Comp. Appl. Math. (2018) 37:641–674
https://doi.org/10.1007/s40314-016-0358-0

An evaluation of point-insertion sequences for incremental Delaunay tessellations

Sanderson L. Gonzaga de Oliveira[1] · **Jéssica Renata Nogueira**[2]

Received: 5 March 2016 / Revised: 29 May 2016 / Accepted: 4 June 2016 / Published online: 5 July 2016

Abstract Currently, incremental algorithms may be seen as the lowest-cost computational methods to generate Delaunay tessellations in several point distributions. In this work, eight point-insertion sequences in incremental algorithms for generating Delaunay tessellations are evaluated. More specifically, four point-insertion sequences in incremental algorithms for generating Delaunay tessellations are proposed: with orders given by the red–black tree with in-order and level-order traversals, spiral ordering, and H-indexing. These four incremental algorithms with such sequences are compared with four incremental algorithms with point-insertion orders given by the following sequences: the Hilbert and Lebesgue curves, cut-longest-edge kd-tree, and random order. Six 2-D and seven 3-D point distributions are tested, with sets ranging from 25,000 to 8,000,000 points. The results of computational and storage costs of these eight algorithms are analyzed. It follows that the incremental algorithm with a point-insertion sequence in the order given by the cut-longest-edge kd-tree shows the lowest computational and storage costs of the sequences tested.

Keywords Mesh generation · Delaunay tessellation · Incremental algorithms · Computer-aided design, engineering and manufacturing · Computational geometry and topology · Insertion sequences · Non-uniform point distributions

1 Introduction

Meshes are employed in a number of applications, particularly in finite element discretizations, and are key tools in scientific computing. In this field of study, common meshes are

✉ Sanderson L. Gonzaga de Oliveira
sanderson@dcc.ufla.br

Jéssica Renata Nogueira
jessica.nogueira@ifsuldeminas.edu.br

[1] Universidade Federal de Lavras, Lavras, Brazil

[2] Instituto Federal de Educação, Ciência e Tecnologia do Sul de Minas Gerais/Campus Passos, Passos, Brazil

Delaunay tessellations. For a d-dimensional point set, a Delaunay tessellation is described as a mesh in which the d-dimensional ball of each polytope lacks inner points. These meshes are popular mainly because they can be built rapidly and have very appealing geometric properties; for example, the Voronoi diagram, a dual mesh of the Delaunay tessellation, may capture proximity (Gonzaga de Oliveira et al. 2014). Furthermore, Delaunay tessellations are used to represent discrete places of a continuous space in a manner that permits the use of numerical methods to compute characteristics of that space (Edelsbrunner 2001). Delaunay tessellations and Voronoi diagrams have been employed in several applications in science and engineering, such as computer graphics, industrial design, medical applications, the modeling of composite and porous materials, deformable objects, molecules, and terrain, the tessellation of solid shapes, and video games (Gonzaga de Oliveira et al. 2014).

Currently, incremental algorithms may be seen as the lowest-cost computational methods to generate Delaunay tessellations in several point distributions. In particular, Liu et al. (2013) showed simulations in which an incremental algorithm with point-insertion order given by the cut-longest-edge kd-tree outperformed the previous possible state-of-the-art algorithm [an incremental algorithm with point-insertion order given by the Hilbert curve (Liu and Snoeyink 2005)] for this task in several 3-D point distributions.

It should be noticed that algorithms for Delaunay tessellations may have different performances in different point distributions. Since Liu et al. (2013) applied the computational geometry algorithms library (CGAL) in their implementation, our approach employs a specific implementation to verify if similar results are found. Moreover, a comparison of different point-insertion sequences for incremental Delaunay tessellation algorithms is conducted with various point distributions, from uniform cases to non-uniform cases. Computational cost and memory requirements are measured. To be more precise, in our approach, eight incremental algorithms were implemented using our own geometric methods. It is important to remark that the execution time under diverse implementations will differ. In our assessment, given the fact that we are re-implementing the Liu–Yan–Lo and Hilbert-curve incremental algorithms, it is possible to verify whether the same behavior of the computational costs and memory requirements is encountered. Then, we could investigate the influence of the CGAL framework in the results. Thereby, our approach focuses on implementation details of incremental algorithms using deterministic orders (without randomness) to generate Delaunay triangulations in seven 3-D point distributions [the same ones employed by Liu et al. (2013)] and also in six 2-D point distributions. To provide more specific detail, our domains are the unit cube and unit square, instead of integer domains, such as employed by Schrijvers et al. (2013) and other authors. Thus, eight point-insertion sequences are evaluated in the 2-D and 3-D unit intervals:

1. the Hilbert space-filling curve (Liu and Snoeyink 2005);
2. the Lebesgue curve, also called Z-order and Morton order (Bader 2012);
3. the H-indexing (Niedermeier et al. 2002);
4. the spiral ordering;
5. the random order;
6. the cut-longest-edge kd-tree (Liu et al. 2013);
7. the in-order and level-order red–black tree traversals.

To our knowledge, this paper is the first (published) instance employing the H-indexing scheme (Niedermeier et al. 2002), the spiral ordering and the in-order and level-order red–black tree traversals being used in incremental algorithms for Delaunay tessellations. In particular, the Peano and Sierpiński curves were not evaluated in this study because Schrijvers

et al. (2013) found that the performances shown by these curves are inferior to the performance shown by the Hilbert curve in incremental algorithms.

Additionally, for seven incremental algorithms – that is, except the incremental algorithm using the cut-longest-edge kd-tree – three approaches are evaluated that seek the tetrahedron that contains the point inserted most recently into the tessellation. The Liu–Yan–Lo incremental algorithm (Liu et al. 2013) was implemented for the cut-longest-edge kd-tree.

Section 2 presents a brief review on insertion schemes in algorithms for generating Delaunay tessellations. Section 3 describes the incremental algorithms, with orders given by the eight point-insertion sequences implemented. Section 4 describes the tests. Section 5 presents and analyzes the results. Finally, Sect. 6 addresses the conclusions.

2 Review on insertion schemes

Delaunay and Voronoi tessellations have been broadly investigated, and many methods have been used to build these structures. A detailed description about the development of algorithms for generating these meshes is provided in Gonzaga de Oliveira et al. (2014). In particular, with the biased randomized insertion order (BRIO) algorithm, Amenta et al. (2003), considered the order in which the points are inserted into the tessellation. In this algorithm, the spatial location of points is deemed to cause a larger number of cache hits. In particular, some results with respect to cache misses are presented below (see Sect. 5).

Carey (1997) presented a number of techniques to generate the Delaunay tessellation, such as the incremental approach. This author explained that the Hilbert curve is a good choice of space-filling curve pattern in terms of minimizing the locality of objects within the multidimensional space. The Hilbert curve is easily constructed by means of repeated reflections and rotations. Since the Hilbert curve leads to a linear sequence, locality is preserved in this approach.

Based on the idea of a larger number of cache hits, Liu and Snoeyink (2005) proposed an incremental algorithm for generating Delaunay tessellations in which points are inserted into the tessellation in the order given by the Hilbert curve. Liu and Snoeyink (2005) and Schrijvers et al. (2013) give very detailed information about the various effects of choosing the order, choosing the right amount of randomness, etc. Moreover, Liu and Snoeyink (2005), Zhou and Jones (2005), Buchin (2005, 2007), and Boissonnat et al. (2009) have shown algorithms that use some combination of randomness and deterministic orders.

On the other hand, one can consider that it is more important that the various parts of the data structure and algorithms used work well together. This is also the key improvement of the Liu–Snoeyink algorithm (Liu and Snoeyink 2005) over the BRIO scheme (Amenta et al. 2003). This in turn means that it is quite important to also consider carefully the specifics of the Delaunay triangulation implementation used, if it is not a standard one (such as the one used here), because it may have a considerably higher effect on the performance than the specific order [see Liu and Snoeyink (2005); Zhou and Jones (2005); Buchin (2005, 2007); Boissonnat et al. (2009)].

Schrijvers et al. (2013) provided computational experiments showing that an algorithm with point-insertion order given by the Hilbert order (Liu and Snoeyink 2005) superseded several algorithms (including algorithms with some kind of randomness) in six 2-D data distributions (which three of them are also considered here). Despite several other proposals and simulations in the following years, probably until 2013, the Liu–Snoeyink algorithm (Liu and Snoeyink 2005) was the possible state-of-the-art algorithm for generating Delaunay

tessellations. In 2013, Liu et al. (2013) proposed an incremental algorithm that outperformed the incremental algorithm using the Hilbert curve (Liu and Snoeyink 2005) to generate Delaunay tessellations in seven 3-D point distributions.

3 Description of point-insertion sequences in the incremental algorithms implemented

In incremental algorithms for Delaunay tessellations, the order in which the points are inserted into the tessellation influences the computational cost of the algorithm. It affects the time for both point location and structure update, and consequently the overall execution time of the incremental DT algorithm (Liu et al. 2013). This influence occurs because the computational costs of these algorithms depend on the number of tetrahedrons checked in the circumsphere test. In this test, a procedure verifies whether a point is inside a tetrahedron using a relatively expensive circumsphere test. Since a low-cost approach must check a small number of tetrahedrons, we evaluated three approaches (termed *hist_tet*, *lst_tet*, and *tet_inc*) to seek the tetrahedron that contains the most recently inserted point into the tessellation. We describe these approaches in Sect. 4. In addition, the computational costs of these algorithms also depend on updates in the tessellation, i.e., the computational cost for updating a Delaunay tessellation is proportional to the number of created and destroyed tetrahedrons.

Algorithm 1 shows a pseudocode related to the algorithms implemented in this work. Entry points of Algorithm 1 can be inserted into the tessellation through different ways. Consequently, entry points are ordered in line 2 of Algorithm 1.

Input: set of points $P = \{p_1, p_2, \cdots, p_n\}$.
Output: Delaunay tessellation DT(P).
1 **begin**
2 rearrange entry points $p_1, p_2, \cdots, p_n$;
3 build a super triangle (tetrahedron) *st*, which contains all the points of P;
4 insert *st* into DT(P);
5 $i \leftarrow 1$;
6 **while** *($i \leq |P|$)* **do**
`// list_t is a list of triangles (tetrahedrons) whose`
7 *list_t* $\leftarrow \varnothing$; `// circumcircle (circumsphere) contains` p_i
`// Section 3 presents the strategies used in this work to`
`// find a triangle (tetrahedron) whose circumcircle`
`// (circumsphere) contains` p_i
8 locate a triangle (tetrahedron) t whose circumcircle (circumsphere) contains p_i;
9 insert t into *list_t*;
10 **foreach** *(triangle (tetrahedron)* t_a *adjacent to triangles (tetrahedrons) stored in list_t)* **do**
11 **if** *(the circumcircle (circumsphere) of* t_a *contains* p_i*)* **then**
12 insert t_a into *list_t*;
13 remove from DT(P) tetrahedrons stored in *list_t*;
14 insert, into DT(P), those tetrahedrons formed by connection of p_i with elements of the cavity border formed after removal of tetrahedrons stored in *list_t*;
15 $i \leftarrow i$+1;
16 remove from DT(P) those triangles (tetrahedrons) that contain vertices of the super triangle (tetrahedron);
17 **return** DT(P);

Algorithm 1: an incremental algorithm for generating Delaunay tessellations.

The following is a brief description of point-insertion sequences in the incremental algorithms implemented.

- Cut-longest-edge kd-tree: This is used in the incremental algorithm that is the possible state-of-the-art algorithm for generating Delaunay tessellations for seven 3-D point distributions (Liu et al. 2013). In this order, points are inserted into the tessellation according to a level-order traversal of the cut-longest-edge kd-tree (Liu et al. 2013).
- Hilbert and Lebesgue curves: aiming to order the set of points using the 2-D or 3-D Hilbert and Lebesgue curves, the domain is divided into 2^{2k} or 2^{3k} subdivisions, respectively, where k is a number large enough so that each subdivision contains a small number of points. When necessary, these subdivisions are divided again until each subdivision contains only one point. The Hilbert curve was implemented here because it was used in the algorithm that was the possible state-of-the-art algorithm for the generation of Delaunay tessellations until 2013. Additionally, the Lebesgue curve was chosen to be implemented because Schamberger and Wierum (2005) noted that for data arranged in a spiral distribution and for data distributed uniformly, the Lebesgue curve showed better results than the $\beta\Omega$-indexing, Hilbert, and Sierpiński curves. The Lebesgue curve required a smaller quantity of domain partitions than the other three orders. Although the 3-D Lebesgue curve was also tested by Snoeyink and Liu (2005), these tests were not in the seven 3-D point distributions tested here and were not in the unit interval.
- Spiral ordering: The domain subdivisions are traversed from left to right, bottom to top, from right to left, and from top to bottom. This path is performed again, more internally until all domain subdivisions are covered. By using the 3-D spiral ordering, initially, the points are ordered by their z coordinates; soon after, points with the same z coordinates are ordered using the spiral ordering. If two or more points are in the same subdivision, the indexing scheme is applied again in this subdivision. Duff and Meurant (1989) studied 17 ordering methods and their effects when ordering unknowns of linear systems arising from finite difference discretizations on the convergence of the Preconditioned Conjugate Gradient Method. These authors concluded, through testing, that the spiral ordering showed good results for all sets tested.
- H-indexing: the H-indexing scheme is based on a 2-D variant of the Sierpiński curve. By using the 3-D H-indexing, initially, the points are ordered by their z coordinates; soon after, points with the same z coordinates are ordered using the H-indexing. It should be noticed that this ordering step is not necessary in 2-D point distributions. If two or more points are in the same subdivision, the indexing scheme is applied again in this subdivision. Niedermeier et al. (2002) proved that this scheme has better locality than the Hilbert indexes.
- Red–black tree with in-order and level-order traversals: Duff and Meurant (1989) showed that the sequence given by the red–black tree was very competitive with other sequences tested.
- Random order: this point-insertion sequence was evaluated here for being the most basic scheme for inserting points into a tessellation.

4 Description of the tests

The eight incremental algorithms were implemented in the C++ programming language. Specifically, the g++ 4.6.3-1 compiler was used, with the optimization flag –O3. A placement test was used to maintain tetrahedron points in a counterclockwise direction. This step was necessary for the circumsphere test to return consistent results. For all 2-D and 3-D point-

insertion sequences, six and seven point distributions were used, which are shown in Figs. 1 and 2, respectively.

Except for the incremental algorithm with a point-insertion sequence in the order given by the cut-longest-edge kd-tree, three approaches to seek a point p to be inserted into the tessellation were tested for the other seven incremental algorithms implemented. These three approaches are described below.

- *hist_tri (hist_tet)*: triangles (tetrahedrons) were verified in the reverse order in which they were inserted into the tessellation.
- *lst_tri (lst_tet)*: a breadth-first search beginning in the last triangle (tetrahedron) created was performed. To provide more specific detail, a procedure checks if the last polytope created (t) contains the point p. If p is not contained in t, its neighbors are checked. This procedure traverses the mesh in breadth-first order to find the polytope that contains p.
- *tri_inc (tet_inc)*: k incident triangles (tetrahedrons) to the last point p inserted into the tessellation were checked. If a circumcircle (circumsphere) of a triangle (tetrahedron) containing the point p was not found, adjacent triangles (tetrahedrons) to the k triangles (tetrahedrons) were also sought, from the least to the most recently verified triangle (tetrahedron).

These searches terminated when a circumcircle (circumsphere) of a triangle (tetrahedron) containing p was found. For the incremental algorithm with a point-insertion sequence given by the cut-longest-edge kd-tree, the search of the triangle (tetrahedron) containing a point p was performed as described by Liu et al. (2013).

The workstations used in the execution of the simulations contained the following (Intel® Core™; Santa Clara, CA, USA):

- (M1) i3 CPU 550 3.20 GHz with 4 MB cache memory and 16 GB of main memory DDR3 1333 MHz;
- (M2) i3-2120 CPU 3.30 GHz with 3 MB of cache memory and 8 GB of main memory DDR3 1333 MHz;
- (M3) i3-2100 CPU 3.10 GHz with 3 MB of cache memory and 8 GB of main memory DDR3 1333 MHz;
- (M4) i7-4790 K CPU 4,00 GHz with 8 MB of cache memory and 12 GB of main memory DDR3 1.6 GHz.

The Ubuntu 14-04 LTS 64-bits operating system was used in the four machines, with kernel 3.13.0-39 generic on the M1 machine, kernel 3.13.0-43 generic on the M2 and M3

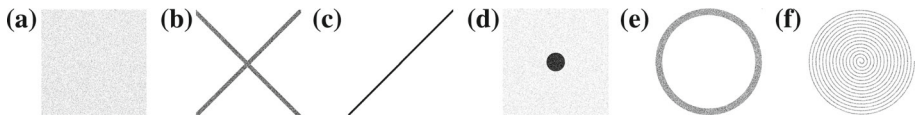

Fig. 1 Six point distributions on the unit square: **a** random; **b** cross; **c** line; **d** cluster; **e** circle; and **f** spiral distributions. Each set was composed of 50,000 points

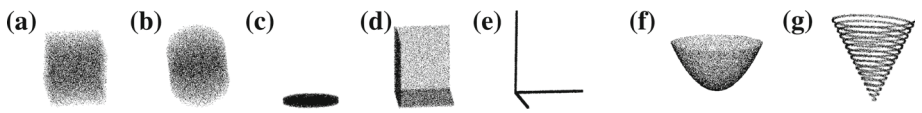

Fig. 2 Seven point distributions on the 3-D unit interval: **a** random points in a unit cube; **b** points on a cylinder; **c** points around a disk; **d** points around three planes; **e** points along three axes; **f** points around a paraboloid; and **g** points around a spiral. Each set was composed of 25,000 points

machines, and kernel 3.19.0-31 generic on the M4 machine. Five executions were carried out for each instance.

It was not our intention that the results of the eight incremental algorithms implemented outperform results from highly enhanced versions that employ corresponding point-insertion sequences in incremental algorithms. Our purpose was to obtain reasonably efficient implementations of these algorithms to allow an appropriate comparison of their results. To provide more specific detail, our goal is to evaluate whether the CPU times of eight point-insertion sequences in these algorithms [such as the one proposed by Liu et al. (2013)] may be independent of a framework. For updated times in Delaunay mesh generation, for instance, we rely on Lo (2015). Lo (2015) reported low execution times when generating a 2-D Delaunay triangulation (1 million points). He argues that the speed of a triangulation is very sensitive to the order of how the points are processed, from a quasi-linear behavior (when applied to sorted data) to a quadratic behavior (when applied to unsorted data).

For the construction of the mesh, a data structure is used to store the point coordinates. More specifically, in this data structure, the coordinates and a unique number for each point are stored. In addition, for each tetrahedron, four indices are stored and each index refers to the number of a point. This number is employed when a tetrahedron is created. Additionally, an adjacency matrix was used.

The incremental algorithm with a point-insertion sequence given by the Hilbert and Lebesgue curves, and the H-indexing order use a C++ vector data structure to store points with indices repeated when applied to the spiral distribution. In addition, the incremental algorithm with a point-insertion sequence given by the H-indexing order sorts the points. A matrix was used to store the resulting order.

For the construction of the cut-longest-edge kd-tree, an array stores the point coordinates. In addition, a data structure keeps the largest difference between point coordinates. This data structure also stores the largest, smaller and median point coordinates in such interval. For each node inserted into the cut-longest-edge kd-tree, indices to child nodes are stored. Additionally, this data structure maintains a reference to the axis where the subdivision was carried out.

5 Results

This section presents results of computational costs and memory requirements in six 2-D and seven 3-D point distributions when using eight point-insertion sequences in incremental algorithms for generating Delaunay tessellations. In addition, three approaches were used to seek the circumcircle (circumsphere) of the triangle (tetrahedron) that contained the point most recently inserted into the tessellation, in instances up to 1,000,000 in six 2-D and 8,000,000 points in seven 3-D point distributions.

Executions that computed for more than 10 and 30 min in 2-D and 3-D point distributions to return a Delaunay tessellation were aborted, respectively. These executions are indicated as "-" in the following tables. Numbers in bold face are the best results.

Table 1 (Table 2, with respect to 3-D Delaunay tessellations) shows the results of the computational costs and memory requirements when using the incremental algorithms with point-insertion sequences given by the orders of Hilbert and Lebesgue curves, H-indexing, spiral ordering, orders given by the red–black tree with in-order and level-order traversals, and random insertion points, in six 2-D (seven 3-D) point distributions. Similarly, Fig. 3 (Fig. 4 with respect to 3-D Delaunay tessellations) illustrates the results of the computational

Table 1 Results of computational costs (in s) and memory requirements (in MB) of incremental algorithms with point-insertion sequences given by seven sequences, and with three approaches to seek the triangle that contains the most recently inserted point in six 2-D point distributions

Distribution	Size of dataset	Insertion order	Results	*hist_tri*	*lst_tri*	*tri_inc*
Random	1,000,000	Hilbert	Time	**5.88**	6.19	5.89
			Memory	336.61	277.34	283.32
		Lebesgue	Time	**7.83**	8.41	8.76
			Memory	314.21	302.40	307.73
		H-indexing	Time	6.47	6.40	**6.35**
			Memory	1342.37	1279.55	1290.69
		Spiral	Time	9.48	8.17	**7.78**
			Memory	292.46	262.39	273.41
		Red–black tree with in-order	Time	**14.97**	366.54	379.11
			Memory	294.34	231.82	240.67
	100,000	Red–black tree with level-order	Time	**159.70**	329.38	378.34
			Memory	29.33	29.69	30.59
		Random	Time	**342.51**	–	–
			Memory	20.02	–	–
Cross	1,000,000	Hilbert	Time	68.83	**16.25**	16.48
			Memory	418.12	369.79	368.19
		Lebesgue	Time	**19.84**	19.86	21.04
			Memory	405.66	383.75	385.78
		H-indexing	Time	94.68	**12.88**	12.92
			Memory	1336.90	1281.23	1292.20
		Spiral	Time	8.77	9.08	**8.38**
			Memory	283.85	218.75	229.39
		Red–black tree with in-order	Time	**62.84**	90.41	110.40
			Memory	344.69	232.93	245.39
	250,000	Red–black tree with level-order	Time	–	**479.95**	510.58
			Memory	–	73.14	72.23
	100,000	Random	Time	**335.76**	–	–
			Memory	20.89	–	–
Line	1,000,000	Hilbert	Time	124.40	**7.56**	8.12
			Memory	580.27	566.98	557.87
		Lebesgue	Time	**8.63**	9.40	9.83
			Memory	592.72	567.25	564.89
		H-indexing	Time	**12.84**	12.90	13.44
			Memory	1342.18	1279.43	1291.42
		Spiral	Time	8.54	**7.96**	8.47
			Memory	264.75	198.92	213.50

Table 1 continued

Distribution	Size of dataset	Insertion order	Results	*hist_tri*	*lst_tri*	*tri_inc*
		Red–black tree with in-order	Time	124.05	**122.61**	135.30
			Memory	510.22	227.52	240.10
		Red–black tree with level-order	Time	–	**355.43**	361.64
			Memory	–	269.91	275.80
	100,000	Random	Time	**286.26**	–	–
			Memory	19.82	–	–
Cluster	1,000,000	Hilbert	Time	6.30	5.78	**5.98**
			Memory	388.50	326.79	333.56
		Lebesgue	Time	**7.55**	7.78	8.22
			Memory	364.75	355.62	364.73
		H-indexing	Time	439.26	**5.91**	6.13
			Memory	1326.00	1275.01	1285.18
		Spiral	Time	384.21	**7.13**	7.31
			Memory	271.74	268.70	269.82
		Red–black tree	Time	**13.95**	116.81	119.85
		with in-order	Memory	286.21	229.59	240.68
	100,000	Red–black tree with level-order	Time	**105.06**	197.49	208.72
			Memory	28.41	30.03	29.00
		Random	Time	**88.80**	316.44	326.62
			Memory	19.04	22.91	21.03
Circle	1,000,000	Hilbert	Time	14.01	**5.82**	6.07
			Memory	390.97	336.87	338.86
		Lebesgue	Time	7.97	**7.95**	8.38
			Memory	369.47	358.16	361.76
		H-indexing	Time	6.72	**5.96**	5.98
			Memory	1335.47	1275.74	1288.02
		Spiral	Time p	**8.15**	8.31	8.56
			Memory	286.72	231.48	244.06
		Red–black tree with in-order	Time	**12.70**	50.80	49.19
			Memory	299.14	228.70	239.71
	100,000	Red–black tree with level-order	Time	**108.27**	295.20	311.67
			Memory	29.03	30.34	30.26
		Random	Time	**309.53**	-	-
			Memory	20.73	-	–
Spiral	1,000,000	Hilbert	Time	6.29	**6.08**	6.26
			Memory	386.21	331.50	335.09
		Lebesgue	Time	10.06	**7.54**	7.88
			Memory	368.29	353.91	356.65

Table 1 continued

Distribution	Size of dataset	Insertion order	Results	*hist_tri*	*lst_tri*	*tri_inc*
		H-indexing	Time	6.67	6.24	**6.21**
			Memory	1334.38	1277.95	1287.79
		Spiral	Time	**9.35**	9.77	10.13
			Memory	265.27	236.38	246.17
		Red–black tree with in-order	Time	**11.74**	111.58	117.47
			Memory	289.21	231.34	242.44
	100,000	Red–black tree with level-order	Time	**118.74**	292.64	–
			Memory	27.96	30.48	–
		Random	Time	**305.78**	–	–
			Memory	20.69	–	–

costs when using the incremental algorithms with a point-insertion sequence given by the orders of Hilbert and Lebesgue curves, H-indexing, spiral ordering, and order given by the red–black tree with level-order traversal, in six 2-D (seven 3-D) point distributions (instances composed of 1,000,000 points). In general, these results corroborate the results obtained in smaller instances (25,000, 50,000, 100,000, 250,000, 500,000, and 750,000 points).

Table 3 (Table 4 with respect to 3-D Delaunay tessellations) resumes the strategies that presented the lowest computational cost in each distribution. These simplex-search (triangle in 2-D and tetrahedron in 3-D Delaunay tessellations) strategies within the point-insertion sequence in the corresponding incremental algorithm were selected to be compared with the Liu–Yan–Lo incremental algorithm. The incremental algorithms with point-insertion sequences given by the random order and the red–black tree with level-order traversal were considered to be dominated by the other five incremental algorithms in all 2-D and 3-D point distributions. In particular, in accordance with the findings presented in the current literature (Liu et al. 2013), the incremental algorithm with random point-insertion sequence obtained a high computational cost, as shown in Tables 1, 2.

Table 5 and Figs. 5 and 6 (Table 6; Figs. 7, 8 with respect to 3-D Delaunay tessellations) show the average computational costs, largest standard deviations and coefficient of variations (in spite of the small number of executions for each algorithm in each instance), and memory requirements of the other six incremental algorithms with point-insertion sequences in six (seven) point distributions in the 2-D (3-D) unit interval: the cut-longest-edge kd-tree, the Hilbert curve, the Lebesgue curve, the H-indexing, the spiral ordering, and the red–black tree with in-order traversal.

One can observe that the incremental algorithms with a point-insertion sequence in the order given by the cut-longest-edge kd-tree and Hilbert curve dominated the other algorithms. Consequently, we performed tests with 8,000,000 points in seven 3-D point distribution only with these two algorithms (see Table 6). Moreover, Table 7 shows the results with respect to cache misses of our implementations for the incremental algorithms with a point-insertion sequence in the order given by the cut-longest-edge kd-tree and Hilbert curve. Cachegrind, a tool of the Valgrind tool suite (Nethercote and Seward 2007), was used as cache profiler (Cachegrind 2016). Additionally, Table 8 shows the results with respect to the number of circumsphere tests and the numbers of created and destroyed tetrahedrons

Table 2 Results of computational costs (in s) and memory requirements (in MB) of incremental algorithms with point-insertion sequences given by seven sequences, and with three approaches to seek the tetrahedron that contains the most recently inserted point in seven 3-D point distributions (random points, points on a cylinder, points around a disk, points around three planes, points along three axes, points around a paraboloid, and points around a spiral)

Distribution	Size of dataset	Insertion order	Results	*hist_tet*	*lst_tet*	*tet_inc*
Random	1,000,000	Hilbert	Time	138.68	104.00	**102.66**
			Memory	1057.08	899.23	898.85
		Lebesgue	Time	**131.30**	171.43	164.68
			Memory	958.80	988.12	988.51
		H-indexing	Time	**361.79**	698.77	371.86
			Memory	1198.89	861.53	861.77
		Spiral	Time	**363.96**	703.88	378.84
			Memory	1205.63	873.70	873.62
		Red–black tree with in-order	Time	**346.49**	745.24	364.31
			Memory	1128.02	789.93	787.75
	100,000	Red–black tree with level-order	Time	**382.14**	–	–
			Memory	98.75	–	–
		Random	Time	**1122.02**	–	–
			Memory	89.51	–	–
Cylinder	1,000,000	Hilbert	Time	225.95	124.31	**123.42**
			Memory	1050.73	903.78	899.47
		Lebesgue	Time	**159.84**	204.51	202.07
			Memory	961.05	985.31	987.83
		H-indexing	Time	462.57	827.80	**461.37**
			Memory	1195.77	863.92	861.23
		Spiral	Time	**454.67**	837.34	461.73
			Memory	1206.66	872.72	872.14
		Red–black tree with in-order	Time	**436.36**	938.14	457.37
			Memory	1131.80	787.91	786.12
	100,000	Red–black tree with level-order	Time	**423.40**	–	–
			Memory	98.41	–	–
		Random	Time	**1408.29**	–	–
			Memory	90.11	–	–
Disk	1,000,000	Hilbert	Time	248.71	106.36	**105.08**
			Memory	1094.93	959.77	958.81
		Lebesgue	Time	**131.83**	153.82	148.57
			Memory	1020.25	990.76	988.73
		H-indexing	Time	**473.02**	988.06	496.89
			Memory	1211.93	859.34	857.29
		Spiral	Time	**483.11**	1797.76	690.21
			Memory	1209.54	860.61	859.39

Table 2 continued

Distribution	Size of dataset	Insertion order	Results	*hist_tet*	*lst_tet*	*tet_inc*
		Red–black tree with in-order	Time	**320.17**	546.79	329.20
			Memory	1119.42	792.05	789.02
	100,000	Red–black tree with level-order	Time	**363.46**	–	–
			Memory	96.64	–	–
		Random	Time	**1169.01**	–	–
			Memory	89.74	–	–
Planes	1,000,000	Hilbert	Time	232.52	**117.85**	117.88
			Memory	1068.29	938.07	937.58
		Lebesgue	Time	144.61	141.55	**135.94**
			Memory	994.46	959.28	959.00
		H-indexing	Time	**495.13**	916.48	498.65
			Memory	1194.69	**858.49**	859.23
		Spiral	Time	**474.24**	1196.39	563.28
			Memory	1204.28	869.51	867.31
		Red–black tree with in-order	Time	**450.26**	1429.88	669.51
			Memory	1130.02	798.62	795.56
	100,000	Red–black tree with level-order	Time	**415.25**	1711.18	1679.78
			Memory	93.33	97.26	98.91
		Random	Time	**379.71**	1626.50	1562.05
			Memory	88.36	92.20	89.55
3 axes	1,000,000	Hilbert	Time	165.27	**103.81**	104.61
			Memory	1123.82	982.65	985.46
		Lebesgue	Time	142.89	136.10	**132.03**
			Memory	1044.75	1005.12	1003.69
		H-indexing	Time	363.57	452.21	**433.49**
			Memory	1212.87	891.68	886.69
		Spiral	Time	**375.51**	796.20	424.74
			Memory	1204.61	892.23	890.46
		Red–black tree with in-order	Time	**348.14**	715.32	417.06
			Memory	1115.52	792.01	789.53
	100,000	Red–black tree	Time	**334.37**	1725.35	1708.03
		with level-order	Memory	94.47	94.69	95.68
		Random	Time	**391.15**	-	-
			Memory	82.89	-	-
Paraboloid	1,000,000	Hilbert	Time	532.67	98.52	**98.28**
			Memory	1141.28	1082.16	1084.50
		Lebesgue	Time	164.27	**120.63**	114.58
			Memory	1136.32	1116.81	1114.06

Table 2 continued

Distribution	Size of dataset	Insertion order	Results	*hist_tet*	*lst_tet*	*tet_inc*
		H-indexing	Time	**301.50**	370.21	308.80
			Memory	1181.54	875.10	873.92
		Spiral	Time	**305.62**	504.46	322.12
			Memory	1191.97	919.92	919.07
		Red–black tree with in-order	Time	273.01	319.30	**265.72**
			Memory	1100.10	786.41	788.17
	100,000	Red–black tree with level-order	Time	**402.23**	–	–
			Memory	94.58	–	–
		Random	Time	**1174.85**	–	–
			Memory	86.76	–	–
Spiral	1,000,000	Hilbert	Time	231.74	103.83	**102.69**
			Memory	1148.40	1058.16	1058.45
		Lebesgue	Time	152.25	123.15	**119.96**
			Memory	1106.25	1069.73	1075.12
		H-indexing	Time	387.80	452.45	**378.31**
			Memory	1195.54	896.50	895.73
		Spiral	Time	**374.30**	619.77	398.44
			Memory	1182.32	895.50	895.96
		Red–black tree with in-order	Time	335.76	393.25	**330.84**
			Memory	1098.58	786.45	788.91
	100,000	Red–black tree with level-order	Time	**417.84**	–	–
			Memory	98.03	–	–
		Random	Time	**1383.22**	–	–
			Memory	86.39	–	–

using the incremental algorithms with a point-insertion sequence in the order given by the cut-longest-edge kd-tree and Hilbert curve. Table 9 shows experimental tests conducted in three dimensions, to evaluate how the incremental algorithms with a point-insertion sequence given by the cut-longest-edge kd-tree and Hilbert curve perform under the CGAL framework. In accordance with the findings presented in the current literature (Liu et al. 2013), the incremental algorithm with a point-insertion sequence in the order given by the cut-longest-edge kd-tree shows very stable performance, whereas the incremental algorithm with a point-insertion sequence in the order given by the Hilbert curve varies with point distributions.

6 Conclusions

Four point-insertion sequences in incremental algorithms for Delaunay tessellation were proposed: the order given by the red–black tree with in-order and level-order traversals; spiral

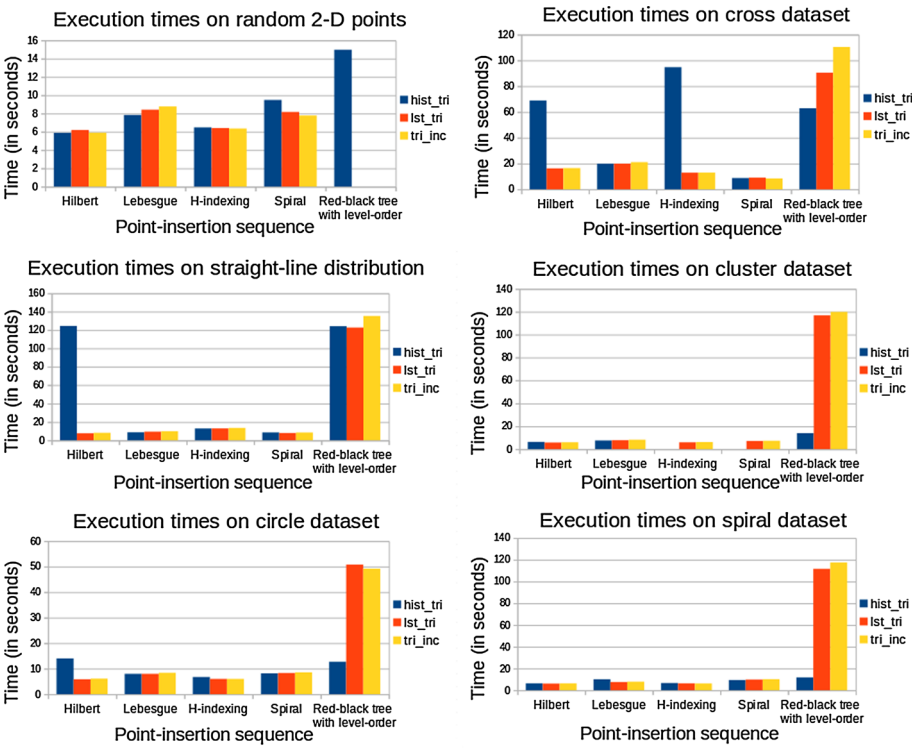

Fig. 3 Computational costs (times lower than 300 s) of incremental algorithms with five point-insertion sequences and three approaches (*hist_tri*, *lst_tri*, and *tri_inc*) to seek the triangle that contains the most recently inserted point into the triangulation tested in six point distributions in the unit square (1,000,000 points): points randomly distributed, cross, straight-line, cluster, circle, and spiral distributions

ordering; and H-indexing. Computational costs and memory requirements of these incremental algorithms were compared to computational costs and memory requirements of four incremental algorithms with point-insertion sequences given by the cut-longest-edge kd-tree (Liu et al. 2013), Hilbert (Liu and Snoeyink 2005) and Lebesgue curves, and random order. Tests were performed in instances up to 8,000,000 points. It follows that the incremental algorithm with a point-insertion sequence given by the cut-longest-edge kd-tree showed the best computational costs in all six 2-D point distributions and in all seven 3-D point distributions tested, in all instances. Therefore, the Liu–Yan–Lo incremental algorithm (Liu et al. 2013) dominated the other seven algorithms, in all point distributions tested. Furthermore, results obtained in this study corroborate the tests presented by Liu et al. (2013), where the Hilbert-curve incremental algorithm was dominated by the cut-longest-edge kd-tree incremental algorithm.

The incremental algorithm using the spiral ordering achieved smaller memory requirements than the cut-longest-edge kd-tree incremental algorithm (Liu et al. 2013) in the largest 2-D instances. In addition, the incremental algorithm using the in-order red–black tree traversal achieved smaller memory requirements than the cut-longest-edge kd-tree incremental algorithm (Liu et al. 2013) in two 3-D point distributions: points around a paraboloid and around a spiral. These results were obtained in the largest instances. However, the memory requirements of the Liu–Yan–Lo incremental algorithm was competitive with those algo-

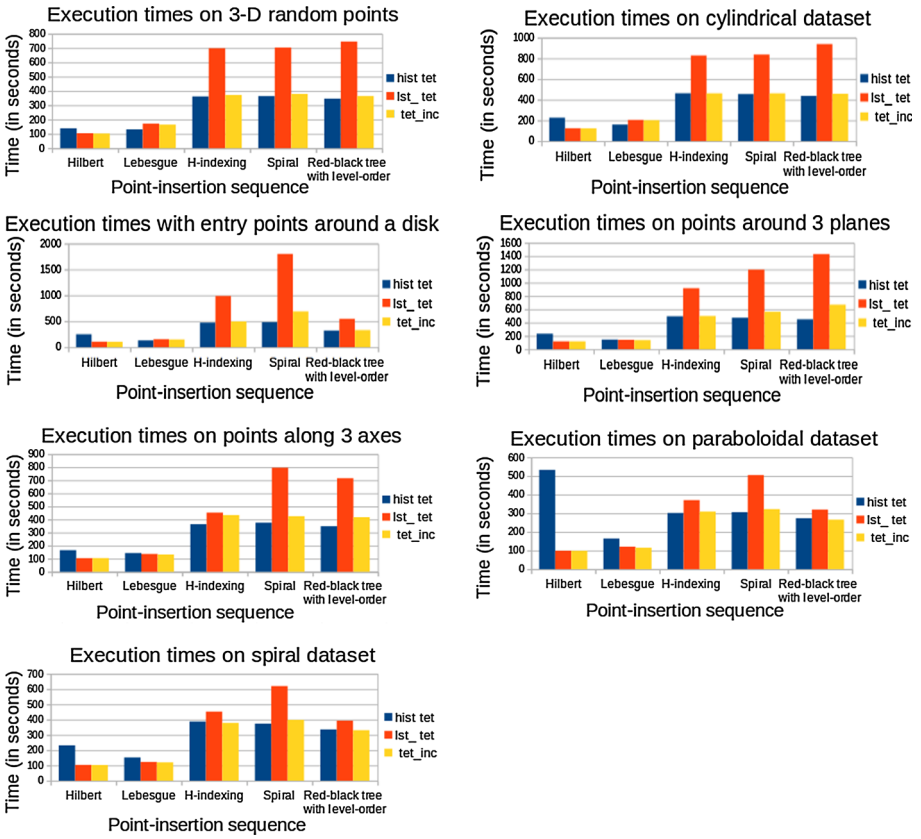

Fig. 4 Computational costs of incremental algorithms with five point-insertion sequences with three approaches (*hist_tet*, *lst_tet*, and *tet_inc*) to seek the tetrahedron that contains the most recently inserted point into the tessellation tested in seven point distributions in the 3-D unit interval (1,000,000 points): random points, points on a cylinder, points around a disk, points around three planes, points along three axes, points around a paraboloid, and points around a spiral

Table 3 Approaches to seek the triangle that contains the most recently inserted point in six 2-D point distributions

Point distribution	Hilbert order	Lebesgue order	*H*-indexing	Spiral order	Red–black tree with in-order traversal
Random	*hist_tri*	*hist_tri*	*tri_inc*	*tri_inc*	*hist_tri*
Cross	*lst_tri*	*hist_tri*	*lst_tri*	*tri_inc*	*hist_tri*
Line	*lst_tri*	*hist_tri*	*hist_tri*	*lst_tri*	*lst_tri*
Cluster	*lst_tri*	*hist_tri*	*lst_tri*	*lst_tri*	*hist_tri*
Circle	*lst_tri*	*lst_tri*	*lst_tri*	*hist_tri*	*hist_tri*
Spiral	*lst_tri*	*lst_tri*	*tri_inc*	*hist_tri*	*hist_tri*

These approaches presented the lowest computational cost in incremental algorithms with point-insertion sequences given by five sequences

Table 4 Approaches to seek the tetrahedron that contains the most recently inserted point in seven 3-D point distributions

Point distribution	Hilbert order	Lebesgue order	*H*-indexing	Spiral order	Red–black tree with in-order traversal
Random	*tet_inc*	*hist_tet*	*hist_tet*	*hist_tet*	*hist_tet*
Cylinder	*tet_inc*	*hist_tet*	*tet_inc*	*hist_tet*	*hist_tet*
Disk	*tet_inc*	*hist_tet*	*hist_tet*	*hist_tet*	*hist_tet*
Planes	*lst_tet*	*tet_inc*	*hist_tet*	*hist_tet*	*hist_tet*
3 axes	*lst_tet*	*tet_inc*	*tet_inc*	*hist_tet*	*hist_tet*
Paraboloid	*tet_inc*	*tet_inc*	*hist_tet*	*hist_tet*	*tet_inc*
Spiral	*tet_inc*	*tet_inc*	*tet_inc*	*hist_tet*	*tet_inc*

These approaches presented the lowest computational cost in incremental algorithms with point-insertion sequences given by five sequences

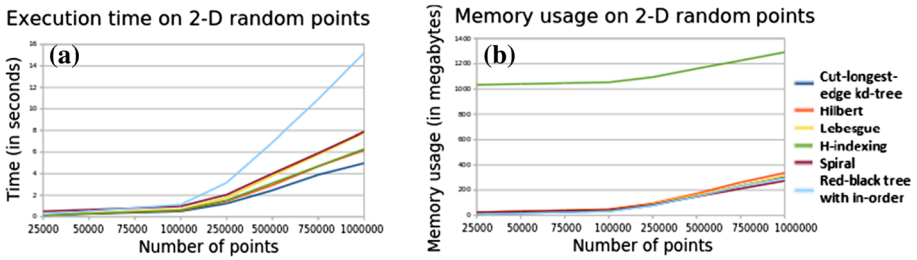

Fig. 5 Computational costs and memory requirements of incremental algorithms with six point-insertion sequences tested in one point distributions in the unit square: **a**, **b** points randomly distributed

rithms in all tests. Therefore, the Liu–Yan–Lo incremental algorithm shows good scalability. Furthermore, one may implement this incremental algorithm with a smaller memory usage than the one used in this work. In particular, the computational costs of the Liu–Yan–Lo and Hilbert-curve incremental algorithms found in our appraisal are slightly higher than those encountered by Liu et al. (2013). This can be explained partly by our usage of specific implementations, and different machines used. To provide more specific detail, the Hilbert-curve and cut-longest-edge kd-tree implementations of Liu et al. (2013) employ the highly enhanced CGAL framework. Despite the higher computational cost encountered here, our programs without using the CGAL framework presented similar behavior to the Liu, Yan and Lo implementations (Liu et al. 2013). This shows evidence that the behavior of these algorithms are independent of implementation.

The Liu–Yan–Lo incremental algorithm (Liu et al. 2013) ensures a more uniform subdivision of the points in the tessellation than the other seven incremental algorithms tested in this computational experiment. Probably for this reason, a smaller number of circumcircle (in 2-D instances) or circumsphere (in 3-D instances), and orientation tests were carried out in relation to the other seven incremental algorithms, and a smaller number of created and destroyed tetrahedrons in the tessellation was required. Therefore, the Liu–Yan–Lo incremental algorithm may be seen as the state-of-the-art algorithm for the generation of Delaunay tessellations in the seven 3-D point distributions tested. Future works will be related to parallel implementations of these algorithms.

Table 5 Computational costs (in s) and memory requirements (in MB) of six incremental algorithms to generate Delaunay triangulations in six point distributions (random, cross, line, cluster, circle, and spiral) in the unit square

Point distribution	Number of points (×1000)	Results	Cut-longest edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral ordering	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Random	25	Time	**0.12**	0.13	0.15	0.15	0.49	0.24	0.01 (H-indexing)	11.20 (H-indexing)	M1
		Mem.	9.94	25.66	25.01	1032.78	24.23	**9.60**	–	–	
	50	Time	**0.25**	0.28	0.32	0.29	0.64	0.52	0.009 (Spiral)	1.96 (Hilbert)	
		Mem.	17.70	33.33	31.71	1039.49	30.61	**16.98**	–	–	
	75	Time	**0.37**	0.42	0.50	0.44	0.79	0.82	0.01 (Spiral)	3.11 (Hilbert)	
		Mem.	25.12	40.21	39.76	1046.21	37.91	**23.35**	–	–	
	100	Time	**0.49**	0.56	0.67	0.60	0.96	1.12	0.01 (Spiral)	1.28 (Spiral)	
		Mem.	33.78	49.34	46.60	1052.68	43.86	**32.05**	–	–	
	250	Time	**1.22**	1.44	1.77	1.51	2.04	3.15	0.01 (Lebesgue)	1.50 (Red–black)	
		Mem.	79.55	95.56	89.58	1093.36	84.53	**78.32**	–	–	
	500	Time	**2.45**	2.92	3.81	3.11	4.00	6.90	0.15 (Lebesgue)	3.94 (Lebesgue)	
		Mem.	155.21	173.07	162.52	1160.45	**148.59**	149.66	–	–	
	750	Time	**3.89**	4.67	5.77	4.69	5.90	10.90	0.21 (Hilbert)	4.68 (Hilbert)	
		Mem.	236.63	261.35	237.36	1225.20	**212.23**	231.09	–	–	
	1000	Time	**4.97**	6.17	7.78	6.28	7.88	15.19	0.24 (Hilbert)	4.52 (Hilbert)	
		Mem.	306.52	336.61	314.21	1290.69	**273.41**	294.34	–	–	
	No. of best r.	Time	8	0	0	0	0	0	–	–	
		Mem.	0	0	0	0	3	5	–	–	

Table 5 continued

Point distribution	Number of points (×1000)	Results	Cut-longest edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral ordering	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Cross	25	Time	**0.12**	0.15	0.16	0.15	0.18	0.21	0.003 (Red–black)	1.49 (Red–black)	M1
		Mem.	10.00	24.32	25.50	1032.55	23.96	**9.64**	–	–	
	50	Time	**0.25**	0.32	0.34	0.30	0.33	0.46	0.01 (H-indexing)	5.20 (H-indexing)	
		Mem.	17.78	30.62	32.89	1038.75	30.48	**17.01**	–	–	
	75	Time	**0.37**	0.49	0.53	0.46	0.50	0.77	0.06 (Red–black)	8.74 (Red–black)	
		Mem.	24.94	37.15	40.91	1045.21	36.64	**24.60**	–	–	
	100	Time	**0.51**	0.68	0.74	0.66	0.65	1.05	0.07 (H-indexing)	11.92 (H-indexing)	
		Mem.	33.39	44.96	48.69	1051.48	44.57	**32.47**	–	–	
	250	Time	**1.23**	1.93	2.17	1.76	1.66	3.77	0.10 (H-indexing)	6.15 (H-indexing)	
		Mem.	79.18	89.46	99.44	1091.04	78.21	**78.14**	–	–	
	500	Time	**2.45**	5.03	5.86	4.32	3.57	13.33	0.13 (H-indexing)	3.17 (H-indexing)	
		Mem.	155.49	175.21	196.75	1154.47	**134.50**	159.38	–	–	
	750	Time	**3.83**	9.92	11.98	8.08	5.88	32.09	0.09 (kd-tree)	1.54 (Spiral)	
		Mem.	237.80	269.89	293.27	1218.55	**181.09**	254.73	–	–	
	1,000	Time	**4.96**	16.41	19.78	13.06	8.47	62.69	0.42 (Red–black)	1.58 (H-indexing)	
		Mem.	307.50	369.79	405.66	1281.23	**229.39**	344.66	–	–	
	No. of best r.	Time	8	0	0	0	0	0	–	–	
		Mem.	0	0	0	0	3	5	–	–	

Table 5 continued

Point distribution	Number of points (×1000)	Results	Cut-longest edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral ordering	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Line	25	Time	**0.11**	0.12	0.12	0.12	0.13	0.17	0.001 (Spiral)	1.21 (Spiral)	M2
		Mem.	10.01	24.61	26.05	1034.11	23.56	**7.85**	–	–	
	50	Time	**0.23**	0.24	0.26	0.26	0.27	0.38	0.003 (Lebesgue)	1.49 (Lebesgue)	
		Mem.	17.79	32.17	34.15	1042.05	29.37	**13.42**	–	–	
	75	Time	**0.35**	0.38	0.39	0.41	0.41	0.65	0.009 (Lebesgue)	2.33 (Lebesgue)	
		Mem.	25.41	40.10	43.44	1049.80	35.53	**20.58**	–	–	
	100	Time	**0.47**	0.52	0.54	0.56	0.56	0.98	0.02 (Spiral)	3.94 (Spiral)	
		Mem.	33.41	48.07	52.98	1058.38	39.89	**24.62**	–	–	
	250	Time	**1.17**	1.41	1.50	1.63	1.47	4.76	0.02 (H-indexing)	1.53 (H-indexing)	
		Mem.	78.22	107.74	114.49	1105.73	70.36	**59.02**	–	–	
	500	Time	**2.33**	3.13	3.44	4.16	3.26	22.08	0.59 (Red–black)	2.70 (Red–black)	
		Mem.	154.34	218.76	234.78	1184.23	**115.21**	115.77	–	–	
	750	Time	**3.66**	5.19	5.85	7.96	5.48	59.09	0.54 (Red–black)	1.40 (H-indexing)	
		Mem.	236.69	362.87	382.12	1271.04	**156.69**	173.67	–	–	
	1000	Time	**4.86**	7.55	8.70	12.89	8.00	122.64	0.11 (H-indexing)	1.25 (Spiral)	
		Mem.	306.55	566.98	592.72	1342.18	**198.92**	227.52	–	–	
	No. of best r.	Time	8	0	0	0	0	0	–	–	
		Mem.	0	0	0	0	3	5	–	–	

Table 5 continued

Point distribution	Number of points (×1000)	Results	Cut-longest edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral ordering	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Cluster	25	Time	**0.11**	0.13	0.15	0.13	0.39	0.23	0.002 (Spiral)	1.28 (Lebesgue)	M2
		Mem.	10.01	24.36	25.26	1032.55	23.94	**9.39**	–	–	
	50	Time	**0.23**	0.27	0.31	0.28	0.61	0.51	0.007 (Red–black)	1.53 (Red–black)	
		Mem.	17.79	30.47	32.65	1038.74	29.93	**16.76**	–	–	
	75	Time	**0.35**	0.40	0.47	0.42	0.76	0.79	0.007 (Spiral)	1.16 (Lebesgue)	
		Mem.	25.65	38.21	40.04	1045.17	35.81	**23.05**	–	–	
	100	Time	**0.47**	0.54	0.64	0.57	0.90	1.08	0.01 (Red–black)	1.52 (H-indexing)	
		Mem.	33.40	44.73	48.62	1053.29	42.05	**32.55**	–	–	
	250	Time	**1.16**	1.37	1.68	1.46	1.84	2.99	0.02 (Lebesgue)	1.56 (Lebesgue)	
		Mem.	78.37	88.46	97.47	1091.61	76.62	**75.51**	–	–	
	500	Time	**2.31**	2.82	3.49	2.94	3.42	6.46	0.03 (Lebesgue)	1.08 (Lebesgue)	
		Mem.	155.40	165.55	185.28	1151.94	**128.82**	143.77	–	–	
	750	Time	**3.59**	4.28	5.44	4.42	5.06	10.34	0.12 (Red–black)	1.34 (Lebesgue)	
		Mem.	237.02	245.89	274.68	1214.02	**180.59**	222.78	–	–	
	1000	Time	**4.63**	5.76	7.50	5.95	7.13	14.09	0.20 (Spiral)	2.87 (Spiral)	
		Mem.	307.62	326.79	364.75	1275.01	**268.70**	286.21	–	–	
	Number of best results	Time	8	0	0	0	0	0	–	–	
		Mem.	0	0	0	0	3	5	–	–	

Table 5 continued

Point distribution	Number of points (×1000)	Results	Cut-longest edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral ordering	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Circle	25	Time	**0.12**	0.13	0.14	0.13	0.23	0.21	0.006 (H-indexing)	4.74 (H-indexing)	M3
		Mem.	10.01	24.13	24.78	1032.57	25.28	**9.65**	–	–	
	50	Time	**0.25**	0.27	0.30	0.28	0.44	0.46	0.008 (Red–black)	1.91 (Red–black)	
		Mem.	17.79	30.21	31.42	1038.77	32.92	**17.22**	–	–	
	75	Time	**0.38**	0.41	0.46	0.42	0.64	0.73	0.007 (Red–black)	1.77 (Hilbert)	
		Mem.	25.40	36.53	39.20	1045.98	39.69	**25.00**	–	–	
	100	Time	**0.50**	0.54	0.62	0.57	0.84	1.00	0.01 (Red–black)	1.34 (Red–black)	
		Mem.	34.10	43.95	46.14	1051.45	49.77	**32.10**	–	–	
	250	Time	**1.23**	1.41	1.69	1.46	2.03	2.73	0.01 (H-indexing)	1.08 (H-indexing)	
		Mem.	79.43	86.28	92.23	1090.49	95.04	**78.81**	–	–	
	500	Time	**2.52**	2.86	3.59	2.98	4.04	5.90	0.11 (Red–black)	3.31 (kd-tree)	
		Mem.	154.72	162.15	175.70	1152.52	160.68	**151.68**	–	–	
	750	Time	**3.81**	4.38	5.67	4.52	6.07	9.32	0.14 (Red–black)	1.76 (Spiral)	
		Mem.	237.06	246.93	265.60	1215.96	**232.66**	233.18	–	–	
	1000	Time	**4.95**	5.78	7.92	6.01	8.02	12.79	0.19 (Red–black)	1.52 (Red–black)	
		Mem.	307.14	336.87	358.16	1275.74	**286.72**	299.14	–	–	
	Number of best results	Time	8	0	0	0	0	0	–	–	
		Mem.	0	0	0	0	2	6	–	–	

Table 5 continued

Point distribution	Number of points (×1000)	Results	Cut-longest edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral ordering	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Spiral	25	Time	**0.12**	0.14	0.16	0.15	0.26	0.24	0.003 (Red–black)	1.59 (Hilbert)	M3
		Mem.	10.00	24.13	24.77	1032.82	25.28	**9.65**	–	–	
	50	Time	**0.25**	0.29	0.33	0.30	0.53	0.50	0.006 (Red–black)	1.33 (Red–black)	
		Mem.	17.69	30.19	31.33	1039.54	32.86	**16.96**	–	–	
	75	Time	**0.39**	0.43	0.52	0.45	0.78	0.78	0.009 (Red–black)	1.25 (Red–black)	
		Mem.	25.17	37.04	39.29	1046.50	40.23	**24.64**	–	–	
	100	Time	**0.51**	0.59	0.69	0.61	1.02	1.06	0.02 (Lebesgue)	3.63 (Lebesgue)	
		Mem.	33.34	43.93	46.78	1054.14	47.42	**32.29**	–	–	
	250	Time	**1.25**	1.47	1.75	1.52	2.38	2.71	0.02 (Lebesgue)	1.35 (Lebesgue)	
		Mem.	79.37	86.91	89.65	1094.21	91.70	**77.95**	–	–	
	500	Time	**2.53**	2.96	3.69	3.08	4.69	5.59	0.07 (Spiral)	1.56 (Spiral)	
		Mem.	154.89	162.00	172.20	1159.52	148.86	**146.21**	–	–	
	750	Time	**3.86**	4.56	5.57	4.64	7.02	8.42	0.15 (Spiral)	2.18 (Spiral)	
		Mem.	237.59	245.30	262.82	1224.70	**211.85**	227.44	–	–	
	1000	Time	**4.97**	6.08	7.57	6.27	9.41	11.68	0.12 (H-indexing)	1.93 (H-indexing)	
		Mem.	306.83	331.50	353.91	1287.79	**265.27**	289.21	–	–	
	Number of best results	Time	8	0	0	0	0	0	–	–	
		Mem.	0	0	0	0	2	6	–	–	

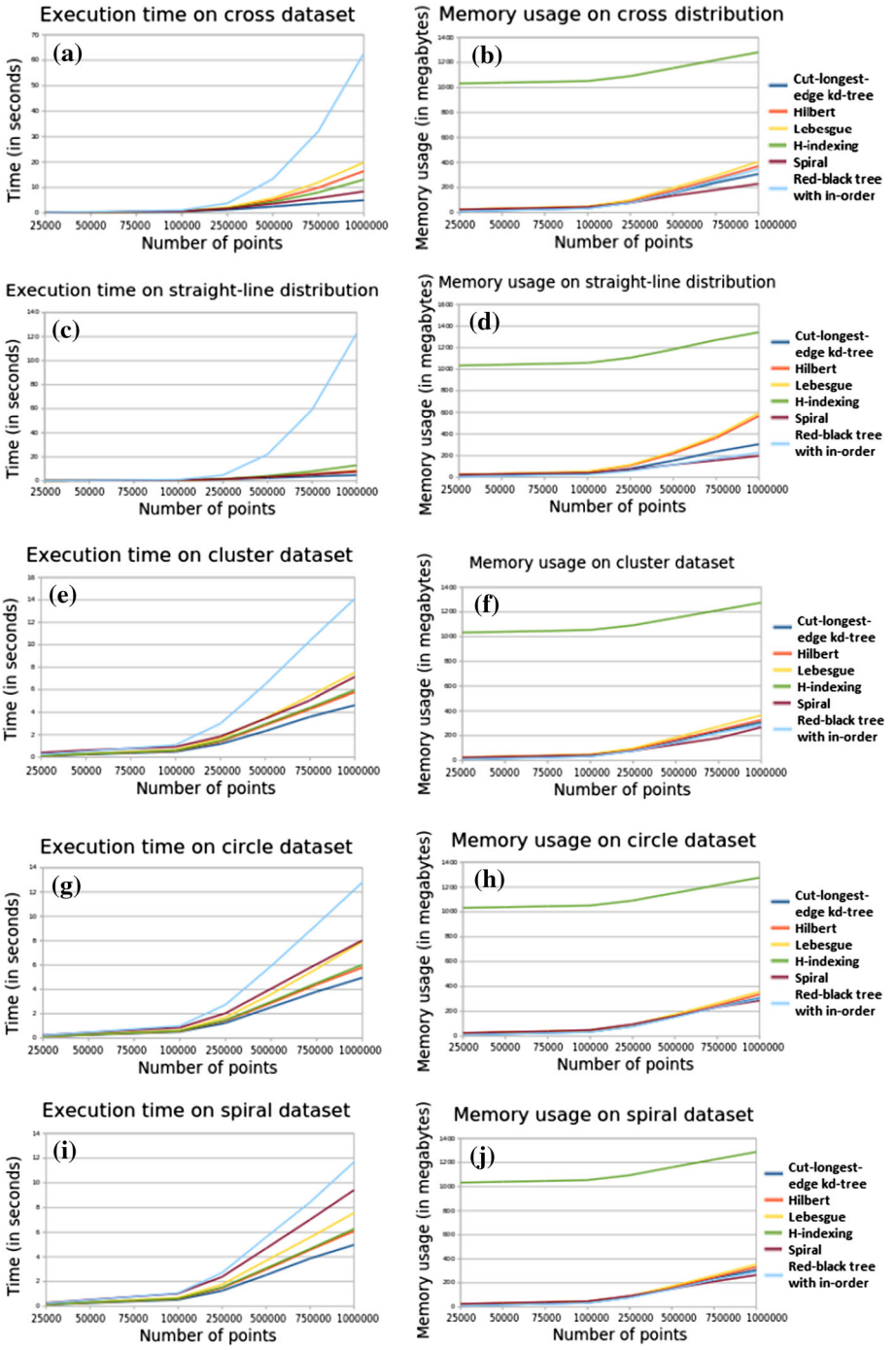

Fig. 6 Computational costs and memory requirements of incremental algorithms with six point-insertion sequences tested in five point distributions in the unit square: **a**, **b** cross distribution; **c**, **d** line distribution; **e**, **f** cluster distribution; **g**, **h** circle distribution; **i**, **j** spiral distribution

Table 6 Computational costs (in s) and memory requirements (in MB) of six incremental algorithms to generate Delaunay tessellations in seven 3-D point distributions (random points, points on a cylinder, points around a disk, points around three planes, points along three axes, points around a paraboloid, and points around a spiral) in the 3-D unit interval

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Random	25	Time	**1.55**	2.18	2.31	5.38	5.40	5.30	0.07 (Hilbert)	3.60 (Hilbert)	M2
		Mem.	**27.74**	90.43	91.78	33.05	35.59	31.35	–	–	
	50	Time	**3.12**	4.55	4.93	12.11	12.13	11.85	0.15 (Red–black)	1.86 (Lebesgue)	
		Mem.	**51.01**	111.32	113.62	65.41	62.33	60.29	–	–	
	75	Time	**4.73**	6.89	7.68	19.40	19.38	18.80	0.26 (Lebesgue)	3.42 (Lebesgue)	
		Mem.	**70.97**	131.10	135.76	93.50	93.44	90.92	–	–	
	100	Time	**6.29**	9.41	10.36	27.00	27.02	26.21	0.23 (H-indexing)	1.91 (Hilbert)	
		Mem.	**97.86**	151.20	160.50	120.62	123.50	117.24	–	–	
	250	Time	**15.49**	24.21	27.58	76.85	76.82	73.77	0.70 (H-indexing)	1.37 (Lebesgue)	
		Mem.	**226.48**	275.25	287.86	301.44	299.43	284.25	–	–	
	500	Time	**31.07**	50.51	59.40	168.04	167.74	160.40	1.71 (Red–black)	1.07 (Red–black)	
		Mem.	**448.72**	481.32	514.43	603.69	601.80	564.22	–	–	
	750	Time	**47.18**	76.56	94.00	263.35	263.59	254.35	3.61 (Red–black)	1.42 (Red–black)	
		Mem.	690.43	**689.92**	750.64	911.16	910.32	859.34	–	–	
	1000	Time	**62.23**	103.19	130.68	362.57	365.60	345.50	3.39 (Red–black)	1.00 (Lebesgue)	
		Mem.	**893.31**	898.66	958.80	1198.89	1205.63	1128.02	–	–	
	8000	Time	**336.12**	591.17	–	–	–	–	–	–	M4
		Mem.	7121.36	**7030.11**	–	–	–	–	–	–	
	N. of Best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	7	2	0	0	0	0	–	–	

Table 6 continued

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Cylinder	25	Time	**1.65**	2.50	2.64	6.28	6.31	6.39	0.09 (Red–black)	3.17 (Hilbert)	M1
		Mem.	**28.48**	91.12	93.23	28.32	33.45	30.61	–	–	
	50	Time	**3.36**	5.26	5.72	14.70	14.30	14.49	0.19 (Red–black)	1.38 (kd-tree)	
		Mem.	**51.25**	113.69	116.49	50.26	63.81	59.51	–	–	
	75	Time	**5.12**	8.06	8.88	23.38	23.16	23.38	0.20 (Red–black)	1.34 (kd-tree)	
		Mem.	**71.21**	132.17	136.76	68.02	93.67	90.28	–	–	
	100	Time	**6.77**	10.81	12.16	33.01	32.36	32.56	0.36 (Red–black)	1.66 (Hilbert)	
		Mem.	**97.89**	150.54	158.26	87.48	120.63	116.72	–	–	
	250	Time	**16.81**	28.85	32.68	95.41	93.35	92.51	0.80 (H-indexing)	1.55 (Hilbert)	
		Mem.	**225.30**	277.94	289.88	219.20	299.10	284.30	–	–	
	500	Time	**33.46**	60.06	69.42	213.56	206.28	201.26	0.92 (H-indexing)	1.31 (Hilbert)	
		Mem.	**450.32**	483.92	513.04	432.24	600.68	564.48	–	–	
	750	Time	**51.17**	91.51	111.75	332.63	326.09	314.84	0.93 (Spiral)	1.18 (kd-tree)	
		Mem.	**688.90**	693.57	744.62	640.55	909.18	859.88	–	–	
	1000	Time	**67.32**	124.69	160.59	460.05	457.07	434.38	2.53 (Hilbert)	2.03 (Hilbert)	
		Mem.	**893.22**	899.47	961.05	861.23	1206.66	1131.80	–	–	
	8000	Time	**326.67**	602.13	–	–	–	–	–	–	M4
		Mem.	7122.02	**7113.43**	–	–	–	–	–	–	
	N. of best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	8	1	0	0	0	0	–	–	

Table 6 continued

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Disk	25	Time	**1.62**	2.19	2.33	7.70	7.74	4.54	0.08 (Spiral)	1.14 (Spiral)	M3
		Mem.	**27.78**	88.00	90.77	35.09	34.93	33.07	–	–	
	50	Time	**3.29**	4.57	4.91	16.93	17.03	10.32	0.09 (H-indexing)	1.30 (kd-tree)	
		Mem.	**51.33**	110.07	112.55	66.29	65.75	59.32	–	–	
	75	Time	**4.99**	7.00	7.67	26.72	26.91	16.62	0.16 (Red–black)	0.96 (Red–black)	
		Mem.	**71.22**	133.23	140.23	98.46	97.86	87.02	–	–	
	100	Time	**6.65**	9.51	10.49	37.21	37.50	23.26	0.35 (Spiral)	1.40 (Red–black)	
		Mem.	**96.41**	154.48	159.86	124.82	128.23	114.33	–	–	
	250	Time	**16.39**	24.90	28.33	102.93	103.46	66.45	0.44 (H-indexing)	1.04 (kd-tree)	
		Mem.	**226.93**	282.31	297.68	304.01	304.57	278.89	–	–	
	500	Time	**33.01**	51.02	60.53	220.94	223.66	147.95	1.68 (Spiral)	0.86 (Lebesgue)	
		Mem.	**447.88**	502.11	530.46	611.03	606.86	554.72	–	–	
	750	Time	**50.26**	78.21	95.72	344.74	349.79	232.88	1.97 (Red–black)	1.03 (kd-tree)	
		Mem.	**690.00**	728.43	778.87	923.04	924.04	846.84	–	–	
	1000	Time	**66.43**	105.94	132.22	471.77	482.99	321.42	3.31 (Spiral)	1.46 (Hilbert)	
		Mem.	**889.88**	958.81	1020.25	1211.93	1209.54	1115.77	–	–	
	8000	Time	**328.78**	658.80	–	-	–	–	–	–	M4
		Mem.	**7116.22**	8178.34	–	–	–	–	–	–	
	No. of best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	9	0	0	0	0	0	–	–	

Table 6 continued

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Planes	25	Time	**1.68**	2.35	2.43	7.03	6.93	6.64	0.08 (Lebesgue)	3.38 (Lebesgue)	M1
		Mem.	**27.72**	90.05	91.34	32.28	31.86	32.25	–	–	
	50	Time	**3.39**	5.00	5.00	15.94	15.60	15.11	0.22 (Spiral)	1.73 (Hilbert)	
		Mem.	**51.06**	109.12	113.35	65.35	63.08	61.53	–	–	
	75	Time	**5.17**	7.67	7.81	25.60	24.83	24.31	0.35 (Red–black)	1.44 (Red–black)	
		Mem.	**71.29**	131.50	135.14	95.17	95.72	89.46	–	–	
	100	Time	**6.84**	10.31	10.73	35.71	34.82	33.77	0.41 (H-indexing)	1.28 (Lebesgue)	
		Mem.	**93.37**	153.27	155.39	121.66	123.75	116.60	–	–	
	250	Time	**17.04**	27.14	29.30	101.10	99.21	95.48	0.93 (H-indexing)	1.55 (Hilbert)	
		Mem.	**224.19**	279.37	286.74	301.96	298.70	281.64	–	–	
	500	Time	**33.94**	56.55	63.26	223.36	219.17	207.58	2.77 (Spiral)	1.26 (Spiral)	
		Mem.	**447.83**	494.51	502.74	599.68	595.09	565.36	–	–	
	750	Time	**51.49**	86.59	99.36	353.74	343.98	327.81	2.32 (H-indexing)	1.07 (kd-tree)	
		Mem.	**684.46**	711.62	735.92	911.46	911.18	856.16	–	–	
	1000	Time	**68.05**	116.37	137.38	492.57	474.55	451.77	2.98 (H-indexing)	0.88 (Hilbert)	
		Mem.	**890.43**	938.07	959.00	1194.69	1204.28	1130.02	–	–	
	8000	Time	**328.04**	572.69	–	–	–	–	–	–	M4
		Mem.	**7105.88**	7905.29	–	–	–	–	–	–	
	No. of best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	9	0	0	0	0	0	–	–	

Table 6 continued

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
3 axes	25	Time	**1.56**	2.17	2.25	6.79	5.65	5.44	0.05 (Red–black)	0.96 (kd-tree)	M2
		Mem.	28.50	91.60	89.01	**25.99**	33.67	35.39	–	–	
	50	Time	**3.19**	4.53	4.74	15.76	12.69	11.99	0.10 (Red–black)	1.08 (Hilbert)	
		Mem.	50.81	112.28	113.03	**47.78**	61.44	59.22	–	–	
	75	Time	**4.82**	6.92	7.35	25.64	20.13	19.11	0.20 (Spiral)	1.03 (Spiral)	
		Mem.	**72.25**	133.15	132.87	72.51	95.84	89.77	–	–	
	100	Time	**6.61**	9.41	10.01	35.93	28.33	26.59	0.17 (Spiral)	0.70 (Hilbert)	
		Mem.	101.23	152.18	154.91	**93.58**	123.88	116.58	–	–	
	250	Time	**15.68**	24.24	27.85	100.05	79.60	74.82	1.15 (Red–black)	1.54 (Red–black)	
		Mem.	227.46	282.32	284.21	**226.13**	301.30	280.76	–	–	
	500	Time	**31.64**	50.69	61.67	214.41	183.58	161.99	11.56 (Spiral)	6.29 (Spiral)	
		Mem.	462.09	511.54	521.37	**449.78**	599.72	556.26	–	–	
	750	Time	**47.28**	76.66	98.32	329.01	280.74	253.84	15.74 (Spiral)	5.60 (Spiral)	
		Mem.	697.89	742.61	767.21	**665.41**	901.47	845.17	–	–	
	1000	Time	**63.27**	103.80	136.62	452.71	397.51	349.48	30.14 (Spiral)	7.58 (Spiral)	
		Mem.	923.35	982.65	1003.69	**886.69**	1204.61	1115.52	–	–	
	8000	Time	**331.98**	795.01	–	–	–	–	–	–	M4
		Mem.	**7118.57**	8294.71	–	–	–	–	–	–	
	No. of best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	2	0	0	7	0	0	–	–	

Table 6 continued

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Paraboloid	25	Time	**1.54**	2.04	2.30	4.21	4.24	3.79	0.03 (H-indexing)	1.15 (Lebesgue)	M3
		Mem.	27.46	91.19	90.17	33.99	32.00	**22.54**	–	–	
	50	Time	**3.14**	4.22	4.69	9.50	9.57	8.54	0.10 (Red–black)	1.36 (kd-tree)	
		Mem.	48.55	110.39	112.71	58.47	58.00	**44.35**	–	–	
	75	Time	**4.85**	6.49	7.21	15.21	15.29	13.75	0.10 (Red–black)	1.04 (Hilbert)	
		Mem.	68.77	133.72	136.05	89.55	88.60	**62.32**	–	–	
	100	Time	**6.45**	8.68	9.79	21.35	21.51	19.06	0.18 (Spiral)	1.14 (Lebesgue)	
		Mem.	93.07	157.14	156.80	117.91	116.78	**82.70**	–	–	
	250	Time	**15.85**	22.46	26.26	61.87	62.95	54.94	0.92 (Spiral)	1.46 (Spiral)	
		Mem.	218.98	302.44	310.82	296.46	292.18	**196.33**	–	–	
	500	Time	**32.10**	46.25	54.04	136.56	138.80	122.23	1.16 (Red–black)	1.39 (Hilbert)	
		Mem.	437.89	555.97	566.90	597.09	589.57	**393.23**	–	–	
	750	Time	**48.74**	70.84	83.65	216.37	218.72	194.57	1.34 (Red–black)	0.75 (Hilbert)	
		Mem.	675.13	814.44	849.87	897.44	887.51	**588.97**	–	–	
	1000	Time	**64.25**	98.33	114.73	301.55	304.12	267.43	3.01 (Red–black)	1.16 (Hilbert)	
		Mem.	873.85	1083.16	1114.06	1181.01	1191.97	**788.17**	–	–	
	8000	Time	**322.50**	521.86	–	–	–	–	–	–	M4
		Mem.	**7058.76**	8065.10	–	–	–	–	–	–	
	No. of best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	1	0	0	0	0	8	–	–	

Table 6 continued

Point distribution	Number of points (×1000)	Results	Cut-longest-edge kd-tree	Hilbert curve	Lebesgue curve	H-indexing	Spiral indexing	In-order red–black tree	Largest standard deviation	Largest coefficient of variation	M.
Spiral	25	Time	**1.67**	2.23	2.53	5.08	5.01	4.57	0.09 (H-indexing)	2.78 (Lebesgue)	M1
		Mem.	27.01	89.83	92.26	25.83	31.40	**24.46**	–	–	
	50	Time	**3.35**	4.55	5.04	11.52	11.40	10.31	0.13 (H-indexing)	1.20 (H-indexing)	
		Mem.	49.00	114.05	113.55	48.82	59.69	**43.24**	–	–	
	75	Time	**5.05**	6.88	7.91	18.60	18.41	16.62	0.21 (H-indexing)	1.16 (H-indexing)	
		Mem.	69.46	134.25	135.31	69.82	90.93	**63.56**	–	–	
	100	Time	**6.67**	9.44	10.40	26.24	25.85	23.48	0.30 (Red–black)	2.47 (Hilbert)	
		Mem.	93.36	154.94	157.37	89.32	119.69	**82.76**	–	–	
	250	Time	**16.63**	24.28	26.59	77.34	76.31	67.63	0.48 (Red–black)	0.97 (Lebesgue)	
		Mem.	222.31	294.15	299.92	230.89	294.41	**200.91**	–	–	
	500	Time	32.99	49.90	56.48	173.47	173.22	152.28	4.54 (Red–black)	2.88 (Red–black)	
		Mem.	443.39	543.96	564.01	436.55	585.70	**395.25**	–	–	
	750	Time	50.35	75.88	87.50	272.01	271.54	240.27	4.43 (H-indexing)	1.62 (H-indexing)	
		Mem.	681.64	788.30	816.44	668.12	885.87	**589.22**	–	–	
	1000	Time	66.47	102.91	118.47	380.35	375.17	332.31	6.20 (H-indexing)	1.63 (H-indexing)	
		Mem.	885.91	1058.45	1075.12	895.73	1182.32	**788.91**	–	–	
	8000	Time	**324.32**	551.40	–	-	–	–	–	–	M4
		Mem.	**7101.38**	8069.85	–	–	–	–	–	–	
	No. of best r.	Time	9	0	0	0	0	0	–	–	–
		Mem.	1	0	0	0	0	8	–	–	

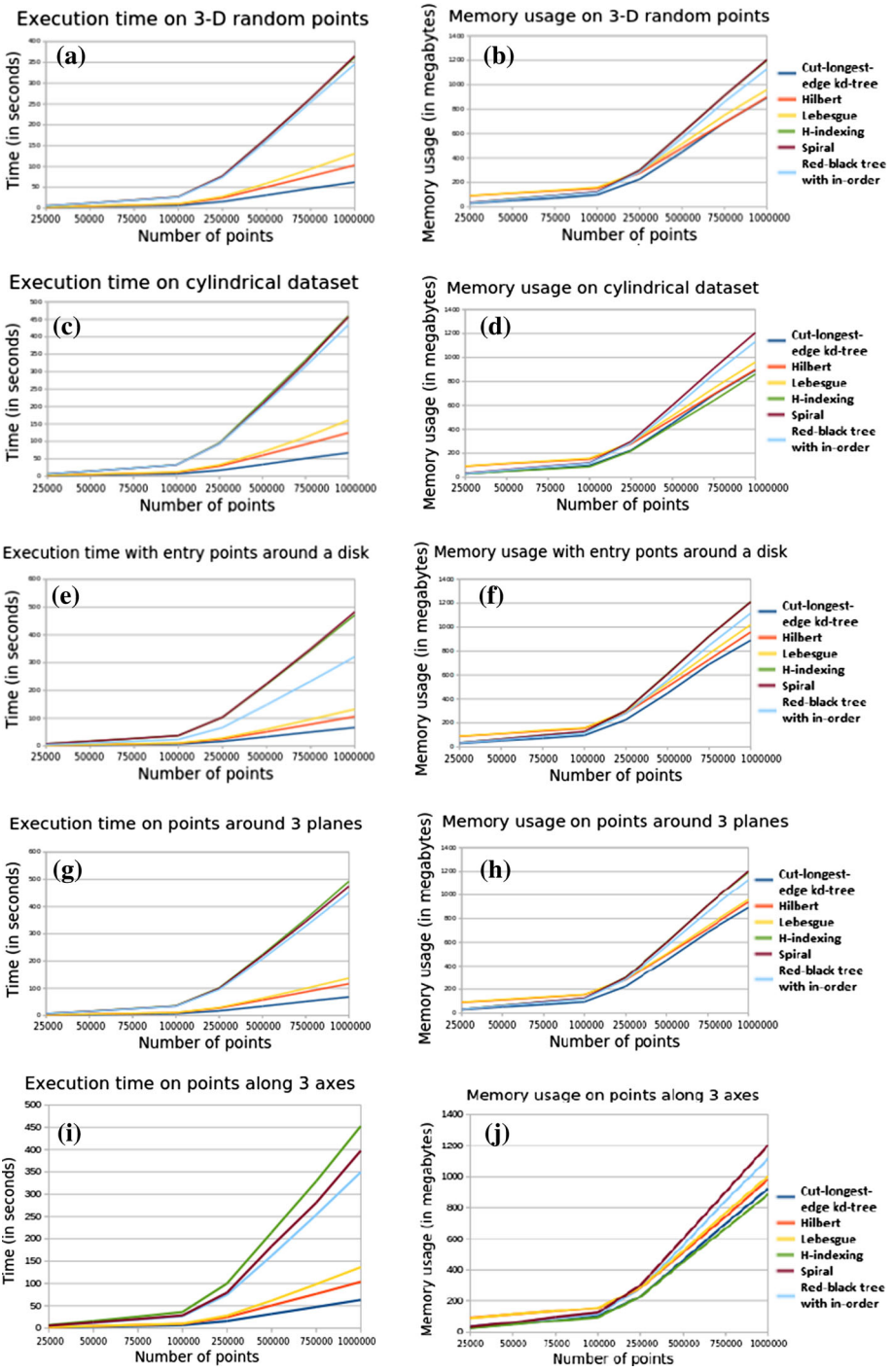

Fig. 7 Computational costs and memory requirements of incremental algorithms with six point-insertion sequences tested in four point distributions in the 3-D unit interval: **a**, **b** random points; **c**, **d** points on a cylinder; **e**, **f** points around a disk; **g**, **h** points around three planes; **i**, **j** points along three axes

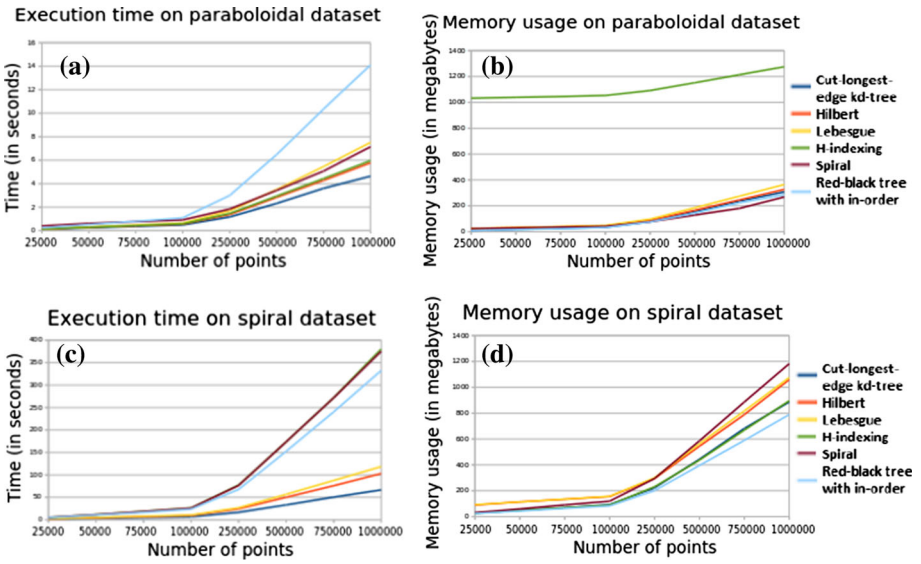

Fig. 8 Computational costs and memory requirements of incremental algorithms with six point-insertion sequences tested in three point distributions in the 3-D unit interval: **a**, **b** points around a paraboloid; and **c**, **d** points around a spiral

Table 7 Cache misses [independent first-level instruction (I1), number of L3 instruction misses (L3i), data cache (D1), number of L3 data misses (L3d), combined instruction and data figures for the L3 cache (L3) miss rates] of the implementations for the incremental algorithms with a point-insertion sequence in the order given by the cut-longest-edge kd-tree and Hilbert curve. Seven point distributions (random points, points on a cylinder, points around a disk, points around three planes, points along three axes, points around a paraboloid, and points around a spiral) were evaluated in the 3-D unit interval. Executions performed on the M4 machine

Point-insertion sequence	Number of points ($\times 10^6$)	I1 (%)	L3i (%)	D1 (%)	L3d (%)	L3 (%)	3-D point distribution
Cut-longest-edge kd-tree	8	0.01	0.0	0.4	0.0	0.0	7 3-D point distributions
	1						
Hilbert curve	8			1.7			Cylinder
				2.7			3 Axes
		0.00		1.1			Spiral
				1.5			Random
				2.3			Disk
				2.6			Planes
				11.2	5.0	1.8	Paraboloid
	1			0.9	0.0	0.0	Random
							Cylinder
							3 Axes
				0.6			Disk
				0.7			Planes
				10.5			Paraboloid
				0.7	0.1		Spiral

Table 8 Number of circumsphere tests, and numbers of created and destroyed tetrahedrons of incremental algorithms with a point-insertion sequence in the order given by the cut-longest-edge kd-tree and Hilbert curve tested in seven point distributions (random points, points on a cylinder, points around a disk, points around three planes, points along three axes, points around a paraboloid, and points around a spiral) in the 3-D unit interval (8×10^6 million points). Executions performed on the M4 machine

Point-insertion sequence	Number of circumsphere tests	Number of tetrahedrons created	Number of tetrahedrons destroyed	3-D point distribution
Cut-longest-edge kd-tree	377,886,370	213,889,614	159,825,125	Random
Hilbert curve	553,308,719	299,596,298	245,531,798	
Cut-longest-edge kd-tree	378,368,065	214,133,904	160,053,937	Cylinder
Hilbert curve	555,573,159	300,176,462	246,096,497	
Cut-longest-edge kd-tree	378,022,644	213,862,664	159,849,400	Disk
Hilbert curve	551,076,946	297,508,522	243,495,309	
Cut-longest-edge kd-tree	384,528,669	213,866,244	159,852,833	3 Axes
Hilbert curve	714,881,399	299,155,300	245,141,860	
Cut-longest-edge kd-tree	384,236,914	213,309,526	159,425,501	Planes
Hilbert curve	712,582,468	295,864,990	241,981,054	
Cut-longest-edge kd-tree	372,279,246	210,162,332	156,775,498	Paraboloid
Hilbert curve	570,664,906	278,895,322	225,508,444	
Cut-longest-edge kd-tree	377,903,796	212,151,622	158,334,481	Spiral
Hilbert curve	535,212,116	284,673,158	230,855,990	

Table 9 Results of incremental algorithms with a point-insertion sequence given by the cut-longest-edge kd-tree and Hilbert curve tested in seven point distributions (random points, points on a cylinder, points around a disk, points around three planes, points along three axes, points around a paraboloid, and points around a spiral) in the 3-D unit interval (10×10^6 million points)

Point-insertion sequence	Random	Cylinder	Disk	3 Axes	Planes	Paraboloid	Spiral
Cut-longest-edge kd-tree	50	49	48	50	48	46	47
Hilbert curve	89	86	96	461	118	64	79

Executions performed on the M4 machine (CPU times in s)

Acknowledgements This work was developed with the support of CAPES - Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (Coordination for Enhancement of Higher Education Personnel, in Brazil), and FAPEMIG - Fundação de Amparo à Pesquisa do Estado de Minas Gerais (Minas Gerais Research Support Foundation, in Brazil). We would like to thank the reviewers for their valuable comments and suggestions.

References

Amenta N, Choi S, Rote G (2003) Incremental constructions con BRIO. Proceedings of the nineteenth annual symposium on Computational geometry., SCG'03ACM, New Yord, NY, pp 211–219

Bader M (2012) Space-filling curves: an introduction with applications in scientific computing. Springer Sc. & Business Media, Berlin

Boissonnat JD, Devillers O, Samuel H (2009) Incremental construction of the Delaunay graph in medium dimension. Proceedings of the 25th Annual Symposium on Computational Geometry. Aarhus, Denmark, pp 208–216

Buchin K (2005) Constructing Delaunay triangulations along space-filling curves. Proceedings of the 2nd International Symposium Voronoi Diagrams (ISVD) in Science and Engineering. Seoul, Korea, pp 184–195

Buchin K (2007) Organizing point sets: Space-filling curves, Delaunay tessellations of random point sets, and flow complexes. Ph.D. thesis, Free University, Berlin

Cachegrind: a cache and branch-prediction profiler: Webpage. URL: http://valgrind.org/docs/manual/cg-manual.html. Accessed May 19, 2016

Carey GF (1997) Computational grids: generations, adaptation & solution strategies. CRC Press, Boca Raton

Duff IS, Meurant GA (1989) The effect of ordering on preconditioned conjugate gradients. BIT Numer Math 29(4):635–657

Edelsbrunner H (2001) Geometry and topology for mesh generation. Cambridge University Press, New York, Cambridge monographs on applied and computational mathematics

Gonzaga de Oliveira SL, Nogueira JR, Tavares JMRS (2014) A systematic review of algorithms with linear-time behaviour to generate Delaunay and Voronoi tessellations. CMES -. CMES Comput Model Eng Sci 100(1):31–57

Liu JF, Yan JH, Lo SH (2013) A new insertion sequence for incremental Delaunay triangulation. Acta Mechanica Sinica 29(1):99–109

Liu Y, Snoeyink J (2005) A comparison of five implementations of 3D Delaunay Tessellation. In: Goodman J, Pach J, Welzl E (eds) Combinatorial and computational geometry, vol 52. MSRI Publications, Cambridge, pp 439–458

Lo DSH (2015) Finite element mesh generation. CRC Press, Boca Raton

Nethercot N, Seward J (2007) Valgrind: a framework for heavyweight dynamic binary instrumentation. In: Proceedings of ACM SIGPLAN 2007 Conference on Programming Language Design and Implementation (PLDI 2007). San Diego, CA

Niedermeier R, Reinhardt K, Sanders P (2002) Towards optimal locality in mesh-indexings. Discrete Appl Math 117(1–3):211–237

Schamberger S, Wierum JM (2005) Partitioning finite element meshes using space-filling curves. Future Gener Comput Syst 21(5):759–766

Schrijvers O, van Bommel F, Buchin K (2013) Delaunay triangulations on the Word RAM: Towards a practical worst-case optimal algorithm. Proceedings of the 10th International Symposium onVoronoi Diagrams in Science and Engineering (ISVD). Saint Petersburg, Russia, pp 7–15

Snoeyink J, Liu Y (2005) TESS3: a program to compute 3D Delaunay tessellations for well-distributed points. In: Proceedings of the 2nd International Symposium Voronoi Diagrams (ISVD) in Science and Engineering. Seoul, Korea

Zhou S, Jones CB (2005) HCPO: an efficient insertion order for incremental Delaunay triangulation. Inf Process Lett 93:37–42