Computational Geometry 46 (2013) 756–773

Contents lists available at SciVerse ScienceDirect

Computational Geometry: Theory and Applications

www.elsevier.com/locate/comgeo

Dynamic well-spaced point sets

Umut A. Acar [a], Andrew Cotter [b,*], Benoît Hudson [c], Duru Türkoğlu [d]

[a] Computer Science Department, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[b] Toyota Technological Institute at Chicago, Chicago, IL 60637, USA
[c] Autodesk, Inc., Montreal, QC, Canada
[d] Department of Computer Science, University of Chicago, Chicago, IL 60637, USA

ARTICLE INFO

Article history:
Received 11 August 2011
Accepted 22 November 2012
Available online 6 December 2012
Communicated by R. Fleischer

Keywords:
Well-spaced point set
Clipped Voronoi cell
Mesh refinement
Dynamic stability
Self-adjusting computation

ABSTRACT

In a *well-spaced point set* the Voronoi cells all have bounded aspect ratio. Well-spaced point sets satisfy some important geometric properties and yield quality Voronoi or simplicial meshes that are important in scientific computations. In this paper, we consider the dynamic well-spaced point set problem, which requires constructing a well-spaced superset of a dynamically changing input set, e.g., as input points are inserted or deleted. We present a dynamic algorithm that allows inserting/deleting points into/from the input in $O(\log \Delta)$ time, where Δ is the geometric spread, a natural measure that yields an $O(\log n)$ bound when input points are represented by log-size words. We show that this algorithm is time-optimal by proving a lower bound of $\Omega(\log \Delta)$ for a dynamic update. We also show that this algorithm maintains size-optimal outputs: the well-spaced supersets are within a constant factor of the minimum possible size. The asymptotic bounds in our results work in any constant dimensional space. Experiments with a preliminary implementation indicate that dynamic changes may be performed with considerably greater efficiency than re-constructing a well-spaced point set from scratch. To the best of our knowledge, these are the first time- and size-optimal algorithms for dynamically maintaining well-spaced point sets.

1. Introduction

Given a hypercube B in $\mathbb{R}^d$, we call a set of points $\mathsf{M} \subset B$ *well-spaced* if for each point $p \in \mathsf{M}$ the ratio of the distance to the farthest point of B in the Voronoi cell of p divided by the distance to the nearest neighbor of p in M is small [32]. Well-spaced point sets are strongly related to meshing and triangulation for scientific computing, which require meshes to have certain qualities. In two dimensions, a well-spaced point set induces a Delaunay triangulation with no small angles, which is known to be a good mesh for the finite element method. In higher dimensions, well-spaced point sets can be post-processed to generate good simplicial meshes [8,20]. The Voronoi diagram of a well-spaced point set is also immediately useful for the control volume method [22].

A well-spaced superset M of a point set N may be constructed by inserting so-called *Steiner* points, although one must take care to insert as few Steiner points as possible. We call the output and such an algorithm *size-optimal* if the size of the output, |M|, is within a constant factor of the size of the smallest possible well-spaced superset of the input. This problem has been studied since the late 1980s (e.g., [6,10,26]), with several recent results obtaining fast runtime [14,16,31].

We are interested in the dynamic version of the problem, which requires maintaining a well-spaced output (M) while the input (N) changes dynamically due to insertion and deletion of points. Upon a modification to the input, the dynamic

* Corresponding author.
E-mail addresses: umut@cs.cmu.edu (U.A. Acar), cotter@ttic.edu (A. Cotter), benoit.hudson@autodesk.com (B. Hudson), duru@cs.uchicago.edu (D. Türkoğlu).

0925-7721/$ – see front matter
http://dx.doi.org/10.1016/j.comgeo.2012.11.007

algorithm should efficiently update the output, preserving size-optimality with respect to the new input. There has been relatively little progress on solving the dynamic problem. Existing solutions either do not produce size-optimal outputs (e.g., [9,25]) or they are asymptotically no faster than running a static algorithm from scratch [12,21,23].

In this paper, we present a dynamic algorithm for the well-spaced point set problem. Our algorithm always returns size-optimal outputs, and requires worst-case $O(\log \Delta)$ time for an input modification (an insertion or a deletion). Here, Δ is the *geometric spread*, a common measure, defined as the ratio of the diameter of the input set to the distance between the closest pair of points in the input. Our update runtime is optimal in the worst-case and our algorithms consume linear space in the size of the output. If the geometric spread is polynomially bounded in the size of the input, then $\log \Delta = O(\log n)$ (e.g., when the input is specified using $\log n$-bit numbers). For the purposes of our bounds, we assume the dimension of the space, d, to be an arbitrary constant.

To solve the dynamic problem, we first present an efficient construction algorithm for generating size-optimal, well-spaced supersets (algorithm in Sections 5 and 6, proofs in Sections 7 and 8). In addition to the output, the construction algorithm builds a *computation graph* that represents the operations performed during the execution and the dependencies between them. A key property of this algorithm is that it is *stable* in the sense that when run with similar inputs, e.g., that differ in only one point, it produces similar computation graphs and outputs. We make this property precise by describing a *distance* measure between the computation graphs of two executions and bounding this distance by $O(\log \Delta)$ when inputs differ by a single point (Section 9). Taking advantage of this bound, we design a change-propagation algorithm that performs dynamic updates in $O(\log \Delta)$ time by identifying the operations that are affected by the modification to the input and deleting/re-executing them as necessary (Section 10). For the lower bound, we show that there exist inputs and modifications that require $\Omega(\log \Delta)$ Steiner points to be inserted into/deleted from the output (Section 11).

The efficiency of our dynamic update algorithm directly depends on stability. In order to achieve stability, we use several techniques in the design of our construction algorithm. Generalizing the recently suggested choices of Steiner points [14,18], we propose an approach for picking Steiner points by making local decisions only, using *clipped Voronoi cells*. Picking Steiner points locally makes it possible to structure the computation into $\Theta(\log \Delta)$ *ranks*, inductively ensuring that at the end of each rank the points up to that rank are well-spaced [31]. Processing points in rank order alone does not guarantee stability: we further partition points at a given rank into a constant number of *color* classes such that the points in each color class depend only on the points in the previous color classes. These techniques enable us to process each point only once and help isolate and limit the effects of a modification. Furthermore, our dynamic update algorithm returns an output and a computation graph that are isomorphic to those that would be obtained by re-executing the static algorithm with the modified input (Lemma 10.2). Consequently, the output remains both well-spaced and size-optimal with respect to the modified input (Theorem 10.3).

The approach of designing a stable construction algorithm and then providing a dynamic update algorithm based on change propagation is inspired by recent advances on *self-adjusting computation* (e.g., [2,3,13,19]). In self-adjusting computation, programs can respond automatically to modifications to their data by invoking a change-propagation algorithm [1]. The data structures required by change propagation are constructed automatically. Our computation graphs are abstract representations of these data structures. Similarly our dynamic update algorithms are adaptations of the change-propagation algorithm for the problem of well-spaced point sets. Self-adjusting computation has been found to be effective in kinetic motion simulation of three-dimensional convex hulls [3]. Although these initial findings are empirical, they have motivated the approach that we present in this paper. Since our approach takes advantage of the structure of a static algorithm to perform dynamic updates, it can be viewed as a dynamization technique, a technique which has been used effectively for a relatively broad range of algorithms (e.g., [7,11,24,27]).

To assess the effectiveness of the proposed dynamic algorithm, we present a prototype implementation, and report the results of an experimental evaluation (Section 12). Our experimental results confirm our theoretical bounds, and demonstrate that dynamic updates to an existing well-spaced point set can be performed far more cheaply than re-computing from scratch. These results suggest that a well-optimized implementation can perform very well in practice.

This paper is the journal version of the following two abstracts: *An efficient query structure for mesh refinement* published in the proceedings of the 20th Annual Canadian Conference on Computational Geometry [17], and *Dynamic well-spaced point sets* published in the proceedings of the 26th Annual Symposium on Computational Geometry [4].

2. Preliminaries

We present some definitions used throughout the paper, describe the technique that we use for selecting Steiner vertices, and present an overview of the point location data structure we use in our algorithms.

Given a set of points N, we define the *geometric spread* (Δ) to be the ratio of the diameter of N to the distance between the closest pair in N. We say that a d-dimensional hypercube B is a *bounding box* if $\mathsf{N} \subset B$ and each edge of B has length within a constant factor of the diameter of N. Without loss of generality, we take the bounding box of N to be $B = [0,1]^d$. Given N as input, our algorithm constructs a well-spaced output $\mathsf{M} \subset B$ that is a superset of N. We use the term *point* to refer to any point in B and the term *vertex* to refer to the input and output points. Consider a vertex set $\mathscr{M} \subset B$. The *nearest-neighbor distance of v in $\mathscr{M}$*, written $\mathrm{NN}_{\mathscr{M}}(v)$, is the distance from v to the nearest other vertex in $\mathscr{M}$. The *Voronoi cell* of v in $\mathscr{M}$, written $\mathrm{Vor}_{\mathscr{M}}(v)$, consists of points $x \in B$ such that for all $u \in \mathscr{M}$, $|vx| \leqslant |ux|$. Following Talmor [32], a vertex v is