Blocked Randomized Incremental Constructions

Nina Amenta and Sunghee Choi [†]
Technical Report number TR-02-54
University of Texas at Austin

Abstract

Randomized incremental constructions are widely used in computational geometry, but they perform very badly on large data because of their inherently random memory access patterns. We define an insertion order which removes enough randomness to significantly improve performance, but leaves enough randomness so that the algorithms remain theoretically optimal, or nearly so.

1 Introduction

A look at recent textbooks [1, 2] shows that randomized incremental algorithms are a central part of computational geometry. Many randomized incremental algorithms construct geometric structures; one of particular importance is the randomized incremental construction of the Delaunay triangulation of a set of input points. The algorithm is simple to state: insert the points into the triangulation one by one in random order, updating the triangulation at each insertion. It is also worst-case optimal and (compared to the alternatives) easy to implement robustly. This accounts for its importance in practice; there are several robust and efficient implementations for 3D Delaunay triangulation, including Clarkson's `hull`, the CGAL Delaunay hierarchy function, and Shewchuk's `pyramid`.

Given these excellent programs for three-dimensional Delaunay triangulation, it is natural to want to apply them to the large data sets which arise in applications such as mesh generation or surface reconstruction. But the optimality of randomized incremental algorithms is based on accessing the geometric data structures randomly, and random access to large data structures works very poorly with modern memory hierarchies. Most virtual memory systems cache recently used data in memory, on the assumption of *locality or reference*, that is, that recently used data is likely to be used again soon. Randomized incremental programs violate this assumption, and soon after the data structure exceeds the size of physical memory, thrashing occurs and the program grinds (audibly!) to a halt [3].

A simple fix is to insert points in an order which improves the locality of reference, while preserving enough randomness to retain the optimality of the algorithm. In section 2 we present such an insertion order, which we call *blocked randomized*. We prove in sections 3, 4 that this order gives

[†]Computer Sciences Dept., Austin, TX 78712, USA. Supported by an NSF CAREER award and an Alfred P. Sloan Foundation Research Fellowship. Contact: `amenta,sunghee@cs.utexas.edu`

an optimal algorithm for 3D Delaunay triangulation in the worst case, and reamins nearly optimal under less severe but realistic assumptions about the output complexity. Using a blocked randomized ordering with `pyramid`, we give experimental evidence in section 5 that we can indeed solve much larger problems than we could before. We conclude in section 6 with a discussion of when this analysis is applicable, in particular to the trapezoidation of sets of segments in the plane, and we point out some directions for future work.

The development of randomized incremental algorithms and their analysis was a major project of computational geometry in the late eighties and early nineties, as described in textbooks [2, 1] and surveys [4, 5]. We touch on a few relevant highlights. A classic paper by Clarkson and Shor [6] showed that the randomized incremental paradigm could be applied to many problems, and gave a general analysis. Mulmuley [7, 8] and and Clarkson, Mehlhorn and Seidel [9], among others, extended this theory. Seidel [10], harking back to an an idea in an early paper of Chew [12], popularized a simplifying idea called *backwards analysis*. Unfortunately we cannot see how to apply backwards analysis when using a blocked randomized insertion order, so we build on results from the earlier work, in particular the bounds on $\leq k$-sets from Clarkson and Shor, and Mulmuley's idea of probabilistic games. We should also mention a nice paper by Devillers and Guigue [17], similar in spirit to this one, which analyzed the tradeoff between on-line and randomized insertion order.

The traditional approach to thrashing is to develop explicit out-of-core algorithms, generally using divide-and-conquer. Unfortunately the divide-and-conquer paradigm seems to be much less practical than the randomized incremental paradigm in computational geometry; an exception is the practical parallel 2D Delaunay triangulation algorithm of Blelloch, Miller, and Talmor [13]. Their approach, however, does not immediately apply to either three-dimensions or out-of-core computation.

2 The insertion order

We define a *blocked randomized insertion order* for a set P of n input objects, which we shall call points. We partition P arbitrarily into *blocks*; there can be any number of blocks, and they may be of different sizes.

The insertion order of the n points is then determined as follows. There are $\lg n + 1$ phases, beginning with phase zero. In each phase j, we visit all of the blocks in any arbitrary order. Within each block, we visit the uninserted points in the block in random order, and select each uninserted point for insertion with probability $2^j/n$. Since we examine $O(n)$ points in each of $O(\lg n)$ phases, the total time spent determining the insertion order is $O(n \lg n)$.

Of course in practice we choose a blocking scheme which improves locality of reference; blocks correspond to contiguous regions of three-dimensional space, eg. the cells of a kd-tree. We also visit the blocks in an order chosen to improve locality of reference.

The intuition is that in the early phases, few if any points are inserted per block, while in the last phase, all uninserted points in a block are inserted in random order. So the insertions in the early phases tend to be sprinkled nearly randomly across all the data, producing a nicely balanced data structure, while in the later phases they are clustered within blocks, accessing local regions of the data structure mostly separately.

3 Key Lemma

We analyze the use of a blocked insertion order in the context of the incremental construction of a three-dimensional Delaunay triangulation. There are $O(n^4)$ *possible tetrahedra* determined by choosing four points of P as the vertices. Now consider an incremental construction of the Delaunay triangulation. Not every possible tetrahedron *appears* as part of one of the intermediate triangulations, or in the final triangulation. We begin our analysis by estimating the probability that a possible tetrahedron does in fact appear during a run of the blocked randomized incremental construction.

We will use some terminology due to Mulmuley. Consider a tetrahedron τ with four points in P as its vertices, known as its *triggers*, and with s other points of P contained in its circumsphere, known as its *stoppers*. Tetrahedron τ appears in some Delaunay triangulation if all of its triggers are selected for insertion before any of its stoppers. The probability that τ appears during the construction thus depends on s; if $|s| = 0$, for instance, τ belongs to the final Delaunay triangulation and the probability that it appears is one. It also depends on the particular blocked randomized insertion order, since the order in which triggers and stoppers are considered for insertion is not random.

Observation 1 *The blocked randomized insertion orders for which τ is* most *likely to appear are those in which*
1) the blocks containing the triggers are disjoint from the blocks containing the stoppers, and
2) in every phase, the blocks containing all of the triggers precede the blocks containing all of the stoppers in the iteration through the blocks.

We upper-bound the probability that τ appears by assuming this worst case.

Tetrahedron τ appears in or before round j if all triggers are chosen in or before round j, and no stopper is chosen in or before round $(j-1)$. We have:

$$\Pr[\text{trigger } t \text{ chosen in or before round } j] \leq \sum_{i=0}^{j} \frac{2^i}{n} \leq \frac{2^{j+1}}{n}$$

Hence

$$\Pr[\text{all four triggers chosen in or before round } j] \leq \left(\frac{2^{j+1}}{n}\right)^4$$

Meanwhile, for the stoppers:

$$\Pr[\text{no stopper chosen in or before round } j-1] \leq \Pr[\text{no stopper chosen in round } j-1] = \left(1 - \frac{2^{j-1}}{n}\right)^s$$

Combining these two bounds, and using the inequality $\left(1 - \frac{1}{r}\right)^r \leq (1/e)$,

$$\Pr[\tau \text{ present in round } j] \leq \left(\frac{2^{j+1}}{n}\right)^4 \left(1 - \frac{2^{j-1}}{n}\right)^s \leq \left(\frac{2^{j+1}}{n}\right)^4 \left(\frac{1}{e}\right)^{s\frac{2^{j-1}}{n}}$$

There are $\lg n + 1$ rounds, so:

$$\Pr[\tau \text{ ever appears}] \leq \sum_{j=0}^{\lg n} \left(\frac{2^{j+1}}{n}\right)^4 \left(\frac{1}{e}\right)^{\frac{s}{n}2^{j-1}}$$

$$= \left(\frac{4}{s}\right)^4 \sum_{j=0}^{\lg n} \left(\frac{s}{n}2^{j-1}\right)^4 \left(\frac{1}{e}\right)^{\frac{s}{n}2^{j-1}}$$

The main idea now is to bound this sum with an integral. For convenience, let us define

$$f(j) = \left(\frac{s}{n}2^{j-1}\right)^4 \left(\frac{1}{e}\right)^{\frac{s}{n}2^{j-1}}$$

so that

$$\Pr[\tau \text{ ever appears}] \leq \left(\frac{4}{s}\right)^4 \sum_{j=0}^{\lg n} f(j)$$

Now define

$$x = \frac{s}{n} 2^{j-1}$$

and

$$f(j) = g(x) = x^4 e^{-x}$$

Then

$$dg/dx = 4x^3 e^{-x} - x^4 e^{-x}$$

Setting the derivative equal to zero, we find a minimum of $g(x)$ at $x = 0$ and a maximum of $g(x)$ at $x = 4$. Since x is monotone as a function of j, $f(j)$ has a single maximum at

$$j = \log n - \log s + 3$$

at which $f(j) = 4^4 e^{-4}$. This value of j is not in general an integer. So let $M = \lfloor \log n - \log s + 3 \rfloor$, so that $f(M+1) \leq 4^4 e^{-4}$. We divide the summands into the monotonically increasing part (sum from 0 to M)and the monotonically decreasing part (sum from $M+2$ to $\log n$).

$$\sum_{j=0}^{\lg n} f(j) \leq \sum_{j=0}^{M} f(j) + \sum_{j=M+2}^{\lg n} f(j) + 4^4 e^{-4}$$

Now bounding the monotonic sums with integrals,

$$\sum_{j=0}^{\lg n} f(j) \leq$$

$$\int_{j=0}^{M+1} f(j)\, dj + \int_{j=M+1}^{\lg n} f(j)\, dj + 4^4 e^{-4} \leq$$

$$\int_0^{\infty} f(j)\, dj + 4^4 e^{-4}$$

We restate this in terms of x. Since

$$dx = (\ln 2)\, \frac{s}{n} 2^{j-1} dj = x \ln 2\, dj$$

we get:

$$\int f(j)\, dj = \int \left(\frac{s}{n} 2^{j-1}\right)^4 \left(\frac{1}{e}\right)^{\frac{s}{n} 2^{j-1}} dj =$$

$$\int (x^4 e^{-x}) \frac{dx}{x \ln 2} =$$

$$\frac{1}{\ln 2} \int (x^3 e^{-x})\, dx \qquad (1)$$

Also,

$$\int_0^{\infty} (x^3 e^{-x})\, dx = 6 \qquad (2)$$

So the probability that a tetrahedron τ with s stoppers ever appears is at most

$$\left(\frac{4}{s}\right)^4 \frac{1}{\ln 2} \int_0^{\infty} x^3 e^{-x}\, dx + 4^4 e^{-4} \quad \text{(by 1)}$$

$$\leq \left(\frac{4}{s}\right)^4 \left(\frac{1}{\ln 2} 6 + 4^4 e^{-4}\right) \quad \text{(by 2)}$$

$$= O\left(\frac{1}{s^4}\right)$$

4 Running Time

First, we review the analysis of the usual randomized incremental algorithm for three-dimensional Delaunay triangulation ([6, 9] or see [1, 2]). The running time can be divided into two parts, the time required to find where each new point should be inserted into the Delaunay triangulation (*location time*) and the time required to delete old tetrahedra and create new tetrahedra so as to actually perform the insertion (*update time*). Point location can be done in various ways; the theoretically optimal methods have been shown to be $O(c(n))$, where $c(n)$ is a quantity known as the *total conflict size*. The total conflict size is the sum, over all tetrahedra τ which ever appear in the construction, of the number of stoppers of τ. Total update time is proportional to the total number of tetrahedra which appear over the course of the construction.

In the worst case, the size of a Delaunay triangulation of n points in $\mathbb{R}^3$ is $O(n^2)$, and it turns out this is also the bound on the total conflict size and hence the running time. But in practice the size of the Delaunay triangulation is generally $O(n)$. If we

assume in the "realistic case" that the expected size of the Delaunay triangulation of a random sample of r of the points is $O(r)$ (which also seems to be true [3]), we get a more realistic bound of $O(n \lg n)$ on the total conflict size and the running time. We show that the algorithm remains optimal using a blocked randomized insertion order in the worst case, and is nearly optimal in this "realistic case".

We begin the analysis of the blocked randomized construction by bounding the expected total number of tetrahedra created. Let k_s be the number of possible tetrahedra with s stoppers, out of the $O(n^4)$ total possible tetrahedra, and let K_s be the number of tetrahedra with *at most* s stoppers. Using the result of the previous section, the expected number of tetrahedra which appear is:

$$k_0 + \sum_{s=1}^{n} k_s O\left(\frac{1}{s^4}\right)$$

Clarkson and Shor proved that K_s is at most $f_{n/s}s^4$ as $n/s \to \infty$, where $f_{n/s}$ is the expected number of tetrahedra in a Delaunay triangulation of a randomly chosen subset of n/s points. This gives us a bound of $k_s = O(K_s) = O(n^2 s^2)$ in the worst case, and $k_s = O(K_s) = O(ns^3)$ in the "'realistic" case.

These upper bounds do not, however, completely determine the values of the k_s for all s, which vary from point set to point set. In particular, the total number of possible tetrahedra defined by n points is only $O(n^4)$ and

$$\sum_{s=0}^{n} O(ns^3) = \sum_{s=0}^{n} O(n^2 s^2) = O(n^5)$$

so the K_s provide only loose upper bounds on the corresponding k_s.

We get a tighter upper bound on the expected total number of tetrahedra by finding the values for k_s, for all s, which obey the bound of Clarkson and Shor, but which maximize the upper bound. Note that the bound only holds for s such that $n/s \to \infty$. Let $m = \lfloor n^{1-\delta} \rfloor$ where $0 < \delta < 1$ is some constant. Then we can say that the bound holds for $1 \leq s \leq m$.

We will work through the argument in the "realistic" case first (when $k_0 = O(n)$). In this case Clarkson and Shor showed that $K_s = O(ns^3)$ for $1 \leq s \leq m$. For larger s, we just use the trivial upper bound of $K_s = O(n^4)$. Fixing an appropriate constant c, we define

$$b_s = \begin{cases} 0, \ s = 0 \\ cns^3, \ 1 \leq s \leq m \\ cn^4, \ m+1 \leq s \leq n \end{cases}$$

Now, fixing another constant a, we need to choose k_s for $1 \leq s \leq n$ which maximizes

$$g(k_1, \ldots, k_n) = \sum_{s=1}^{n} \frac{ak_s}{s^4}$$

and such that the constraint

$$K_s = \sum_{i=1}^{s} k_i \leq cns^3 = b_s \tag{3}$$

is satisfied for all $1 \leq s \leq n$ and also

$$K_n = \sum_{i=1}^{n} k_i \leq cn^4. \tag{4}$$

is satisfied.

Intuitively, making k_s as large as possible for the smaller values of s maximizes $g(k_1, \ldots, k_n)$. The following claim formalizes that idea.

Claim 2 *$g(k_1, \ldots, k_n)$ is maximized when*

$$k_s = b_s - b_{s-1}$$

for all $1 \leq s \leq n$

Notice that given the definition of b_s, this means that $k_s = cn(3s^2 - 3s + 1)$ for $1 \leq s \leq m$, $k_{m+1} = c(n^4 - nm^3)$, and $k_s = 0$ for all $m + 1 < s \leq n$.

We prove the claim by induction on s. When $s = 1$, $k_1 = b_1$ maximizes $g(k_1)$. Now suppose the claim is true for $s-1$. We break the possible choices of the k_i into three groups: those depending on whether K_{s-1} is less than, equal to, or greater than b_{s-1}. The case in which $K_{s-1} > b_{s-1}$ violates the constraint. When $K_{s-1} = b_{s-1}$, we see that $g(k_1, \ldots, k_s)$ is maximized when k_s is chosen to be $b_s - b_{s-1}$ and $g(k_1, \ldots, k_{s-1})$ is maximized, which, together with the inductive assumption, satisfies the claim. Finally we consider the case in which $K_{s-1} < b_{s-1}$. Notice that when the claim is satisfied for $k_1, \ldots, k_{s-1}$,

$$K_{s-1} = \sum_{s=1}^{s-1} b_s - b_{s-1} = b_{s-1}$$

Hence, the pigeon hole principle implies that in the current case, when $K_{s-1} < b_{s-1}$, there must exist some $1 \leq j \leq s-1$ such that $k_j \leq b_j - b_{j-1}$. Now we note that

$$g(k_1, \ldots, k_j + 1, \ldots, k_s - 1) =$$

$$g(k_1, \ldots, k_s) + a(\frac{1}{j^4} - \frac{1}{s^4})$$

and hence g cannot be maximized in this case. This establishes the claim.

Now, replacing k_s as defined in Claim 2, we find that the expected total number of tetrahedra is

$$\leq O(n) + \sum_{s=1}^{m} cn(3s^2 - 3s + 1)O\left(\frac{1}{s^4}\right)$$

$$+c(n^4 - m^3)O\left(\frac{1}{m^4}\right)$$

$$\leq O(n) + \sum_{s=1}^{m} O\left(\frac{n}{s^2}\right) + cn^{4\delta}$$

$$= O(n + n^{4\delta})$$

$$= O(n)$$

when $\delta \leq \frac{1}{4}$.

We can use a similar argument to bound the total conflict size and hence the location time. The total conflict size assesses a charge of s for every tetrahedron with s stoppers that appears over the course of the construction, and hence is:

$$c(n) = \sum_{s=1}^{n} k_s O\left(\frac{1}{s^4}\right) \times s$$

Again, $c(n)$ is maximized by assuming that the tetrahedra which appear have as few stoppers as possible, so that the total conflict size is at most

$$\sum_{s=1}^{m} cn(3s^2-3s+1)O\left(\frac{1}{s^3}\right)+c(n^4-m^3)O\left(\frac{1}{m^3}\right)$$

$$\leq \sum_{s=1}^{m} O\left(\frac{n}{s}\right) + cn^{1+3\delta}$$

$$= O(n \lg n + n^{1+3\delta}) = O(n^{1+3\delta})$$

We also get optimal bounds under the quadratic worst-case assumption (when $k_0 = O(n^2)$) about the expected size of the Delaunay triangulation and the values of the k_s. In this case $b_s = cn^2 s^2$. To bound the total number of tetrahedra created, we want to find k_s for $1 \leq s \leq n$ which maximizes

$$g(k_1, \ldots, k_n) = \sum_{s=1}^{n} \frac{ak_s}{s^4}$$

such that

$$K_s = \sum_{i=1}^{s} k_i \leq cn^2 s^2 = b_s$$

for all $1 \leq s \leq m$ and

$$K_n = \sum_{i=1}^{n} k_i \leq cn^4$$

.

A similar proof show that choosing $k_s = b_s - b_{s-1} = cn^2(2s-1)$ for $1 \leq s \leq m$ and $k_{m+1} = c(n^4 - n^2 m^2)$ and $k_s = 0$ for $m+1 < s \leq n$ maximizes g.

Under the worst-case assumption, then, the expected number of tetrahedra ever created is at most

$$O(n^2)+\sum_{s=1}^{m} n^2 c(2s-1)O\left(\frac{1}{s^4}\right)+c(n^4-m^4)O\left(\frac{1}{m^4}\right)$$

$$= O(n^2) + \sum_{s=1}^{n} O\left(\frac{n^2}{s^3}\right) + O(n^{4\delta})$$

$$= O(n^2)$$

when $\delta \leq 1/2$.

and the expected number of total conflict change is at most

$$\sum_{s=1}^{m} n^2 c(2s-1) O\left(\frac{1}{s^3}\right) + c(n^4 - m^4) O\left(\frac{1}{m^3}\right)$$

$$= \sum_{s=1}^{m} O\left(\frac{n^2}{s^2}\right) + O(n^{1+3\delta})$$

$$= O(n^2)$$

when $\delta \leq 1/3$.

5 Experiments

We used Clarkson's `hull` and Shewchuk's `pyramid` to test the effect of our blocked randomized insertion order on the thrashing behavior of a three-dimensional Delaunay triangulation program. We used `hull` because it implements the theoretically optimal randomized incremental algorithm on which our analysis is based. Due to the huge size of its point location structure, history DAG, `hull` begins to thrash relatively early and therefore cannot handle large data. Shewchuk's `pyramid` is a more recent three-dimensional Delaunay triangulation program. It is faster and it can process a lot more data than `hull`, using theoretically non-optimal point location scheme, without history DAG or conflict graph. We want to show that our blocked randomized insertion order still enables it to process a lot bigger data even faster.

While we expected the effect of the increased locality of reference on the performance to be beneficial, it is not easy to predict. A fundamental problem with the algorithm is that insertions in one part of the Delaunay triangulation also affect parts that may be quite distant in three-dimensional space. Moreover, since the Delaunay triangulation is represented by a pointer structure, there is no requirement that adjacent tetrahedra are stored together in virtual memory; this is implementation dependent. Both `hull` and `pyramid` do their own memory management, to avoid making too many calls to `malloc`. Tetrahedra are stored in a list, and in `pyramid` records are freed as tetrahedra are destroyed and reused as new tetrahedra are created, further reducing spatial locality. [1]

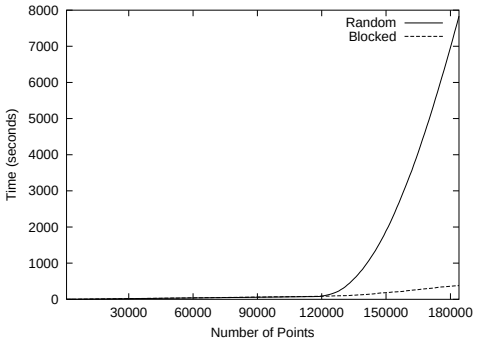

Figure 1: The running time of Hull on MTD data using random order and blocked random order

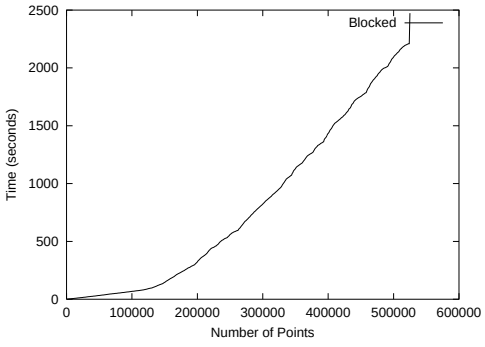

Figure 2: The running time of Hull on B1 data using blocked random order

The data come from two sources - isosurfaces of volumetric data (MTD, B1) and laser range scanner (happy budda). The

[1] We thank Jonathan Shewchuk for pointing out this issue.

MTD dataset (184895 points) are samples from an iso-surface of electron density map of a protein, and the B1 data (525296 points) is obtained by applying one level of butterfly subdivision scheme to MTD to make a denser, bigger data set. The"happy budda" data is from the Stanford 3D scanning repository. We chose the raw scanner data, consisting of 2643633 points with some noise, as better example of typical input to a surface reconstruction computation than the smaller, cleaner, and more evenly distributed vertex set of the completed model. Since we were interested in pushing the limits our our technique, we made larger data sets by duplicating and translating the budda data, making inputs that were the union of two and of four buddas. We divided the data into blocks using a *kd*-tree, stopping when we had 512 blocks for the MTD and B1 data and 4096 blocks for all of the "happy budda" datasets.

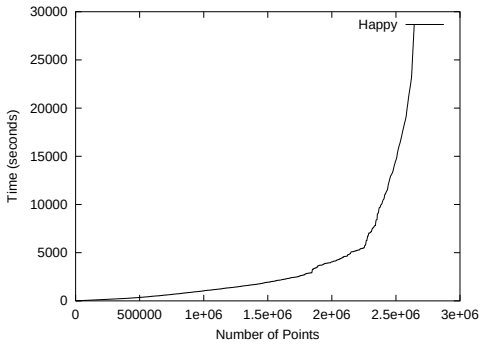

Figure 3: The running time of pyramid for happy budda data using blocked random order

For the `hull` experiment, we used Linux machine using Intel Pentium III (864 MHz) with 511 M RAM with 4 GB of virtual memory. Note that the large virtual memory is essential; the program fails once virtual memory is exceeded.

In Figure 1, the running time for random insertion order and our blocked randomized insertion are shown. We could finish the B1 data using blocked randomized

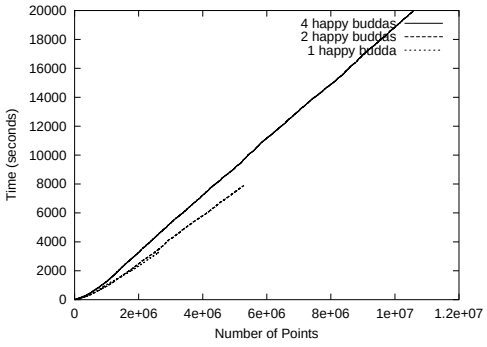

Figure 4: The running time of pyramid on 1,2,4 happy budda data sets using blocked random order and without selecting $n^{1/4}$ random samples for point location

insertion as shown in Figure 2. Though the slope becomes steep around 120000 points where there was a serious thrashing for random order, the blocked order maintains a roughly constant slope and shows a near-linear running time.

For `pyramid`, we used a much less powerful machine, a Sun UltraSPARC 360 MHz CPU with a small 128M physical memory, configured with four gigabytes of local disk as virtual memory.

Using the blocked randomized insertion order, we were able to complete the Delaunay triangulation of the "happy budda", but we found that as the size of the data structure grew, the asymptotic effects of the point location strategy began to dominate the running time. See Figure 3.

Pyramid uses a non-optimal jump-and-walk point location strategy; at the insertion of the i_{th} point p_i, it selects $O(i^{1/4})$ already-inserted points at random, and finds the closest of these to p_i. It then selects either this point, or the last point inserted, whichever is closer to p_i, and begins "walking" in the Delaunay triangulation from there to find the place at which to insert p_i. Using the blocked insertion order, the last point inserted was almost always the closest to p_i, and the expensive search

for a closer point was generally wasted.

Eliminating the $O(i^{1/4})$ search and always starting from the last point inserted gave us an essentially linear running time, as seen in Figure 4. We could complete the Delaunay triangulation of four buddas, over 10 million points. Of course, with this point location strategy the theoretical bounds on the expected running time are very bad.

6 Other applications

Although we give the proofs in terms of the 3D Delaunay triangulation construction, the analysis applies to other similar randomized incremental constructions, in particular the optimal construction of the trapezoidation of a set of non-intersecting segments in the plane [6, 7, 8] (and the similar construction for intersecting segments). This algorithm is practical, and using it on large input sets of segments might be important, for instance in geographic information systems.

Let us refer generically to the objects created in the incremental construction (eg. Delaunay tetrahedra in previous sections) as the *regions*. A drawback of the analysis in this paper is that it depends on every region having the same number of triggers. Thus, although it seems natural to use trapezoids as the regions in an analysis of trapezoidation, we cannot apply this analysis as a trapezoid may have as many as four or as few as two triggers. Fortunately, as pointed out by Mulmuley [7], trapezoidations can be analyzed using *attachments* as the regions. An attachment is the vertical line segment inserted at the endpoint of an input segment; it is defined by the endpoint and the two segments hit by the top and the bottom of the attachment, hence its number of triggers is always three. The stoppers of an attachment are the segments intersecting the attachment.

In general, when the number of triggers is b, the analysis of section 3 implies that the probability that a region with s stoppers appears in the incremental construction is $O(1/s^b)$, in this case $O(1/s^3)$. The bound of Clarkson and Shor on K_s, the number of possible regions with at most s stoppers, is in general $f_{n/s}s^b$. For trapezoidation we get $K_s = O(ns^2)$, since $f_{n/s} = O(n/s)$. Using the argument in section 4 to get a worst-case choice of the k_s, we get a bound of $O(n)$ on the expected total number of attachments and of $O(n^{1+\delta})$ on the expected running time.

For both trapezoidation and Delaunay triangulation, it seems likely that the $O(n^{1+\delta})$ bounds can be improved to $O(n \lg n)$. It would also be nice to find an analysis that handles situations in which the number of triggers can differ, but is upper-bounded by some constant.

We believe that our analysis also applies to randomized incremental algorithms which use *tracing*, such as Seidel's practical $O(n \lg^* n)$ algorithm for trapezoidation of a simple polygon [11], and we are currently working in this direction.

Another important class of randomized incremental algorithms are the LP-type (aka GLP) problems, which optimize an objective function over a set of input regions. Blocked randomized insertion orders may also give optimal algorithms for LP-type problems, although the fact that many LP-type problems have regions which are inherently defined by different numbers of triggers will require a different analysis. In any case, this research direction, while natural, does not seem as pressing since LP-type algorithms do not build large data structures.

On the other hand, the performance of LP-type algorithms can be enhanced in other ways by heuristic insertion orders [15]. Similarly Barber's `qhull` program for arbitrary-dimensional convex hull uses a heuristic insertion order designed to

insert points on the convex hull early [16]. Particular blocked randomized insertion orders, or some other partially-random scheme, might allow these heuristics to be applied while still maintaining optimality.

Finally, we have in no way shown that a blocked randomized insertion order is *guaranteed* to improve the performance of an incremental construction. Theoretical results in this direction would certainly be interesting.

References

[1] K. Mulmuley. Computational Geometry: An Introduction Through Randomized Algorithms. Prentice Hall, New York, 1993.

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

[3] S. Choi and N. Amenta. Delaunay triangulation programs on surface data, The 13th ACM-SIAM Symposium on Discrete Algorithms, 2002.

[4] R. Seidel. Backwards analysis of of randomized geometric algorithms. In J. Pach, editor, *New Trends in Discrete and Computational Geometry*, Pages 37-68, Springer-Verlag, Berlin, 1993.

[5] K. Mulmuley and O. Schwarzkopf. *Randomized Algorithms*, Chapter 34 in "Handbook of Discrete and Computational Geometry", J. E. Goodman and J. O'Rourke, eds. CRC Press, 1997.

[6] K.L. Clarkson, and P.W. Shor, Applications of random sampling in computational geometry, II. Discr. and Comp. Geometry 4 (1989), pp. 387–421.

[7] K. Mulmuley. *A Fast Planar Partition Algorithm, I*, Journal of Symbolic Computation, (1990) 10, 253-280.

[8] K. Mulmuley. A Fast Planar Partition Algorithm, II. Journal of the ACM, 38(1):74-103, January 1991

[9] K. L.Clarkson, K.Mehlhorn, and R.Seidel. Four results on randomized incremental constructions. Comp. Geom.: Theory and Applications, pages 185–121, 1993.

[10] R. Seidel. Small-dimensional linear programming and convex hulls made easy. Discr. and Comp. Geometry 6 (1991), pp. 423–434.

[11] R. Seidel. A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. Comput. Geom. Theory Appl., 1:51–64, 1991.

[12] L. P. Chew. Building voronoi diagrams for convex polygons in linear expected time. CS Tech Report TR90-147, Dartmouth College, 1986.

[13] G. Blelloch, J.Hardwick, G.Miller, and D.Talmor. Design and Implementation of a Practical Parallel Delaunay Algorithm. Algorithmica, 24(3/4), 1999.

[14] P. K. Agarwal, M. de Berg, J. Matousek, and O. Schwarzkopf. Constructing levels in arrangements and higher order Voronoi diagrams. SIAM J. Comput., 27:654– 667, 1998.

[15] E. Welzl. *Smallest enclosing disks (balls and ellipsoids)*, in New Results and New Trends in Computer Science, (H. Maurer, ed.), Lecture Notes in Computer Science 555 (1991) 359–370.

[16] C. B. Barber, D. Dobkin, and H. Huhdanpaa, The quickhull algorithm

for convex hulls, ACM Trans. Math. Software 22(1996), 469-483.

[17] O. Devillers and P. Guigue. *The shuffling buffer*, International Journal of Computational Geometry and its Applications, 11:5, pp 555–572, (2001).