Available online at www.sciencedirect.com
SCIENCE DIRECT®
Information Processing Letters 93 (2005) 37–42
Information Processing Letters
www.elsevier.com/locate/ipl

HCPO: an efficient insertion order for incremental Delaunay triangulation

Sheng Zhou *, Christopher B. Jones

School of Computer Science, Cardiff University, Cardiff, CF24 3XF, UK

Received 19 December 2003; received in revised form 15 June 2004

Communicated by R. Backhouse

Keywords: Computational geometry; Incremental Delaunay triangulation; Insertion order

1. Introduction

Among the five flavors [15] of sequential 2D Delaunay triangulation algorithms, incremental insertion methods (e.g., [8]) are most popular mainly because they are potentially dynamic, simple to implement and easy to be generalized to higher dimensions. However, normally they are not regarded as among the fastest methods.

The basic principle of constructing Delaunay triangulation (DT) by incremental insertion is well known (see textbooks such as [2] for detail). A point is inserted into the triangulation in two steps: *point location* to finds the triangle or component of the triangle in which the point lies, and *triangulation update* to restore the Delaunay property of the triangulation. Point location is dominated by an orientation (normally counterclockwise, or CCW) test. Triangulation update mainly involves repeated use of in-circle tests and (assuming Lawson's algorithm [8] is used) edge flips. To a great extent, the cost of triangulation may be measured more objectively by counting these operations [15] than by plain running time.

For n points inserted in random order, the expected number of structural changes is $O(n)$ [7] (below 9 in-circle tests and 3 edge flips per insertion). The number of CCW tests depends on the adopted location scheme and point insertion order. One class of commonly used point location schemes is triangulation-walking [5,8,9], which starts from a "*search-hint*" (an existing triangle, or one of its edges or vertices which reference it) and "walks" through triangles towards the target location. Obviously if the search-hint is proximal to the target, fewer triangles will be walked through and fewer CCW tests will be performed. The jump-and-walk (J&W) scheme [9] in *Triangle* [12] performs 63.2 CCW tests per insertion on a 10k uniform dataset. The dynamic grid-based bucketing scheme [15] performs an average of 10.5 tests regardless of dataset

* Corresponding author.
E-mail addresses: gis_zhou@yahoa.co.uk (S. Zhou), c.b.jones@cs.cf.ac.uk (C.B. Jones).

0020-0190/$ – see front matter
doi:10.1016/j.ipl.2004.09.020

size. In comparison, Dwyer's divide-and-conquer algorithm [4] in *Triangle* performs 7.49 in-circle and 10.98 CCW tests per insertion. These statistics clearly indicates that point location is the performance bottleneck of incremental insertion algorithms.

In the following sections, we will present two strategies for efficient point location which in combination can improve the empirical performance of incremental insertion algorithms to a level comparable to or better than that of divide-and-conquer algorithms at the cost of no extra storage.

2. Order out of chaos

In a dataset possessing some sort of natural order (e.g., vertex sequence in digitized cartographic contour lines), consecutive points tend to be close to each other geometrically and using the last inserted point as the search-hint often produces good performance. A dataset without such an order may be processed in some way to impose upon it an order with good geometric proximity between consecutive points to facilitate point location. Some space-filling curves such as Hilbert and column-prime (or row-prime) curves that minimize the distance between any two consecutive points are obvious candidates for this purpose. A column-prime order (CPO) on a (not necessarily random) point set may easily be constructed in $O(n \log n)$ time using comparison-based sorting:

Algorithm 1. 2D-CPO.

- *Input*: a dataset of n points in a rectangular domain with width W and height H ($W \geqslant H$; otherwise row-prime should be used), stored in a random access-able array **P**.
- Sort **P** by x-ascending order; divide **P** into $m = \lceil c \times \lceil \sqrt{n \times W/H} \rceil \rceil$ slots (c is 0.5 by default); sort slot $j = 1,\ m$ in y-ascending order for odd j or y-descending otherwise.
- *Output*: the n points are in column-prime order now.

Note that the idea of manipulating insertion order in incremental Delaunay triangulation is really not new. For example, in [14], points are allocated into bins organized in row-prime order to achieve an $O(n^{5/4})/O(n^2)$ average/worst-case time. Although CPO maybe provide better proximity for location, it is in fact an $O(n^{3/2})/O(n^2)$ average/worst-case scheme so we do not suggest that it should be used standalone.

3. Hierarchy of orders

Instead of using a single level of insertion order, both the quaternary tree bucketing scheme [10] and the recent BRIO scheme [1] (proposed for improving locality of reference in geometric computation on a large dataset) possess a hierarchical sampling structure and points are inserted in "rounds" with each round corresponding to a level in the hierarchy.

With the presence of a sampling hierarchy, points inserted in earlier rounds are likely to divide the domain into smaller regions and act as "stoppers" to restrict the "scope" of triangles (i.e., number of uninserted points in the circumscribing circle of a triangle) [7] created by point insertion in later rounds. Consequently, the $O(n^2)$ worst-case scenario becomes highly improbable regardless of insertion order at each round. Following the same intuition, we propose the scheme of hierarchical column-prime order (HCPO) as follows:

Algorithm 2. 2D-HCPO.

- *Input*: a dataset of n points stored in an randomly access-able array **P**, sampling rate $sr \in (0, 1)$ and minimum sampling size *minsz*.
- *Step* 1: Randomly shuffle **P** if points in the dataset are not random.
- *Step* 2: Divide **P** into two parts $\mathbf{P}_{\text{hd}}$ and $\mathbf{P}_{\text{tl}}$ so that $\mathbf{P}_{\text{hd}}$ contains the first $\lceil \mathbf{N}^* sr \rceil$ points in **P** and $\mathbf{P}_{\text{tl}}$ contains the remaining points; build CPO on $\mathbf{P}_{\text{tl}}$.
- *Step* 3: If the number of points in $\mathbf{P}_{\text{hd}}$ is not less than $\lceil minsz/sr \rceil$, $\mathbf{P} \leftarrow \mathbf{P}_{\text{hd}}$, $\mathbf{N} \leftarrow \lceil \mathbf{N}^* sr \rceil$ and go to step 2; otherwise, build CPO on $\mathbf{P}_{\text{hd}}$ and return.
- *Output*: the n points are in HCPO.

It may be more desirable to use a stricter random sampling scheme in step 2 but the shuffle-once approach is practically sufficient. In addition, the sorting direction in CPO building may be altered so that the last point in one level of CPO is close to the first point in the next level of CPO.

Algorithm 2 in fact describes a more generic process (*hierarchical space-filling insertion order*, or HSFIO) of which HCPO is an instance. We may get different hierarchical orders if we substitute other space-filling curves (e.g., Hilbert) for column-prime. Under such a framework, the quaternary tree bucketing scheme is effectively a Z-curve based HSFIO with sampling rates varying at different levels. BRIO is a hierarchy of Z-ordered bins with a fixed sampling rate of 0.5. If bin size is set to 1, BRIO also becomes a Z-curve based HSFIO. Note that Z-curve is not optimal for location. For example, for a uniform dataset of 10^5 points at a sampling rate of 0.5, the CCW count for HCPO is 6.786 where it is 8.579 for a hierarchical Z-order (i.e., a BRIO with bin size of 1). In addition, the use of bin in BRIO (in its current form) makes it rather difficult to avoid using auxiliary memory ($\Omega(c^n),\ c > 1$) or to adopt direct key computation methods (e.g., bit-interleaving, which may be used for building Z-curve based HSFIO using $O(n)$ auxiliary memory).

4. Experimental running time of HCPO

We have tested 2D-HCPO, 2D-BRIO, *Triangle*'s implementations of Dwyer's $O(n \log\log n)$ and the original $O(n \log n)$ divide-and-conquer (e.g., [6], using vertical cut only, denoted as D&C) algorithms on three types of dataset: random points uniformly distributed in a square, random points on a parabola (x is uniform on $[0, 1)$ and $y = x^2$), and vertices of cartographical contour lines (UK Ordnance Survey Landform data, combinations of 1 to 6 adjacent tiles). Two triangulators are used: *Triangle* [12] and *MGLIB-2*. *MGLIB-2* is a fully dynamic C++ CDT package with a triangulation-walking method similar to the remembering stochastic walk in [3]. It is less scalable and slightly slower than *Triangle* and we adopt it mainly for the easy collection of various statistics. The 2D-BRIO is built with a temporarily allocated quadtree and a bin size of 32 (64 for the largest dataset) that produces best overall performance for BRIO in our systems.

Table 1 shows the running times of HCPO and BRIO using *MGLIB-2* and the two divide-and-conquer methods in *Triangle* on uniform datasets. The test platform is an AMD-XP2600$^+$ ($\sim$ 2.0 GHz) + Windows XP system with 1408 MB 333 MHz DDR RAM sufficient for triangulating approximately 10^7 points in-memory in practice. Generally speaking, HCPO is the fastest even when the slower *MGLIB-2* is used. BRIO is only slightly slower for smaller datasets. However, for the largest dataset (2×10^7) where impact of secondary memory access is significant, BRIO becomes the slowest. This is rather ironic as the main objective of BRIO has been to improve data locality. Another surprising result is that Dwyer's algorithm is the slowest for datasets larger than 10^6 points and the proportion of its pre-processing time is extraordinarily high.

Table 1
Pre-process, construction and total time on uniform dataset (in seconds)

Dataset		10^4	2×10^4	5×10^4	10^5	2×10^5	5×10^5	10^6	2×10^6	5×10^6	10^7	2×10^7
HCPO + MGLIB	Prep	0.003	0.003	0.025	0.062	0.153	0.44	1.172	3.006	10.72	27.1	68.84
($sr = 0.1$)	Ins	0.019	0.041	0.09	0.188	0.384	1.013	2.103	4.372	11.5	23.45	116.2
	Total	0.022	0.044	0.115	0.25	0.537	1.453	3.275	7.378	22.22	50.56	185
BRIO + MGLIB	Prep	0	0.003	0.028	0.075	0.162	0.587	1.372	3.325	10.33	23.32	54.75
	Ins	0.022	0.044	0.113	0.231	0.481	1.197	2.462	5.059	12.71	26.57	509
	Total	0.022	0.047	0.141	0.306	0.643	1.784	3.834	8.484	23.04	49.89	563.8
Dwyer in *Triangle*	Prep	0.003	0.028	0.066	0.188	0.438	1.256	3.638	10.52	38.5	102.5	245.8
	Cons	0.013	0.016	0.044	0.081	0.166	0.438	0.919	1.931	5.01	10.74	87.08
	Total	0.016	0.044	0.11	0.269	0.604	1.694	4.557	12.45	43.51	113.2	332.9
D&C in *Triangle*	Prep	0.003	0.006	0.034	0.066	0.15	0.431	1.078	2.934	9.487	23.34	56.19
	Cons	0.016	0.037	0.1	0.215	0.506	1.257	2.678	5.719	16.70	33	119.1
	Total	0.019	0.043	0.134	0.281	0.656	1.688	3.756	8.653	26.19	56.34	175.3

Table 2
Total running time on point-on-parabola datasets (in seconds)

Dataset	10^5	2×10^5	5×10^5	10^6	2×10^6	5×10^6	10^7	2×10^7
HCPO ($sr = 0.1$)	0.234	0.656	2.36	5.734	13.03	36.77	81.55	250.7
BRIO	0.640	1.563	4.89	10.73	23.55	66.36	143.1	trash
D&C	0.406	0.922	2.781	6.985	17.52	56.02	140.3	478.6

Table 3
Total running time on real datasets (seconds per 10^4 points)

#Points	J&W	Natural order	BRIO + *MGLIB*	HCPO + *MGLIB*	D&C	Dwyer	HCPO + *Triangle*
22 011	0.3685	0.1499	0.1153	0.1075	0.08632	0.06815	0.06361
62 788	0.5280	0.1835	0.1141	0.1046	0.1005	0.07979	0.06864
205 754	0.7637	0.1387	0.1199	0.1066	0.1091	0.08569	0.07251
475 731	1.023	0.1562	0.1254	0.1126	0.1341	0.09303	0.07643
657 962	1.157	0.1625	0.1292	0.1152	0.1120	0.09695	0.07868

Table 2 shows the results of overall running time on parabola datasets (exact arithmetic is used and the same robust predicates [13] used in *Triangle* are adopted in *MGLIB-2*). Here HCPO has a clear advantage over other schemes. On the other hand, the performance of BRIO is worse comparing to that on uniform datasets. Results of Dwyer's (slower than D&C's) are omitted.

Table 3 are results (calibrated to overall run-time per 10^4 points) for real datasets on a slower system with mobile PIII-700 MHz CPU and 384 MB 133 MHz SDRAM. Results of the Jump-and-walk scheme (J&W), *natural order* based insertion (both using *MGLIB-2*) and HCPO with *Triangle*'s, incremental triangulator are also shown for comparison.

On all three series of datasets, the HCPO scheme produces performances comparable to or even better than that of divide-and-conquer algorithms. Furthermore, the result of HCPO in combination with *Triangle* suggests that if the same data structures and primitives are used, incremental insertion with HCPO will potentially be significant faster than divide-and-conquer algorithms, contrary to the widely accepted claim that divide-and-conquer algorithms are practically the most efficient methods for DT construction. To our knowledge the results presented here are among the first to demonstrate that an incremental insertion algorithm may practically outperform divide-and-conquer algorithms on both uniform and non-uniform distributions.

5. Empirical results on time complexity of HCPO

The theoretical analysis in [7] for triangulation update during random insertion cannot be applied to HCPO without modification as HCPO is not completely random. Nevertheless, our empirical results (Table 4) indicate that under the HCPO scheme over a large range of sampling rates, CCW test, edge flip and in-circle test counts are all effectively $O(n)$.

To fit a power function $c = An^{1+\varepsilon}$ (c is the count and n is dataset size) to the total number of the three counts (shown in Table 5), for edge flips and in-circle tests, HCPO ($sr = 0.1$ and 0.75) have epsilon factors at the same magnitude as that of the theoretical $O(n)$ random insertion scheme (J&W). On the other hand, CCW counts for HCPO is in fact sub-linear on uniform datasets and virtually $O(n)$ on parabola datasets. The results on the (relatively small) real datasets (not shown in Tables 4 and 5) display the same trend. For example, the results (in the form of (A, ε)) of HCPO ($sr = 0.1$) are Flip(3.5132, -0.0024), In-circle(10.03, -0.0017) and CCW(12.262, -0.003) where the corresponding results of random insertion are Flip(2.9689, 0.0001), In-circle(8.9354, 0.0001) and CCW(2.0294, 0.2959).

Table 6 shows the relations between HCPO sampling rates and edge flips, in-circle and CCW tests per insertion on a uniform random dataset of 10^7 points (similar results are acquired on datasets of different sizes). Although larger sampling rates may result in lower flip and in-circle test counts, in practice much

Table 4
CCW, flip and in-circle counts (per insertion)

Dataset			10^5	2×10^5	5×10^5	10^6	2×10^6	5×10^6	10^7	2×10^7
Uniform	HCPO	Flip	3.245	3.241	3.251	3.251	3.252	3.255	3.256	3.260
	($sr = 0.1$)	In-cir	9.490	9.481	9.502	9.502	9.504	9.510	9.513	9.521
		CCW	5.563	5.559	5.553	5.549	5.546	5.543	5.538	5.528
	BRIO	Flip	2.997	3.000	3.000	3.002	3.003	3.004	3.003	3.004
		In-cir	8.993	9.000	9.001	9.005	9.007	9.008	9.007	9.009
		CCW	11.62	12.32	11.85	12.12	11.85	11.41	11.88	11.41
	Random	Flip	2.993	2.995	2.998	2.998	2.999	2.999	2.999	–
	J&W	In-cir	8.985	8.990	8.995	8.996	8.997	8.998	8.998	–
		CCW	57.15	71.06	95.02	118.5	148.2	199.2	249.4	–
Parabola	HCPO	Flip	1.028	1.028	1.031	1.032	1.035	1.052	1.091	1.178
	($sr = 0.1$)	In-cir	5.056	5.055	5.061	5.063	5.071	5.105	5.182	5.356
		CCW	4.452	4.463	4.498	4.498	4.496	4.491	4.474	4.448
	BRIO	Flip	1.205	1.102	1.189	1.095	1.186	1.304	1.309	–
		In-cir	5.409	5.204	5.378	5.190	5.372	5.607	5.618	–
		CCW	31.67	32.30	33.99	34.66	35.81	37.35	37.22	–
	Random	Flip	1.302	1.151	1.295	1.149	1.284	1.490	–	–
	J&W	In-cir	5.604	5.302	5.591	5.296	5.567	5.979	–	–
		CCW	945.1	1497	2779	4423	7035	12 870	–	–

Table 5
CCW, flip and in-circle counts ($c = An^{1+\varepsilon}$)

Dataset	Uniform						Parabola					
	Flip		In-circle		CCW		Flip		In-circle		CCW	
	A	ε	A	ε	A	ε	A	ε	A	ε	A	ε
Random J&W	2.9799	0.0004	8.9579	0.0003	1.4287	0.3201	0.809	0.0338	4.4786	0.016	0.4286	0.6687
BRIO (bsz = 32)	2.9844	0.0004	8.9661	0.0003	13.042	−0.007	0.8182	0.0275	4.5481	0.0124	20.405	0.0383
HCPO (sr: 0.75)	2.9865	0.0004	8.971	0.0003	8.9053	−0.0029	0.8061	0.0323	4.4948	0.0149	18.431	0.0009
HCPO (sr: 0.1)	3.2084	0.0009	9.4144	0.0007	5.6334	−0.0011	0.7937	0.0203	4.5333	0.0085	4.4746	0.0

Table 6
CCW, flip and in-circle counts and sampling rate on 10^7 uniform dataset

sr	0.005	0.01	0.05	0.1	0.125	0.2	0.25	0.333	0.5	0.667	0.75	0.8	0.9	0.95	0.99
Flip	3.798	3.663	3.375	3.256	3.219	3.142	3.108	3.069	3.025	3.007	3.003	3.001	2.999	2.999	2.999
In-cir	10.60	10.33	9.749	9.512	9.438	9.285	9.217	9.138	9.051	9.014	9.006	9.002	8.999	8.998	8.998
CCW	5.786	5.64	5.479	5.538	5.587	5.758	5.892	6.133	6.749	7.715	8.511	9.206	11.91	15.70	31.66

smaller (but not too small) sampling rate will strike a better balance between the costs of order-building, edge flips, in-circle and CCW tests to produce better overall performance. In our systems, a rate of about 0.1 gives the best result. In addition, empirical results also suggest that as a side-product, smaller sampling rates provide better data locality when handling large datasets.

An interesting result regarding sampling rate is that for a sampling rate greater than $2/3$, there may exist sampling hierarchies where the 3 edge flip and 9 in-circle test bounds may be retained no matter what insertion order is applied at each round of insertion. On the other hand, for a sampling rate not greater than $2/3$, we may always be able to find an insertion order to make the above bounds un-retainable, no mat-

ter how we create the sampling hierarchy. This may be derived by a simple backward analysis.

The average degree of a vertex (i.e., number of triangles/edges coincident to it) in a triangulation is 6. For two points p_1 and p_2, if p_2 is to be deleted independent of the deletion of p_1, p_2 should *not* be a vertex of one of the 6 triangles coincident to p_1 (in other words, $p_1 - p_2$ should not be an edge). Given a DT of (sufficiently large) n points and assuming we can freely choose any $1 < m < n$ points from the point set, if we choose these m points in a way that any two points are not connected by an edge, they may be deleted in any order while the cost of deletion will be identical. To reverse this process, if we construct the DT of $n - m$ points first and then insert the remaining m points, the insertion order of this these m points is also insignificant and cost of insertion is identical (to that of random insertion order). The largest m for order independency is roughly $m_{\max} = \lfloor n_{\text{tri}}/6 \rfloor$ where $n_{\text{tri}} = 2n - h - 2$ is the total number of triangles in the triangulation. Here $h = \mathrm{O}(\ln(n))$ (due to A. Rényi and R. Sulanke, quoted as Theorem 4.1 in [11]) is the number of points on the convex hull of the point set. Consequently, $m_{\max} = (2n - h - 2)/6$ and the sampling rate bound $sr = (n - m_{\max})/n \to 2/3$.

As for order-building time, it is obvious that HCPO may be built in $\mathrm{O}(n \log n)$ time. For datasets with uniform or quasi-uniform distribution on at least one dimension, slots in Algorithm 1 may also be created on this dimension not by sorting adaptive to data but in a hashing or bin-sort style to improve order-building time by a constant factor, which could be significant for larger datasets. This approach however requires auxiliary memory and is less adaptive to non-uniform distributions. Furthermore, better linearity in order-building may be achieved by adopting radix sorting although actual performance gain may not be observed except for very large datasets.

References

[1] N. Amenta, S. Choi, G. Rote, Incremental constructions con BRIO, in: Proc. ACM Symp. on Comput. Geometry, 2003, pp. 211–219.

[2] M. de Berg, M. van Kreveld, M. Overmars, O. Schwarzkopf, Computational Geometry: Algorithms and Applications, Springer-Verlag, Berlin, ISBN 3-540-61270-X, 1997.

[3] O. Devillers, S. Pion, M. Teillaud, Walking in a triangulation, in: Proc. ACM Symp. on Comput. Geometry, 2001, pp. 106–114.

[4] R.A. Dwyer, A faster divide-and-conquer algorithm for constructing Delaunay triangulations, Algorithmica 2 (1987) 137–151.

[5] P.J. Green, R. Sibson, Computing Dirichlet tessellations in the plane, Comput. J. 21 (1978) 168–173.

[6] L.J. Guibas, J. Stolfi, Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams, ACM Trans. on Graphics 4 (2) (1985) 74–123.

[7] L.J. Guibas, D.E. Knuth, M. Sharir, Randomized incremental construction of Delaunay and Voronoi diagrams, Algorithmica 7 (4) (1992) 381–413.

[8] C.L. Lawson, Software for Cl surface interpolation, in: J.R. Rice (Ed.), Mathematical Software III, Academic Press, New York, 1977, pp. 161–194.

[9] E.P. Mücke, I. Saias, B. Zhu, Fast randomized point location without preprocessing in two- and three-dimensional Delaunay triangulations, Comput. Geometry 12 (1–2) (1999) 63–83.

[10] T. Ohya, M. Iri, K. Murota, A fast Voronoi-diagram algorithm with quaternary tree bucketing, Inform. Process. Lett. 18 (1984) 227–231.

[11] F.P. Preparata, M.I. Shamos, Computational Geometry: an Introduction, Springer-Verlag, Berlin, ISBN 0-387-96131-3, 1985.

[12] J. Shewchuk, Triangle: engineering a 2D quality mesh generator and Delaunay triangulator, in: Proc. WACG'96, 1996, pp. 203–222.

[13] J. Shewchuk, Adaptive precision floating-point arithmetic and fast robust geometric predicates, Discrete Comput. Geom. 18 (3) (1997) 305–368.

[14] S.W. Sloan, A fast algorithm for constructing Delaunay triangulations in the plane, Adv. Eng. Softw. 9 (1) (1987) 34–55.

[15] P. Su, R.L.S. Drysdale III, A comparison of sequential Delaunay triangulation algorithms, Comput. Geometry 7 (1997) 361–385.