changes to the common intersection. We use this data structure to derive an $O(n^2 \log n)$-time algorithm for the min-# path approximation problem. To turn this into an efficient algorithm for the min-ε problem we give an application of the parametric-search paradigm that is based upon a nontrivial parallel version of our off-line inclusion testing data structure.

When the ε-tolerance zones are defined using the L_1 or L_∞ metrics, we can do better than this. We present an $O(n^2)$-time algorithm for the min-# problem and an $O(n^2 \log n)$-time algorithm for the min-ε problem. For the latter algorithm we avoid using parametric search.

For $d > 3$, we show how to achieve a subcubic time for the min-# problem under the L_2 metric. We also give $O(n^2)$- and $O(n^2 \log n)$-time algorithms (for a fixed $d > 3$) for the min-# and min-ε problems, respectively, under the L_1 and L_∞ metrics. The constants hidden in the big-Oh notation depend on d and on the used metric: they are $O(d2^d)$ for L_1 and $O(d^2)$ for L_∞.

The general version of the problem can be reduced to the version solved in this paper (where $i_1 = 1$ and $i_m = n$) by adding to G two vertices p_0 and p_{n+1}, and edges $p_0 p_i$ (resp., $p_i p_{n+1}$), for $1 \leq i \leq n$, if all the vertices p_j, for $1 \leq j < i$ (resp., $i < j \leq n$) are contained in an ε-ball centered at p_i. This additional processing takes $O(n^2)$ time, and we then solve the restricted problem for p_0 and p_{n+1}.

The rest of the paper is organized as follows. In Section 2 we discuss some useful structures for solving the min-# problem in three dimensions. In Section 3 we solve the off-line disk-inclusion problem, which is a central ingredient of our algorithms. Then in Section 4 we present an algorithm for solving the min-ε problem in three dimensions.

In Section 5 we generalize our algorithms for higher dimensions. Finally, in Section 6 we generalize our algorithms for the L_1 and L_∞ metrics. We terminate in Section 7 with some concluding remarks.

2. The Min-# Problem in Three Dimensions. In this section we present the main ideas behind our $O(n^2 \log n)$-time algorithm for solving the three-dimensional min-# problem. The important part of this computation involves a reduction to the off-line ball-inclusion testing problem, which we solve in Section 3.

Suppose then that we are given a three-dimensional polygonal curve $P = (p_1, p_2, \ldots, p_n)$, and an error value $\varepsilon > 0$, and we wish to find the smallest $m \leq n$ such that there is a subpath $P' = (p_{i_1}, p_{i_2}, \ldots, p_{i_m})$ that satisfies the ε-tolerance-zone criterion (with $i_1 = 1$ and $i_m = n$). As outlined in the Introduction, we solve this problem by the following general approach:

1. Construct a graph $G = (V, E)$, where V is the vertex set of P and E consists of directed edges (p_i, p_j), $1 \leq i < j \leq n$, such that $\overrightarrow{p_i p_j}$ is an approximating line segment of the subchain $(p_i, p_{i+1}, \ldots, p_j)$ of P.
2. Find a shortest path from p_1 to p_n in G (i.e., a path with the fewest number of edges). This path gives us the sought approximating curve P'.

Since the second step is easily solved by a breadth-first search computation, we concentrate on an efficient technique for constructing G from P.

As mentioned above, we construct G as the intersection of two graphs $\overrightarrow{G}$ and $\overleftarrow{G}$. Say that edge (p_i, p_j) is in the graph $\overrightarrow{G}$ if the ray $\overrightarrow{p_i p_j}$ originating at p_i and passing through p_j intersects each radius-ε ball $B(p_k, \varepsilon)$ centered at a vertex p_k for $i < k < j$. Likewise, say that edge (p_i, p_j) is in the graph $\overleftarrow{G}$ if the ray $\overleftarrow{p_i p_j}$ originating at p_j and passing through p_i intersects each $B(p_k, \varepsilon)$, for $i < k < j$. It is an easy observation that an edge (p_i, p_j) is in G if and only if it is in both $\overrightarrow{G}$ and $\overleftarrow{G}$. Since the constructions of these two graphs are symmetric computations, we concentrate further upon the construction of the graph $\overrightarrow{G}$.

In this section we show how to reduce the problem of determining the edges in $\overrightarrow{G}$ to the off-line disk-inclusion problem. We fix an index i, with $1 \leq i < n$, and focus on computing all the edges of the form (p_i, p_j) in $\overrightarrow{G}$, for $i < j \leq n$. Say that the ray $\overrightarrow{p_i p_j}$ *properly approximates* the polygonal chain $(p_i, \ldots, p_j)$ if it intersects every ball $B(p_k, \varepsilon)$ for $i < k < j$. Now imagine a "large" sphere S_i centered at p_i, and project from p_i all the balls $B(p_k, \varepsilon)$ $(i < k < j)$ onto this sphere. Each ball $B(p_k, \varepsilon)$ is then projected to a disk D_k on the sphere S_i (see Figure 3).

It is trivial to show that:

LEMMA 1. *The ray $\overrightarrow{p_i p_j}$ is a proper approximation if and only if*:

1. *The intersection of the set of disks $\mathcal{D}(i, j) = \{D_{i+1}, \ldots, D_{j-1}\}$ on S_i is nonempty.*
2. *The projection of p_j from p_i onto S_i is found within the intersection of the disks in $\mathcal{D}(i, j)$.*

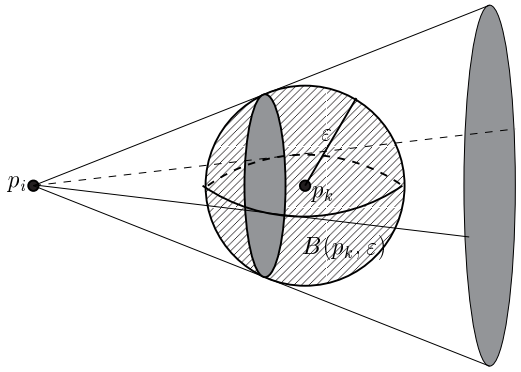

Fig. 3. The projection of the radius-ε ball $B(p_k, \varepsilon)$ centered at p_k onto the sphere S_i. (S_i is not shown.)

This lemma gives us the ability to reduce the construction of all edges of the form (p_i, p_j) in $\overrightarrow{G}$, for our fixed p_i, to the off-line disk-inclusion problem. Specifically, we define the operations:

- `insert`(D_j): Inserts disk D_j to a data structure, where D_j is the projection of $B(p_j, \varepsilon)$ onto the sphere S_i using p_i as the center of projection.
- `inside`(p'_j): Returns "true" or "false," depending on whether point p'_j, the projection of p_j onto S_i using p_i as the center of projection, is contained by all the disks currently in the data structure.

We then construct a sequence Σ_i of `insert` and `inside` operations, as follows:

1: Initialize Σ_i to insert a disk containing the entire sphere S_i.
2: **for** $j = i + 1$ to n
3: Append the operation `inside`(p'_j) to Σ_i.
4: Append the operation `insert`(D_j) to Σ_i.
5: **end for**

Again, it is straightforward to show that:

LEMMA 2. *The edge* (p_i, p_j) *is in* $\overrightarrow{G}$ *if and only if the response to the operation* `inside`(p'_j) *in* Σ_i *is "true."*

Thus, we have reduced the problem of constructing $\overrightarrow{G}$ to the problem of solving $n - 1$ independent instances of the off-line disk-inclusion problem. We outline in the next section how we can solve this problem efficiently.

3. The Off-Line Disk-Inclusion Problem. Suppose we are given a sequence Σ of n `insert`(D_i) and `inside`(p_i) operations defining an instance of the off-line disk-inclusion problem. In this section we show how to determine the answers to all the `inside`(p_i) queries in $O(n \log n)$ total time. Before we provide our solution we remark

that Sharir [30] and Eppstein [12] give data structures for maintaining the on-line intersection of a collection of equal-radius disks. This data structure cannot be applied to our disk-inclusion problem, however, as the disks in our problem do not in general have equal radii.

We describe our solution to the off-line disk-inclusion problem. We begin by building a complete binary tree T such that each leaf of T is associated with a different disk given in the sequence Σ. We order the leaves of T from left to right in the order they are given in the sequence Σ. For each internal node v in T, let $I(v)$ denote a representation of the common intersection of the disks stored at leaf-descendants of v. We compute $I(v)$ for all vertices v in T in a divide-and-conquer manner. In each node we maintain the lower and upper envelopes of the intersections of the respective disks. Over all, the total time required for constructing $I(v)$ for all vertices v is $O(n \log n)$. It is important to observe that each leaf of T is associated with a different index in Σ. Thus, if an `inside` query appears in position i in Σ, we can form a search path π_i from the root of T to the leaf corresponding to the first insertion after position i. Say that a node w of T is on the *left fringe* of π_i if w is not on π_i but is a left child of a node on π_i. Noting that the `inside` query is a decomposable search problem, we can answer an `inside`(p) query for position i by determining if p is inside $I(w)$ for each w that is a left fringe node of π_i (answering "true" if and only if it is inside them all). For each w, the task is therefore to determine whether p is above (resp., below) the lower (resp., upper) envelope of $I(v)$. This amounts to locating the envelope arcs below and above p and then deciding whether p is indeed contained in $I(v)$. By using the fractional-cascading technique of Chazelle and Guibas [7], [8] we are able to answer each `inside` query in Σ in $O(\log n)$ time. This gives us the following:

THEOREM 3. *Given a sequence Σ of n disk* `insert` *updates and point* `inside` *queries, one can answer all the* `inside` *queries in Σ in $O(n \log n)$ time.*

Combining this with the discussion from the previous section immediately gives us the following:

THEOREM 4. *Given an n-vertex polygonal path P in three-dimensional space, and a parameter $\varepsilon \geq 0$, one can find the fewest-vertex approximating path P' to P in $O(n^2 \log n)$ time under the ε-tolerance-zone criterion.*

4. The Min-ε Problem in Three Dimensions. We now turn to the min-ε problem. Suppose then that we are given a three-dimensional polygonal curve $P = (p_1, p_2, \ldots, p_n)$, and an integer parameter $m < n$, and we wish to find the smallest $\varepsilon \geq 0$ such that there is a subpath $P' = (p_{i_1}, p_{i_2}, \ldots, p_{i_m})$ that satisfies the ε-tolerance-zone criterion. For solving the min-ε problem we parallelize the min-# algorithm and invoke it as a subroutine in a parametric search algorithm, where the sought solution ε^* is the minimum ε for which there exists an approximating curve of no more than m vertices.

In fact, we do not really have to parallelize the entire min-# algorithm described above. We need only parallelize the portions of the algorithm that depend upon the parameter

ε. In the case of our min-# algorithm, the only place where ε plays a role is in the sizes of the disks that are used in the off-line disk-inclusion algorithm. Thus, the only part of the min-# algorithm that we need to parallelize is the off-line disk-inclusion algorithm itself.

The following problem is a key to our algorithm: In the plane, given n different-radius disks $d_1, d_2, \ldots, d_n$ and n points $a_1, a_2, \ldots, a_n$, determine for every $j = 1, 2, \ldots, n$, whether $a_j \in \Pi_j = \bigcap_{i=1}^{j} d_i$. We now give an $O(\log n)$-time, $O(n)$-processor algorithm for solving this problem in the parallel comparison model.

Our algorithm is based on Goodrich's paradigm [16] for computing the upper envelope of n surfaces in the so-called k-intersecting class. We first build an n-leaf complete binary tree T. The ith leaf of T stores d_i and a_i. Every internal node v of T is associated with (1) the intersection $I(v)$ of the disks stored at the leaves of the subtree T_v rooted at v (of course, $I(v)$ needs to be computed); and (2) the point set $A(v)$ consisting of all the points a_i stored at the leaves of T_v. Observe that, for every j, the question of whether $a_j \in \Pi_j = \bigcap_{i=1}^{j} d_i$ can be answered by checking whether $a_j \in I(v)$ for $O(\log n)$ nodes v of the tree T.

One could use a naive parallel algorithm for computing the $I(v)$'s and checking whether $a_j \in I(v)$: Go up the tree T level by level, starting from the leaves; for every node v at each level, first compute $I(v)$ by merging the sorted sequences of the boundary vertices of $I(x)$ and $I(y)$, where x and y are the left and right children of v, and then check whether the points of $A(y)$ belong to $I(x)$. This checking can also be done by merging a sequence of $A(y)$ (sorted lexicographically) with the boundary vertices of $I(x)$. At the root of T, we have the answers for all questions of whether $a_j \in \Pi_j$. This naive algorithm can be easily implemented in $O(\log n \log\log n)$ time and $O(n/\log\log n)$ processors in the CREW PRAM model.

To obtain a faster algorithm in the parallel comparison model, we pipe-line the merge operations in computing $I(v)$ from $I(x)$ and $I(y)$ and in checking whether $a_j \in I(v)$, as in Goodrich's paradigm. A logarithmic number of stages are used for this pipe-lined procedure. Every $I(v)$ is maintained as a linked list. At the end of each stage t, a list $I_t(v)$ is stored at v. We say v is *full* when $I_t(v) = I(v)$. The merge at the children of v is pipe-lined to v by maintaining a list which is an *approximately uniform subsequence* (see [16]) of $I_t(v)$. Then $I_{t+1}(v)$ is defined as $I_t(x) \cup I_t(y)$. The information for the points in $A(v)$ can be maintained and pipe-lined in a similar fashion as for $I(v)$. These structures at every node v can be maintained in $O(1)$ time per stage. By following the techniques of [16] it easily follows that the problem of checking whether $a_j \in \Pi_j$ for every j is solvable in $O(\log n)$ time using n processors in the parallel comparison model.

We then use this parallel algorithm to drive a parametric-search algorithm [25]. We simulate each step of this parallel algorithm sequentially, noting that some of the computations in this step depend upon whether the optimal value ε^* falls in specific intervals $[\varepsilon_1, \varepsilon_2]$. We find the median ε_i in linear time and use the min-# algorithm as a subroutine, running in $O(n^2 \log n)$ time, to determine if ε^* is above or below this median. The answer to this question resolves half of the comparisons for this parallel step. We can therefore repeat this process recursively and in $O(\log n)$ calls to the min-# subroutine we will have resolved all the comparisons for this parallel step. We then repeat this process for the next parallel step. Since our parallel algorithm takes time $O(\log n)$, this implies that the total running time for our min-ε algorithm is $O(n^2 \log^3 n)$. Thus, we have the following:

THEOREM 5. *Given an n-vertex polygonal path P in three-dimensional space, and an integer parameter $m < n$, one can find the best m-vertex approximating path P' to P in $O(n^2 \log^3 n)$ time under the ε-tolerance-zone criterion.*

The above discussion assumes that the underlying metric for defining radius-ε balls is the L_2 metric. In Section 6 we explore improvements and simplifications that can be achieved if we use the L_1 or L_∞ metrics for this purpose.

5. The Min-# and Min-ε Problems in $d \geq 4$ Dimensions. As in three dimensions, the key to solving the min-# problem in d dimensions is to reduce it to the off-line ball-inclusion problem in a lower-dimensional space and apply decomposability to the ball-inclusion problem. We describe below the ideas of this technique.

In order to approximate a d-dimensional curve, we solve n off-line ball-inclusion problems, each performing $O(n)$ inclusion tests of points in $(d-1)$-balls. By a standard lifting map, testing if a point is in a set of $(d-1)$-balls is equivalent to asking whether a point in d dimensions is below the lower envelope of a collection of d-dimensional hyperplanes: The query point is lifted to a paraboloid, and the balls are mapped to hyperplanes; the intersections of the d-dimensional hyperplanes with the paraboloid, projected into $d-1$ dimensions, are the original balls. (See [4] for more details.)

Thus, to solve one instance of off-line ball inclusion, we build a static data structure for answering ray-shooting queries (see [13] and [23]), using a parameter s to control a tradeoff between query time and space/preprocessing time. In particular, we build a ray-shooting data structure in $O(s \operatorname{polylog} n)$ space and preprocessing time, that can answer each ray-shooting query in $O(n \log n / s^{1/\lfloor d/2 \rfloor})$ time. In the Appendix we give a more complete discussion of the time/space tradeoff for the off-line ball-inclusion problem.

Now, to solve a min-# problem, we have to build n data structures and ask $O(n^2)$ queries. If we balance the time required for the preprocessing and for performing the queries, we find out that $s = O(n^{2-2/(\lfloor d/2 \rfloor+1)} \operatorname{polylog} n)$ and consequently obtain an $O(n^{3-2/(\lfloor d/2 \rfloor+1)} \operatorname{polylog} n)$-time algorithm. For $d = 4$ or 5, for example, this gives us an algorithm for the min-# problem (under the L_2 metric) that runs in $O(n^{7/3} \operatorname{polylog} n)$ time. Only for $d > 19$ does the running time of the algorithm exceed $O(n^{2.8} \operatorname{polylog} n)$.

Since each ray-shooting data structure has subquadratic space, and we only need one structure at a time, the amount of space required by this algorithm is still dominated by the maximum size of the graph G, which is $\Theta(n^2)$. Thus, for d dimensions we obtain:

THEOREM 6. *Given an n-vertex polygonal path P in d-dimensional space, and an integer parameter $m < n$, one can find the best m-vertex approximating path P' to P in $O(n^{3-2/(\lfloor d/2 \rfloor+1)} \operatorname{polylog} n)$ time by using $O(n^2)$ space.*

For the min-ε problem in four dimensions we need only observe (as for the three-dimensional version of the problem) that we have to parallelize only the steps in the sequential min-# algorithm that depend on ε. For this purpose we use the ray-shooting data structure of [17] and [18]. With some preprocessing, the construction of this data structure does not depend on the value of ε. In this data structure the points are sorted

according to their respective coordinates in parallel. As in the three-dimensional case, we perform the ray-shooting queries in parallel, which costs us an $O(\log^2 n)$ factor in the running time. Over all, we obtain an $O(n^{7/3} \operatorname{polylog} n)$-time algorithm for the four-dimensional min-ε problem under the L_2 metric. Note that this method does not easily extend to dimensions higher than four because of the possibly large complexities of the ray-shooting cells in high dimensions.

6. The Min-# and Min-ε Problems for L_1 and L_∞ Metrics in d Dimensions. In this section we present our algorithms for the min-# and min-ε problems under the L_1 and L_∞ metrics. Decomposability of on-line ball inclusion, and the fact that the unit balls in these metrics are polytopes of constant complexity for constant d, allow us to give efficient general solutions in d dimensions. We describe our algorithm for the ε-tolerance-zone measure of error.

Recall that to compute the graph $\overrightarrow{G}$ we had to determine, for a fixed index $1 \le i < n$ and each $i < j \le n$, whether the ray $\overrightarrow{p_i p_j}$ intersects each radius-ε ball centered at a vertex p_k for $i < k < j$. As Figure 3 illustrates, the rays from p_i that intersect the radius-ε ball around p_k form a *cone*. If the ball at p_k contains p_i, then the cone is the entire space.

Under L_1, L_∞, or any metric whose ball is a polytope whose complexity depends only on d, we can reduce the problem of constructing the graph $\overrightarrow{G}$ to a constant number of instances of the one-dimensional version of the ball-inclusion problem, which is one-dimensional off-line interval-inclusion testing.

LEMMA 7. *In d dimensions, the question of whether a ray $\overrightarrow{p_i p_j}$ intersects each ball $B(p_k, \varepsilon)$, for $i < k < j$, reduces to $O(d^2)$ instances of off-line interval-inclusion testing for the L_∞ metric or $O(d2^d)$ instances for the L_1 metric.*

PROOF. Construct the cones of the rays from p_i through radius-ε balls centered at each p_k. The ray $\overrightarrow{p_i p_j}$ intersects all balls if and only if it is in the intersection of these cones. The key is to observe that $(d-1)$-dimensional facets that define the boundary of these cones can be separated into a constant number of families, where each family projects to a halfplane in two dimensions.

Since the metric is defined by a convex polytope, each cone is a convex polyhedral simplex: it is the intersection of hyperplanes through p_i and the $(d-2)$-dimensional faces, called *ridges*, that are on the silhouette of the ball at p_k. For example, ridges in three dimensions are one-dimensional edges of the 3-cube. Different views of a 3-cube give cones of different cross sections, as shown in Figure 4, but all boundary facets of the corresponding cone are planes through p_i and a ridge of the 3-cube.

To determine whether a point p_j is in the intersection of cones defined by balls at p_k for $i < k < j$, we form an instance of off-line interval-inclusion testing for each ridge: A translated version of the ridge r generates a halfspace whenever it appears on the silhouette when viewed from p_i; since all halfspaces generated by translates of r are bounded by hyperplanes parallel to r that contain p_i, we can project each translate onto the circle at infinity that is contained in the 2-flat through p_i and orthogonal to ridge r.

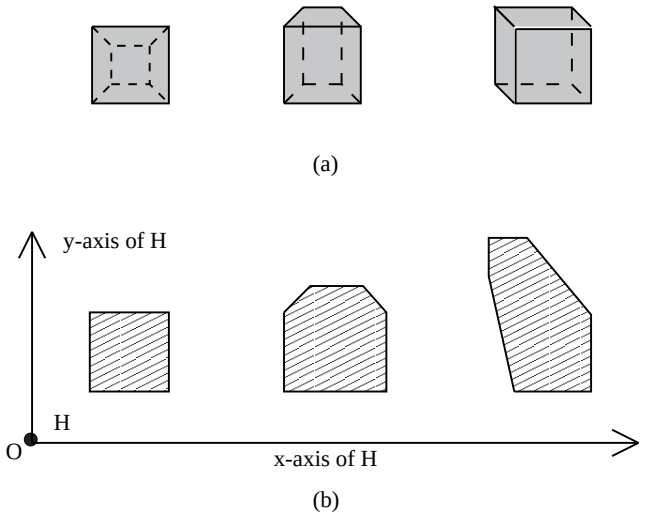

Fig. 4. (a) Three different views of a cube from a point. (b) Three corresponding cross sections for a cone.

(Knowing the projection, the ridge, and p_i we can reconstruct the hyperplane that forms one of the cone's boundary facets.)

By decomposability, point p_j is in the intersection of cones if and only if it is in the intersection of their halfspaces, which happens if and only if the projection of p_j is included in each instance of interval-inclusion testing. Thus, $\overrightarrow{G}$ can be constructed by solving an instance of interval-inclusion testing for each ridge and intersecting the solutions.

We complete the proof by counting ridges in L_∞ and L_1 polytopes. In the d-cube, every pair of nonparallel planes generates a ridge, giving $2d(d-1)$ ridges. In the d-crosspolytope (a d-dimensional L_1 ball), a ridge is obtained by choosing positive or negative vertices for $d-1$ of the d coordinate directions, giving $d2^{d-1}$ ridges. Not every ridge is on the silhouette: for the cube at most $2d$ will be. Ridges not on the silhouette and ridges whose cubes contain p_i do not contribute constraints. □

It is now straightforward to solve the min-# problem in $O(n^2)$ time for any constant dimension d with a constant-sized polytope for the metric. Note that the reduction to a constant number (depending only on d) of two-dimensional problems allows us to use only $O(n)$ space, as in [9].

THEOREM 8. *Given an n-vertex polygonal path P in d dimensions, and a parameter $\varepsilon \geq 0$, one can find the fewest-vertex approximating path P' to P in $O(n^2)$ time and $O(n)$ space under the ε-tolerance-zone criterion using the L_∞ or L_1 error metric.*

We solve the d-dimensional min-ε problem in $O(n^2 \log n)$ time without using parametric search. Instead, we can explicitly construct the set, $ERR(P)$, of all candidate optimal ε values and then perform a simple binary search in this set using the min-#

algorithm as the "probe." We use the fact that the optimal ε value must be determined by some edge (p_i, p_j) in G such that the farthest point p_k, with $i < k < j$, from the segment $\overline{p_i p_j}$ is at a distance *exactly* ε from $\overline{p_i p_j}$. When we fix the starting point p_i, we can determine the farthest such point p_k for each edge $\overline{p_i p_j}$ in $O(n \log n)$ time. Thus, we can compute the set $ERR(P)$ in $O(n^2 \log n)$ time, and then perform a binary search in this quadratic-size set in $O(n^2 \log n)$ time using our quadratic-time min-# algorithm as the "probe."

We can actually achieve an $O(n)$ space bound, but to do so we cannot afford to maintain the error set $ERR(P)$ explicitly (since $|ERR(P)| = O(n^2)$). Instead, we make use of the sampling technique of [9], which allows us to store only $O(n)$ errors of $ERR(P)$, while still being able to perform binary search on the $O(n^2)$ errors. Therefore, we can achieve the following:

THEOREM 9. *Given an n-vertex polygonal path P in d dimensions, and an integer parameter $m \leq n$, one can find the best m-vertex approximating path P' to P in $O(n^2 \log n)$ time under the ε-tolerance-zone error criterion using the L_∞ or L_1 metric.*

7. Conclusion. In this paper we present near-quadratic-time algorithms for solving the min-# and min-ε polygonal-path approximation problem in three dimensions, and subcubic-time algorithms for solving similar problems in $d \geq 4$ dimensions. We also generalize our solutions for the L_1 and L_∞ metrics.

A major related open problem is to show lower bounds for the problems discussed in this paper.

Appendix. A Time/Space Tradeoff for the Off-Line Ball-Inclusion Problem. In this Appendix we analyze space–time tradeoffs for the problem:

PROBLEM 1 (Reporting Points in Halfspaces). Given n halfspaces and m points in E^d, report all points in the intersection of all halfspaces.

The solution to this problem is an important subroutine in $(d-1)$-dimensional off-line ball inclusion, as outlined in Section 5. Balls and points are lifted to the paraboloid to become hyperplanes and points, respectively, and the merge tree must solve the above problem.

In the discussion in this Appendix, we use the notation $O^*(\cdot)$ instead of the usual $O(\cdot)$ when we omit polylogarithmic factors and/or ε exponents. We can begin by using divide and conquer to build a query structure on the halfspaces [13], [23], and then query with the points. Balancing preprocessing and query time in divide and conquer gives the following:

THEOREM 10. *The problem of reporting n points in n halfspaces can be solved in time and space*

$$O^*(n^{2-2/(\lfloor d/2 \rfloor + 1)}).$$

PROOF. With $O^*(s)$ space and preprocessing time, the data structure can answer a query in $O^*(n^2/s^{1/\lfloor d/2 \rfloor})$ time for n queries. These balance at

$$s = n^{2-2/(\lfloor d/2 \rfloor + 1)}.$$ □

Divide-and-conquer approaches to compute many faces in arrangements exploit duality and the batched nature of the problem to obtain better bounds [1], [11]. Unfortunately, we cannot do so because our problem is not self-dual: we can solve the problem for few halfspaces and many points by intersecting halfspaces and querying the $n^{\lfloor d/2 \rfloor}$ structure with points, but for many halfspaces and few points we have to build the arrangement of points and query an m^d structure with hyperplanes. Thus we can only give a data structure that has better space usage for $d > 5$.

THEOREM 11. *The problem of reporting n points in n halfspaces can be solved in time*

$$O^*(n^{2-(d+\lfloor d/2 \rfloor - 2)/(d\lfloor d/2 \rfloor - 1)})$$

and space

$$O^*(n^{3/2-1/(4\lfloor d/2 \rfloor - 2)}).$$

We solve the problem by a recursive procedure $P(m, n)$, using four different approaches depending on the values of m and n:

1. If $n^{\lfloor d/2 \rfloor} \leq m$, then compute the intersection of the n halfspaces, preprocess for point location, and locate the m points. This takes $O^*(m)$ time and space.
2. If $\sqrt{n} \leq m < n^{\lfloor d/2 \rfloor}$, then choose a random sample of r of the n halfspaces, construct their intersection, and form less than $r^{\lfloor d/2 \rfloor}$ cells where the ith cell is cut by $n_i \leq n/r$ hyperplanes and contains m_i of the points. Points that are not in the intersection of the r sample hyperplanes will not be included in any cell. Recursively solve the problem for the cells that contain points. We choose

$$r = \left(\frac{m^2}{n}\right)^{1/(2\lfloor d/2 \rfloor - 1)},$$

   which implies that the total number of hyperplanes in all subproblems is bounded by

$$r^{\lfloor d/2 \rfloor - 1} = m^{1-1/(2\lfloor d/2 \rfloor - 1)} n^{1/2+1/(4\lfloor d/2 \rfloor - 2)}.$$

   (This, by the way, gives us our space bound when $m = n$.) Notice that r is constant when $n \approx m^2$, and $r = n$ when $m \approx n^{\lfloor d/2 \rfloor}$.
3. If $n \geq m^d$, then solve the problem in dual space by constructing the arrangement of the hyperplanes that are dual to the m points, processing for point location, and assigning the points dual to the n hyperplanes to their cells in the arrangement. We can then use graph search to determine which hyperplanes (dual to points) have no occupied cells above them in $O^*(n)$ time and space.

4. If $m^d > n > m^2$, then, in dual space, form the hyperplane arrangement of a sample of t of the m duals of points and decompose the cells into simplices such that the ith simplex is cut by $m_i \leq m/t$ hyperplanes dual to points and contains n_i points dual to hyperplanes. The primal view of this process is that the hyperplanes are partitioned into bundles that have the same orientation with respect to the sample points, and the points are distributed to each bundle that has halfspaces that do and do not contain the point. Recursively solve the problem for the bundles that contain points and are not outside of the intersection of all halfspaces outside of the bundle. We choose

$$t = \left(\frac{n}{m^2}\right)^{1/(d-2)},$$

which implies that the total number of points in all subproblems is bounded by

$$mt^{d-1} = \frac{n^{1+1/(d-2)}}{m^{1+2/(d-2)}}.$$

Notice that t is constant when $n \approx m^2$ and $t = m$ when $n \approx m^d$.

The sample sizes r and t are chosen so as to maintain the number of points at the square of the number of halfspaces.

If we let $P(m, n)$ denote the time to solve an instance of the problem with m points and n halfspaces, then we can show by induction that, for $\alpha = 1-(d-1)/(d\lfloor d/2\rfloor - 1)+\epsilon/2$ and $\beta = 1 - (\lfloor d/2\rfloor - 1)/(d\lfloor d/2\rfloor - 1) + \epsilon/2$,

$$P(m, n) \leq m^\alpha n^\beta + n \operatorname{polylog} + m \operatorname{polylog},$$

where "polylog" stands for terms polylogarithmic in n and m. We consider the following cases:

1. Base case $n^{\lfloor d/2\rfloor} \leq m$: As argued above, $P(m, n) \leq m$ polylog.
2. Case $n \leq m^2 < n^d$: We must pay for partitioning the points, distributing hyperplanes to cells that they cut, and recursive computation. Using the fact that $n_i \leq n/r$ and that the maximum for Hölder's inequality occurs when the points are evenly distributed, we observe that

$$\begin{aligned}
P(m, n) &\leq \sum_{\text{cell } i} (P(m_i, n_i) + n_i \operatorname{polylog} + m_i \operatorname{polylog}) \\
&\leq nr^{\lfloor d/2\rfloor - 1} \operatorname{polylog} + m \operatorname{polylog} + \sum_{\text{cell } i} P\left(m_i, \frac{n}{r}\right) \\
&\leq r^{\lfloor d/2\rfloor} \left(\frac{m}{r^{\lfloor d/2\rfloor}}\right)^\alpha \left(\frac{n}{r}\right)^\beta + nr^{\lfloor d/2\rfloor - 1} \operatorname{polylog} + m \operatorname{polylog} \\
&\leq m^\alpha n^\beta r^{\lfloor d/2\rfloor - \alpha\lfloor d/2\rfloor - \beta} + nr^{\lfloor d/2\rfloor - 1} \operatorname{polylog} + m \operatorname{polylog} \\
&\leq m^\alpha n^\beta + n \operatorname{polylog} + m \operatorname{polylog}.
\end{aligned}$$

3. Base case $n \geq m^d$: As argued above, $P(m, n) \leq n$ polylog.

Table 2. Time/space tradeoff (exponents shown).

	Theorem 10	Theorem 11		
d	Time and Space	Time	Improved Time	Space
3	1.0	1.0	1.0	1.0
4	1.333	1.428	1.4	1.333
5	1.333	1.444	1.428	1.333
6	1.5	1.588	1.571	1.4
7	1.5	1.6	1.588	1.4
8	1.6	1.677	1.666	1.428
9	1.6	1.685	1.677	1.428
10	1.667	1.734	1.727	1.444

4. Case $m^d > n > m^2$: We must pay for partitioning the hyperplanes, distributing the points, and recursive computation. Similar to Case 2,

$$\begin{aligned} P(m,n) &\le \sum_{\text{cell } i} (P(m_i,n_i) + m_i \operatorname{polylog} + n_i \operatorname{polylog}) \\ &\le mt^{d-1} \operatorname{polylog} + n \operatorname{polylog} + \sum_{\text{cell } i} P\left(\frac{m}{t}, n_i\right) \\ &\le t^d \left(\frac{m}{t}\right)^{\alpha} \left(\frac{n}{t^d}\right)^{\beta} + mt^{d-1} \operatorname{polylog} + n \operatorname{polylog} \\ &\le m^{\alpha} n^{\beta} t^{d-\alpha-d\beta} + mt^{d-1} \operatorname{polylog} + n \operatorname{polylog} \\ &\le m^{\alpha} n^{\beta} + n \operatorname{polylog} + m \operatorname{polylog}. \end{aligned}$$

A small improvement in time is possible in our application: since all points lie on a paraboloid, the recursion continues in only $O(t^{d-1})$ cells in case 4. One can therefore choose

$$\alpha = 1 - \frac{d-2}{(d-1)\lfloor d/2 \rfloor - 1} + \frac{\epsilon}{2}$$

and

$$\beta = 1 - \frac{\lfloor d/2 \rfloor - 1}{(d-1)\lfloor d/2 \rfloor - 1} + \frac{\epsilon}{2}$$

and obtain the running time $O^*(n^{2-(d+\lfloor d/2 \rfloor - 3)/((d-1)\lfloor d/2 \rfloor - 1)})$.

Table 2 shows the exponents on n for space and time in dimensions 3–10.

As we point out in Section 5, the space in this discussion is used only for solving the off-line ball-inclusion problem, while the total space used by the algorithm is dominated by the size of the graph G, which is $\theta(n^2)$ in the worst case.

References

[1] P. K. Agarwal, J. Matoušek, and O. Schwarzkopf, Computing many faces in arrangements of lines and segments, *SIAM Journal on Computing*, **27** (1998), 491–505.

[2] H. Alt and M. Godau, Measuring the resemblance of polygonal curves, *Proc. 8th Ann. ACM Symp. on Computational Geometry*, Berlin, pp. 102–109, 1992.

[3] E. M. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell, An efficiently computable metric for comparing polygonal shapes, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13** (1991), 209–216.
[4] K. Q. Brown, Geometric transforms for fast geometric algorithms, Ph.D. Thesis, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, PA, 1980.
[5] B. Buttenfield, Treatment of the cartographic line, *Cartographica*, **22** (1996), 1–26.
[6] W. S. Chan and F. Chin, Approximation of polygonal curves with minimum number of line segments or minimum error, *International Journal of Computational Geometry and Applications*, **6** (1996), 59–77.
[7] B. Chazelle and L. J. Guibas, Fractional cascading: I. A data structuring technique, *Algorithmica*, **1** (1986), 133–162.
[8] B. Chazelle and L. J. Guibas, Fractional cascading: II. Applications, *Algorithmica*, **1** (1986), 163–191.
[9] D. Z. Chen and O. Daescu, Space-efficient algorithms for approximating polygonal curves in two-dimensional space, *Proc. 4th Ann. Internat. Computing and Combinatorics Conf.*, Taipei, Lecture Notes in Computer Science, vol. 1449, pp. 45–54, Springer-Verlag, Berlin, 1998.
[10] D. H. Douglas and T. K. Peucker, Algorithms for the reduction of the number of points required to represent a digitized line or its caricature, *Canadian Cartographer*, **10** (1973), 112–122.
[11] H. Edelsbrunner, L. J. Guibas, and M. Sharir, The complexity and construction of many faces in arrangements of lines and of segments, *Discrete & Computational Geometry*, **5** (1990), 161–196.
[12] D. Eppstein, Faster construction of planar two-centers, *Proc. 8th ACM–SIAM Symp. on Discrete Algorithms*, New Orleans, LA, 131–138, 1997.
[13] J. Erickson, Space–time tradeoffs for emptiness queries, *SIAM Journal on Computing*, **29** (2000), 1968–1996.
[14] D. Eu and G. T. Toussaint, On approximation polygonal curves in two and three dimensions, *CVGIP: Graphical Models and Image Processing*, **56** (1994), 231–246.
[15] R. Fleischer, K. Mehlhorn, G. Rote, E. Welzl, and C.-K. Yap, Simultaneous inner and outer approximation of shapes, *Algorithmica*, **8** (1992), 365–389.
[16] M. T. Goodrich, Using approximation algorithms to design parallel algorithms that may ignore processor allocation, *Proc. 32nd Ann. IEEE Symp. on Foundations of Computer Science*, San Juan, Puerto Rico, pp. 711–722, 1991.
[17] M. T. Goodrich and R. Tamassia, Dynamic ray shooting and shortest paths in planar subdivisions via balanced geodesic triangulations, *Journal of Algorithms*, **23** (1997), 51–73.
[18] M. T. Goodrich and R. Tamassia, Dynamic trees and dynamic point location, *SIAM Journal on Computing*, **28** (1999), 612–636.
[19] L. J. Guibas, J. E. Hershberger, J. S. B. Mitchell, and J. S. Snoeyink, Approximating polygons and subdivisions with minimum link paths, *International Journal of Computational Geometry and Applications*, **3** (1993), 383–415.
[20] I. Ihm and B. Naylor, Piecewise linear approximations of digitized space curves with applications, in *Scientific Visualization of Physical Phenomina* (N. M. Patrikalakis, ed.), pp. 545–569, Springer-Verlag, Tokyo, 1991.
[21] H. Imai and M. Iri, Computational-geometric methods for polygonal approximations of a curve, *Computer Vision, Graphics, and Image Processing*, **36** (1986), 31–41.
[22] H. Imai and M. Iri, Polygonal approximations of a curve-formulations and algorithms, in *Computational Morphology* (G. Toussaint, ed.), pp. 71–86, North-Holland, Amsterdam, 1988.
[23] J. Matoušek and O. Schwarzkopf, On ray shooting in convex polytopes, *Discrete & Computational Geometry*, **10** (1993), 215–232.
[24] M. McAllister and J. Snoeyink, Medial axis generalisation of hydrology networks, *AutoCarto* 13: *ACSM/ASPRS Ann. Convention Technical Papers*, Seattle, WA, pp. 164–173, 1997.
[25] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *Journal of the ACM*, **30** (1983), 852–865.
[26] A. Melkman and J. O'Rourke, On polygonal chain approximation, in *Computational Morphology* (G. Toussaint, ed.), pp. 87–95, North-Holland, Amsterdam, 1988.
[27] B. K. Natarajan, On comparing and compressing piecewise linear curves, Technical Report, Hewlett Packard, 1991.
[28] B. K. Natarajan and J. Ruppert, On sparse approximations of curves and functions, *Proc. 4th Canadian Conf. on Computational Geometry*, St. John's, Newfoundland, pp. 250–256, 1992.

[29] G. Rote, A new metric between polygons, and how to compute it, *Proc*. 19*th Internat*. *Colloq*. *Automata*, *Languages*, *and Programming*, Vienna, Lecture Notes in Computer Science, vol. 623, pp. 404–415, Springer-Verlag, Berlin, 1992.

[30] M. Sharir, A near-linear algorithm for the planar 2-center problem, *Discrete & Computational Geometry*, **18** (1997), 125–134.

[31] K. R. Varadarajan, Approximating monotone polygonal curves using the uniform metric, *Proc*. 12*th Ann*. *ACM Symp*. *on Computational Geometry*, Philadelphia, PA, pp. 311–318, 1996.