Fundamenta Informaticae XI (1988) 275-288
North-Holland

CONSTRUCTING DELAUNAY TRIANGULATIONS BY MERGING BUCKETS IN QUADTREE ORDER

Jyrki KATAJAINEN and Markku KOPPINEN

Department of Mathematical Sciences, University of Turku
SF-20500 Turku, Finland

Accepted April 1988

ABSTRACT. Recently Rex Dwyer [D87] presented an algorithm which constructs a Delaunay triangulation for a planar set of N sites in $O(N \log\log N)$ expected time and $O(N \log N)$ worst-case time. We show that a slight modification of his algorithm preserves the worst-case running time, but has only $O(N)$ average running time. The method is a hybrid which combines the cell technique with the divide-and-conquer algorithm of Guibas & Stolfi [GS85]. First a square grid of size about $\sqrt{N}$ by $\sqrt{N}$ is placed on the set of sites. The grid forms about N cells (buckets), each of which is implemented as a list of the sites which fall into the corresponding square of the grid. A Delaunay triangulation of the generally rather few sites within each cell is constructed with the Guibas & Stolfi algorithm. Then the triangulations are merged, four by four, in a quadtree-like order.

AMS SUBJECT CLASSIFICATIONS: 51M15, 68P05, 68Q25, 68U05.

ADDITIONAL KEY WORDS AND PHRASES: Voronoi diagrams, Delaunay triangulations, bucket methods, quadtrees.

1. INTRODUCTION

Let S be a set of N points $s_1, s_2, \ldots, s_N$, called sites, in the Euclidean plane. The *proximal polygon* $V(s_i)$ of site s_i consists of all points of the plane having s_i as a nearest site in the set S. The union of the boundaries of the polygons $V(s_i)$, i=1,2,...,N, is a graph that is called the *Voronoi diagram* of S. The dual graph, with the sites as the vertices and straight line segments as edges, is the *Delaunay diagram* of S. By adding, if necessary, some new edges, this diagram can be made into a triangulation, called a *Delaunay triangulation* of S and denoted DT(S). If S does not contain any four cocircular sites the Delaunay diagram and triangulation coincide.

Voronoi diagrams and Delaunay triangulations are among the most fundamental geometrical structures, having numerous applications in various disciplines. For these the reader is referred to [LS80], [AB83], and [PS85], as well as the references therein. The aim of this paper is to present and analyse a fast method for constructing a Delaunay triangulation.

Many algorithms for making the construction in $O(N \log N)$ worst-case time are known (see for example [S78], [LS80], [GS85], [PS85], [F87]). All of these are optimal at least in those models of computation where sorting requires $\Omega(N \log N)$ time. Also various algorithms running in optimal $O(N)$ expected time have been presented (see for example [BWY80], [M84], [OIM84]). Now it is natural to ask whether these objectives could be combined, that is, whether there exists an algorithm

---

* Present address: Department of Computer and Information Science, Linköping University, S-581 83 Linköping, SWEDEN. This author's research was supported by the Academy of Finland.

which is optimal in both the worst and also in the average case. One such algorithm is easily achieved by running parallel two algorithms, one from each class, or by executing the instructions of the algorithms alternately. A more practical way of combining two algorithms will be found in [BWY80] (see below).

Most expected-time efficient geometrical algorithms are based on the *cell technique* (also called the bucketing technique), although some other algorithm paradigms might also be useful for this purpose (see [D85]). In the cell technique the smallest closed square that covers the data is partitioned into about N equal-sized squares (cells), and within each cell the data are kept in a chain. To determine all the cell memberships takes $O(N)$ time if, as is assumed in the sequel, the model of computation is *real-RAM* (see [PS85]), where real numbers can be multiplied, divided, added, subtracted, and truncated in constant time. The cell structure is usually needed to achieve fast access to the near neighbours of a site.

Bentley, Weide & Yao [BWY80] utilize the cell structure by performing a spiral search around each site, in order to find the Delaunay edges emanating from it. If the search happens to take "too long" for some site, then one switches to an optimal worst-case algorithm for the whole set of sites. The average-case running time is $O(N)$. The worst case is not analysed in [BWY80], but an easy modification of the algorithm guarantees an upper bound $O(N \log N)$.

The cell technique is applied in very different ways in the algorithms of Ohya, Iri & Murota [OIM84] and that of Maus [M84]. In both cases, $O(N)$ expected-time and $O(N^2)$ worst-case running time are achieved.

Dwyer [D87] presented yet another efficient expected-time algorithm, which uses $O(N \log\log N)$ time in the average case and $O(N \log N)$ time in the worst case. The square enclosing the sites is partitioned into N/log N cells, the triangulation of the sites within each cell is constructed, and then the triangulations are merged to form the triangulation of the entire set of sites. The triangulation of the cells and their merging is performed by the algorithm of Guibas & Stolfi [GS85]. The merging order is as follows: first the triangulations within each row of cells are merged in pairs and then the row triangulations are merged in pairs. In particular, merging of very narrow rows is avoided; such rows cause the ordinary divide-and-conquer algorithm to take $\Omega(N \log N)$ time on the average, as was shown by Ohya, Iri & Murota [OIM84].

We show that a rather similar algorithm runs in $O(N)$ expected time and $O(N \log N)$ worst-case time. Our method differs from Dwyer's in two respects: (1) the square is partitioned into about N cells and (2) the merging of the subtriangulations is done in a quadtree-like order.

More precisely, the merging order is the following. Assume that the enclosing square U is divided into 4^k cells. Imagine a complete quaternary tree of depth k, in which a node is represented by some subsquare U_1 of U and its children are obtained by splitting U_1 into four smaller squares. The root is the whole square U and the leaves are the cells. The merging proceeds from the leaves to the root by constructing the triangulation of each internal node from those of its four children. In an actual implementation, the merging order can, for example, follow the logic of the pyramid data structure or the Morton matrix (see [S84]). Then only a part of the whole quadtree is to be kept simultaneously in the memory. The idea of merging the subtriangulations in quadtree order was

inspired by the work of Ohya, Iri & Murota [OIM84], where the quadtree was, however, used for a different purpose.

Our algorithm is introduced more formally in Section 2. The worst-case and average-case running times of the algorithm are analysed in Section 3; in particular, Section 3.2 contains some apparently new facts about triangulations that could be of independent interest. The results obtained by experiments in which our algorithm was compared to previous ones are reported in Section 4. Finally, some concluding remarks are presented in Section 5.

2. THE ALGORITHM

Let S be a set of N points in the Euclidean plane. Fix an integer α and let $K=\max\{4^{\lfloor \log_4 N \rfloor - \alpha}, 1\}$.

Our algorithm for triangulating S consists of three major steps. First, the set S is enclosed in a square, which is then divided into K squares (called cells) of equal size, and the sites are inserted into the cells. Second, Delaunay triangulations of the cells are constructed with a suitable algorithm. Third, these are merged in quadtree (or pyramid) order.

The significance of the parameter α is that N/K, the average number of sites in a cell, belongs to $[4^{\alpha},4^{\alpha+1})$, unless $\alpha > \lfloor \log_4 N \rfloor$, in which case K=1. Usually α is chosen to be a small non-negative integer, such as 0,1, or 2; the optimal value depends on the implementation.

The algorithm is shown below in greater detail. There the cells are indexed C_{ij}, $i,j=0,\ldots,\sqrt{K}-1$, in the natural way. They are triangulated with the divide-and-conquer algorithm of Guibas & Stolfi [GS85]; this guarantees $O(N \log N)$ running time even when almost all sites belong to a single cell. In practice, the sets within the cells are normally small, and thus their triangulations could perhaps be constructed faster with some simpler algorithm; one could even choose the algorithm separately for each cell according to the number of sites in it.

```
procedure Pyramid_DT(S: siteset) returns(triangulation)
{Assume that S={s1,s2,...,sN} and K=max{4^⌊log4N⌋-α,1}, where α is a constant.}
{Step 0: Distribute the sites into cells}
    Determine the smallest square that contains the sites of S.
    Partition the square into K cells Cij. Initially each cell is empty.
    for each s∈S do
        Insert s into the cell Cij where it falls.
{Step 1: Triangulate the cells}
    for i:=0 to √K-1 do
        for j:=0 to √K-1 do
            DTij := Guibas&Stolfi_DT(Cij)
{Step 2: Merge the triangulations}
    for h:= log4K downto 1 do
        for i:=0 step 2 to 2^(h-1) do
            for j:=0 step 2 to 2^(h-1) do
                T1:= Merge(DTij, DTi+1,j);
                T2:= Merge(DTi,j+1, DTi+1,j+1)
                DTi/2,j/2:= Merge(T1,T2)
    return(DT00)
end Pyramid_DT.
```

The *Merge* procedure is basically the usual one, which runs as follows (for details, see [GS85]): Let DT(L) and DT(R) be the triangulations to be merged, lying to the left and to the right of a separating line. To compute DT(L∪R), we first search for the lower common tangent of L and R. The search begins with a rightmost site of L and a leftmost site of R, and advances downwards along the convex hulls of L and R until the tangent is encountered. This is an edge of DT(L∪R). Then, with a scan upwards, the other new edges crossing the line are determined. During the scan, some of the old edges in DT(L) or DT(R) are removed.

For the algorithm *Pyramid_DT*, the *Merge* procedure must be modified to handle also the special cases where a triangulation to be merged is empty or singleton.

Furthermore, since the merges are performed in two directions, it is good to maintain for each square, in addition to the leftmost and rightmost sites, also the uppermost and lowermost sites. This will make the search for common tangents more efficient. On the other hand, it is quite sufficient to maintain only one boundary site for each square, and, as the subsequent analysis shows, the time complexity of the algorithm is still asymptotically the same.

In the following section we prove that the algorithm has $O(N \log N)$ overall time, and $O(N)$ expected running time for a quasi-uniform distribution. Let us, however, first give a quick intuitive explanation of the $O(N)$ expected time when the sites are drawn from a uniform distribution. Insertion of the sites into the cells (Step 0) takes linear time, and since the number of sites in a cell is obviously approximately constant, the triangulation of the cells (Step 1) also takes linear time. As for the merging step (Step 2), consider the "highest level" partition of the square into four smaller squares, each of which contains about N/4 sites. When these squares are merged, only sites in the border regions are likely to be affected. Hence the running time seems to follow the recurrence relation $T(N) = 4\,T(N/4) + O(\sqrt{N})$, and the solution to this is $O(N)$.

The storage requirements of our algorithm are the same as those for the algorithm of Guibas & Stolfi, except that some extra storage is needed for the cell structure. An entry in the cell directory contains either a pointer to a list of sites or a pointer to a boundary site of a subtriangulation. It is possible to implement the site lists inside an array of size N, because they are mutually disjoint. Hence a total of K+N extra memory slots are required (plus K bits to indicate whether a cell contains a site list or a triangulation). When the cells have been merged, there are only K/4 subtriangulations left. Thereafter one can start maintaining pointers to four boundary sites for each square, as suggested above, and this does not lead to any further memory requirements.

There are at least two ways of reducing the extra storage needed. The array for the site lists can be removed if reordering of the input sites is allowed. Then the site lists for the cells can be implemented sequentially in the input arrays, as has been pointed out by Maus [M84]. Let us rename the cells $C_1,\ldots,C_K$. If *index*(C_i) denotes the position for the first site of the cell C_i, the sites in C_i are those between *index*(C_i) and *index*(C_{i+1}), where *index*(C_K)=N+1. Free storage is of course needed for the reordering of the sites, but this is of little concern because one can use the storage required later for the edges of the Delaunay triangulation. Hence, if the input is reordered, the cell structure requires only K memory slots.

Observe that $K \leq N/4^{\alpha}$. This means that by choosing a large value for the constant α, further storage space can be saved. This will, however, make the algorithm slower (cf. Section 4).

Finally, it should be observed that there exists another efficient merging order, the so-called Morton order (see [S84, Section, 2.3]), which could be used as well. A recursive top-down implementation for the merge is shown below. The *Morton_merge* procedure allows us to choose the parameter K more freely; it need not be a power of 4, and yet the procedure runs correctly. After creating the cell structure, the procedure is called with parameters (a,b,c,d)=(0,$\sqrt{K}$,0,$\sqrt{K}$).

```
procedure Morton_merge(C: cell_structure; a,b,c,d: integer) returns(triangulation)
{This procedure triangulates the union of the cells C_ij with a ≤ i < b and c≤j<d.}
if a=b-1 and c=d-1 then return(Guibas&Stolfi_DT(C_ac))
elseif d-c > b-a then
    T1:= Morton_merge(C,a,b,c, ⌊(c+d)/2⌋ )
    T2:= Morton_merge(C,a,b, ⌊(c+d)/2⌋ ,d)
    return(Merge(T1,T2))
elseif d-c ≤ b-a then
    T1:= Morton_merge(C,a, ⌊(a+b)/2⌋ ,c,d)
    T2:= Morton_merge(C, ⌊(a+b)/2⌋ ,b,c,d)
    return(Merge(T1,T2))
end Morton_merge.
```

3. ANALYSIS OF TIME COMPLEXITY

In this section we prove the following two theorems, the first of which implies that the running time of *Pyramid_DT* is $\Theta(N \log N)$.

THEOREM 3.1. a) The algorithm *Pyramid_DT* takes $O(N \log N)$ time. b) Both the triangulation step (Step 1) and the merging step (Step 2) demand $\Omega(N \log N)$ time, and these two lower bounds can be proved by using the same critical site sets.

In the second theorem we assume that the sites are drawn independently from a fixed distribution in the unit square U with density function f. Following Dwyer [D87] we only consider cases where f is *quasi-uniform in* U *with bounds* c_1,c_2, by which we mean that f=0 outside U and that inside U we have $c_1 \le f(x,y) \le c_2$ where c_1,c_2 are positive constants.

THEOREM 3.2. Assume that the density function f is quasi-uniform with bounds c_1,c_2. Then the expected running time of the algorithm *Pyramid_DT* is at most $k(c_1,c_2)N$, where the coefficient depends on c_1 and c_2 but not on N.

These results are valid even if the merges are perfomed in the Morton order. This is so because the *Morton_merge* procedure involves exactly the same *Merge* operations as the algorithm *Pyramid_DT*, only in a different order.

3.1. WORST-CASE COMPLEXITY

Proof of Theorem 3.1a. We follow here the lines of the proof of [D87, Theorem 3.1]. Step 0 of the algorithm obviously takes $O(N)$ time, because the floor function is assumed to be a constant time operation.

Suppose that the cells are triangulated in Step 1 with an algorithm that takes at most time $T_1(q)$ for a set of q sites, where $F(q)=T_1(q)/q$ is a non-decreasing function. If the cells contain $n_1,\ldots,n_K$ sites, their triangulation time is bounded by

$$\sum_{i=1}^{K} T_1(q) = \sum_{i=1}^{K} n_i F(n_i) \le \sum_{i=1}^{K} n_i F(N) = T_1(N),$$

i.e. $O(N \log N)$, when we use an optimal divide-and-conquer algorithm.

The *Merge* operations of Step 2 are performed at $\log_4 K = O(\log N)$ levels. At each level a site is involved in at most two merges, one vertical and one horizontal. A planar graph on N vertices always contains less than 3N edges. Hence the total number of edges created or deleted at one level is less than $2 \times 2 \times 3N$. Also the total number of sites visited during the searches for common tangents at one level is bounded by $2 \times N$. To sum up, Step 2 takes $O(N \log N)$ time. □

Proof of Theorem 3.1b. Step 1 obviously requires time $\Omega(N \log N)$ in the worst case, i.e. in the case when almost all the sites fall into a single cell. It is perhaps more surprising that a similar result holds for Step 2. The theorem follows when we construct, for each $N \ge N_0$, a set of N sites, such that the total number of edges created in the course of the merging is at least $A N \log N$, and such that at least $N - B \log N$ sites are in a single cell. Here N_0, A, and B are constants.

Draw a spiral arc in the unit square U so that it passes through the origo and runs approximately as shown in Figure 1. The spiral could, for example, be logarithmic.

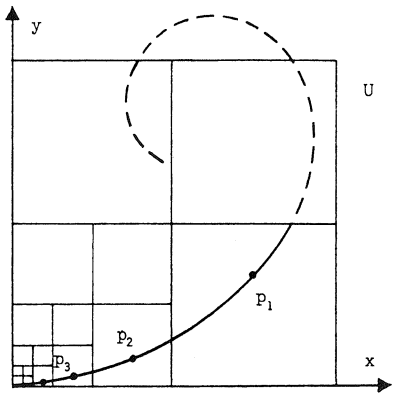

FIGURE 1
A worst-case site set.

We choose the sites $p_1,\ldots,p_N$ from the arc so that the x-coordinate of p_i is $3/2^{i+1}$. When $K=4^k$ is the number of cells, the sites $p_1,\ldots,p_k$ belong to different cells, whereas $p_{k+1},\ldots,p_N$ are all in the bottom left corner cell.

The idea in using the spiral is in that for $a < b < c < d$ the site p_a is just inside the circle through p_b, p_c, and p_d. Consequently, the Delaunay triangulation of a set $\{p_i,\ldots,p_N\}$ consists of the edges (p_j,p_{j+1}), $i \le j < N$ "along" the arc, and of the edges (p_i,p_j), $i+2 \le j \le N$ that connect p_i to the other sites. Therefore, insertion of p_{i-1} into the triangulation means deletion of the latter set of edges and creation of new edges connecting p_{i-1} to $p_i,\ldots,p_N$.

The merging step starts with the corner cell triangulated, and at each merging level a new site p_i is inserted into the triangulation. The total number of edges created during the merges is

$(N-k)+\ldots+(N-1) = k(N-(k+1)/2)$. Here $k=\log_4 K = \lfloor \log_4 N \rfloor - \alpha$. Since α is a constant, the theorem follows. □

Observe, incidentally, that the site set in the proof, with the same insertion order of sites, is also very difficult for an incremental Delaunay triangulation algorithm (see for example [GS77], [L77], [GS85]), proving the lower bound $\Omega(N^2)$.

It is interesting to see why the above argument does not contradict Theorem 3.2. Suppose that the site set was obtained from a distribution. To carry the argument through, the set must be concentrated close enough to the spiral and the corner cell must contain a sufficient number of sites. In particular, the density function must have a peak near the origo, and since the cell becomes smaller when N grows, the peak must become ever higher and narrower. Thus the distribution must vary with N, whereas in Theorem 3.2 it is assumed to be fixed.

3.2. General triangulations and finished sites

We discuss in this section some general features of triangulations that will be used later in connection with Delaunay triangulations. Recall that a *triangulation* of a set S of sites in the Euclidean plane is a straight-line planar graph on S, with a maximal set of edges. That is, the sites of S are joined by non-intersecting straight-line segments so that every face internal to the convex hull of S is a triangle. Observe also that the trivial cases 0,1, or 2 sites, or more sites on the same line, are regarded as triangulations.

DEFINITION 3.3. Let $T(S_1)$, $T(S_2)$ be triangulations of the sets S_1, S_2. We use the notation $T(S_1) < T(S_2)$ if $S_1 \subseteq S_2$ and if $T(S_1)$ contains each edge of $T(S_2)$ whose endpoints belong to S_1.

This gives a partial order for the set of triangulations. Note that $T(S_1) < T(S_2)$ does not imply that the edges of $T(S_1)$ belong to $T(S_2)$. Note also that $DT(S_1) < DT(S_2)$ is equivalent to $S_1 \subseteq S_2$ if S_1 has a unique Delaunay triangulation.

DEFINITION 3.4. Let $T(S_1) < T(S_2)$ be two triangulations and $s \in S_1$. We say that s is *finished in* $T(S_1)$ *relative to* $T(S_2)$ if the sets of edges emanating from s in $T(S_1)$ and $T(S_2)$ coincide, otherwise it is *unfinished.*

PROPOSITION 3.5. Let $T(S_1) < T(S_2)$ be two triangulations. A site $s \in S_1$ is finished in $T(S_1)$ relative to $T(S_2)$ if and only if S_1 contains the endpoints of all the edges emanating from s in $T(S_2)$.

Proof. Let $(s,p_1),\ldots,(s,p_m)$ be the edges emanating from s in $T(S_2)$. If s is finished, then the p_i's belong to S_1 by definition.

Conversely, assume that the p_i's belong to S_1 but that s is nevertheless unfinished. Then $(s,p_1),\ldots,(s,p_m)$ belong to $T(S_1)$ because $T(S_1) < T(S_2)$, but $T(S_1)$ contains yet another edge (s,r). Since the region covered by $T(S_2)$ is convex, it contains the entire edge (s,r). Hence $(s,r)\backslash\{s\}$ either intersects one of the edges (s,p_i) or one of the open triangles of $T(S_2)$ with s as a vertex. The first alternative is clearly impossible. The particular triangle contains no sites of S_2, and especially not r. Therefore (s,r) intersects the opposite side (p_i,p_j). But even this is impossible, because (p_i,p_j) belongs also to $T(S_1)$, which is a triangulation. □

COROLLARY 3.6. Let $T(S_1) < T(S_2) < T(S_3)$ be three triangulations. A site $s \in S_1$ is finished in $T(S_1)$ relative to $T(S_3)$ if and only if it is finished both in $T(S_1)$ relative to $T(S_2)$ and in $T(S_2)$ relative to $T(S_3)$.

Proof. The latter condition implies the former by Definition 3.4.

Conversely, assume that s is finished in $T(S_1)$ relative to $T(S_3)$. If (s,q) is an edge in $T(S_3)$ then $q \in S_1$; then also $q \in S_2$, and using 3.5 we obtain that s is finished in $T(S_2)$ relative to $T(S_3)$. Finally, by 3.4, s is also finished in $T(S_1)$ relative to $T(S_2)$. □

3.3. AVERAGE-CASE COMPLEXITY

Our next aim is to prove that the expected running time of the algorithm *Pyramid_DT* is linear. For this purpose, we need two lemmas. In first we treat the complexity of Step 1 in which the cells are triangulated.

LEMMA 3.7. Assume that the density function f is quasi-uniform, with bounds c_1,c_2. The expected running time of Step 1 is at most $k(c_2)N$, even if an exponential algorithm is used for the triangulation of the cells. Here $k(c_2)$ does not depend on N.

Proof. Let $T_1(q)$ be the running time of the algorithm for a set of q sites. Enumerate the cells $C_1,\ldots,C_K$. The expected value of the total running time T of Step 1 is

$$E(T) = \sum_{i=1}^{K} \sum_{q=0}^{N} P(|S \cap C_i| = q) \cdot T_1(q).$$

Write

$$m_i = \int_{C_i} f \leq \int_{C_i} c_2 = c_2/K.$$

Then, because the sites are independently drawn from the distribution,

$$P(|S \cap C_i| = q) = \binom{N}{q} m_i^q (1-m_i)^{N-q}.$$

Inserting further $T_1(q) \leq c\, e^q$ we obtain from the binomial formula

$$E(T) = c \sum_{i=1}^{K} (1+(e-1)m_i)^N \leq cK(1+(e-1)c_2/K)^N.$$

Here $N/4^{\alpha+1} < K \leq N/4^{\alpha}$, so

$$\begin{aligned} E(T) &\leq 4^{-\alpha}\, cN(1+4^{\alpha+1}(e-1)c_2/N)^N \\ &< 4^{-\alpha}\, cN \exp(4^{\alpha+1}(e-1)c_2). \end{aligned}$$ □

REMARK. The more realistic assumption $T_1(q) \leq cq^2$ would have yielded

$$E(T) < c\, c_2 (4^{\alpha+1} c_2 + 1)N.$$

We shall estimate the work done in Step 2 by calculating the number of unfinished sites at the various merging levels. There we shall use the corollary to the following lemma.

LEMMA 3.8. Assume that the density function f is quasi-uniform with bounds c_1,c_2. Let $U_1 \subseteq U$ be a rectangle and $DT(S \cap U_1) < DT(S)$. If $s \in S \cap U_1$ is a site whose minimum distance from the boundary of U_1 is t, then the probability that s is unfinished in $DT(S \cap U_1)$ relative to DT(S) is at most $16(1-c_1\pi t^2/32)^{N-1}$.

Proof. Denote by C_1 the condition that s is unfinished in $DT(S \cap U_1)$ relative to DT(S). Then, by 3.5 $C_1 \Rightarrow C_2$, where C_2 is the condition that DT(S) contains an edge (s,q) whose length is greater than t. Consider a circle with centre s and radius $t/\sqrt{2}$. Divide it into 16 open sectors $A_1,\ldots,A_{16}$ with

angles $\pi/8$. Let C_3 be the condition that at least one of the open sectors is site-free. Assuming for a moment that $C_2 \Rightarrow C_3$ we obtain

$$P(C_1) \le P(C_3) \le \sum_{i=1}^{16} P(S \cap A_i = \emptyset)$$
$$= \sum_{i=1}^{16} (1 - \int_{A_i} f)^{N-1} \le 16(1 - c_1 \pi t^2/32)^{N-1},$$

where we used the fact that $f(x,y) \ge c_1$.

It remains to be shown that $C_2 \Rightarrow C_3$. Assume C_2, and consider the square with a diagonal along (s,q), s as one vertex, and side length $t/\sqrt{2}$ (see Figure 2). The diagonal divides the square into two triangles. Since (s,q) is in DT(S), there is a site-free circle passing through s and q. By [D87, Lemma 5.3] the circle contains at least one of the two triangles, and hence also one of the sectors A_i. So this A_i is site-free. □

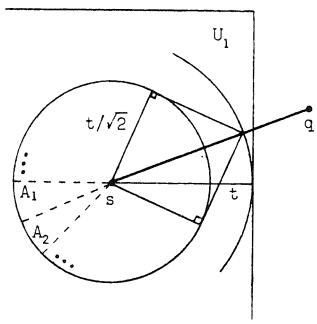

FIGURE 2
Illustration to the proof of Lemma 3.8.

COROLLARY 3.9. If the situation is as in the above lemma and if U_1 has sides a and b, then the expected number E(a,b) of unfinished sites in $DT(S \cap U_1)$ relative to DT(S) is at most $103(c_2/\sqrt{c_1})(a+b)\sqrt{N}$.

Proof. Assume $a \le b$. Let $0 \le t < a/2$. Those points of U, for which the minimum distance from the boundary of the rectangle U_1 is in the infinitesimal interval (t,t+dt), form a certain region A_t of area 2(a+b-4t)dt. The probability that a given site belongs to A_t is

$$\int_{A_t} f \le c_2\, 2(a+b-4t)dt \le 2c_2(a+b)\,dt.$$

Using Lemma 3.8 we can now estimate

$$E(a,b) \le N \cdot \int_0^{a/2} 16(1 - c_1\pi t^2/32)^{N-1} \cdot 2c_2(a+b)dt$$
$$= 32c_2(a+b)N\sqrt{32/(\pi c_1)} \int_0^{a\sqrt{\pi c_1/128}} (1-x^2)^{N-1}dx,$$

where $x = t\sqrt{\pi c_1/32}$. Denote the integral by I. Since $a < 1$ and $c_1 \le 1$,

$$I < \int_0^1 (1-x^2)^{N-1}dx = \frac{2\cdot 4\cdot 6\cdots(2N-2)}{1\cdot 3\cdot 5\cdots(2N-1)} \le \frac{1}{\sqrt{N}}.$$

Hence

$$E(a,b) < 103(c_2/\sqrt{c_1})(a+b)\sqrt{N}. \square$$

Proof of Theorem 3.2. Clearly Step 0 of the algorithm *Pyramid_DT*, which creates the cells and their site lists, can be performed in linear time. Step 1, triangulating the cells, has been concluded in Lemma 3.7.

Step 2, merging the triangulations, consists of $\log_4 K$ levels, and each level is started with the square U partitioned into smaller squares for which the triangulations are already constructed. A level can be thought to consist of two phases: first each small square is merged with one of its neighbours in the direction of the x-axis, and this results in rectangles with the ratio of sides 1:2, then each rectangle is merged with one of its neighbours in the direction of the y-axis, which leaves us again with a partition of U into squares.

Consider one such phase. Let U_1 be one small rectangle with triangulation $DT(S\cap U_1)$ to be merged with its neighbour, and let U_2 be the resulting larger rectangle with triangulation $DT(S\cap U_2)$. Then $DT(S\cap U_1) < DT(S\cap U_2)$, since the *Merge* procedure does not create any new edges with both endpoints in U_1 (for details, see the algorithm in [GS85]). Similarly $DT(S\cap U_2) < DT(S)$, where DT(S) is the final triangulation. The sites of U_1 that are affected (i.e. receive or lose edges) are precisely those that are unfinished in $DT(S\cap U_1)$ relative to $DT(S\cap U_2)$, and by Corollary 3.6 these are all among those sites that are unfinished in $DT(S\cap U_1)$ relative to DT(S).

We conclude that, at each phase, an upper bound of the total number in U of affected sites is obtained by estimating the total number in U of unfinished sites relative to DT(S). The crucial point here is that a site which once becomes finished relative to DT(S) is never again touched upon in the subsequent merges (see Corollary 3.6).

Corollary 3.9 gives us the estimation for one rectangle. Next we put all the levels together. Let $K=4^k$ and enumerate the levels $1,\ldots,k$. When we begin merging at level p, the small squares have sides $a=b=2^{p-k-1}$, and after the first phase the rectangles have sides $a=2^{p-k-1}$, $b=2^{p-k}$. Now we get an upper bound for the expected value of the total number of unfinished sites during the various merging levels:

$$\sum_{p=1}^{k} (4^{k-p+1}\cdot E(2^{p-k-1},2^{p-k-1}) + 2\cdot 4^{k-p}\cdot E(2^{p-k-1},2^{p-k}))$$

$$< 7\cdot 2^k\cdot 103(c_2/\sqrt{c_1})\sqrt{N} < 721(c_2/\sqrt{c_1})\, 2^{-\alpha} N,$$

where we used the fact that $2^k=\sqrt{K} \le \sqrt{N}/2^\alpha$. Finally, multiplying this by 6 we obtain an upper bound for the expected value of the total number of edges created or deteted during Step 2 (see [D87, Corollary 2.2]). Hence the expected work here is $O(N)$.

We still have to estimate the work done in the searches for common tangents. Consider the merging of a small rectangle U_1 with one of its neighbours. The sites in $S\cap U_1$ that are visited

during the search for a common tangent are all on the convex hull of $S \cap U_1$. Consider the situation in the proof of 3.8. If the site s therein is on the convex hull of $S \cap U_1$, then one of the 16 sectors around it must be site-free. Hence all such sites are among those that were counted as unfinished in the estimation of 3.9. Therefore the same argument as above shows that the expected work done in the searches is $O(N)$. □

REMARK. If, for example, $c_1=c_2=1$, we have $2163 \cdot 2^{-\alpha} N$ as an upper bound for the number of edges created during Step 2. The large coefficient is due to our generous estimations at several points but our experiments suggest that its value is 3.5 if $\alpha=0$ and 5 if $\alpha>0$.

4. EXPERIMENTAL RESULTS

We made experimental comparisons of our algorithm with three other Delaunay triangulation algorithms: Dwyer's algorithm (the variation in [D87, Section 6]), the divide-and-conquer algorithm and the simple incremental algorithm described in [GS85]. The implementations of the first two algorithms, as well as those of the underlying quad-edge data structure of [GS85], were made by Dwyer & Webb (cf. [D87]).

We implemented our algorithm *Pyramid_DT* essentially in the form described in Section 2, except that we used the incremental algorithm to triangulate the cells. In [GS85] the code for the incremental algorithm was given by assuming that a new site to be inserted is always inside the convex hull of the previous sites. Special cases, where a new site happens to be outside the convex hull, were handled in the manner proposed by Shapiro [S81].

Since we used the quad-edge data structure in every algorithm, we decided that the number of *Splice* operations (see [GS85]) would serve as a good measure for the work done in the programs: it is the splices that make up the bulk of the work in the graph manipulating operations, such as adding, deleting, or swapping an edge. Addition of an edge requires two *Splice* operations, deletion also two, and a swap four.

Our results are summarized in Figure 3. The input sites were drawn from the uniform distribution in the unit square. Five inputs of size 2^k or 4^k-1 were generated for $2^4 \le N \le 2^{15}$, and the average number of splices per N performed by the different algorithms was calculated. The pyramid algorithm was run for $\alpha=0,1,2$. The sawtooth structure of the curves results from the fact that when N reaches a power of 4, one merging level is added, i.e. the cells are split into four; the shapes of the teeth are mainly caused by the incremental algorithm that was used to triangulate the cells.

We also measured the actual running times of the programs. As one might expect, the curves obtained are very similar to those in Figure 3, with two main exceptions. First, the curves of Dwyer's algorithm and those of the pyramid algorithm for $\alpha=0$ are closer to each other. Second, the curve of the incremental algorithm is convex rather than concave. This is due to the fact that the simple *Locate* operation (see [GS85]) becomes slow for large N, and this is not apparent from the number of splices.

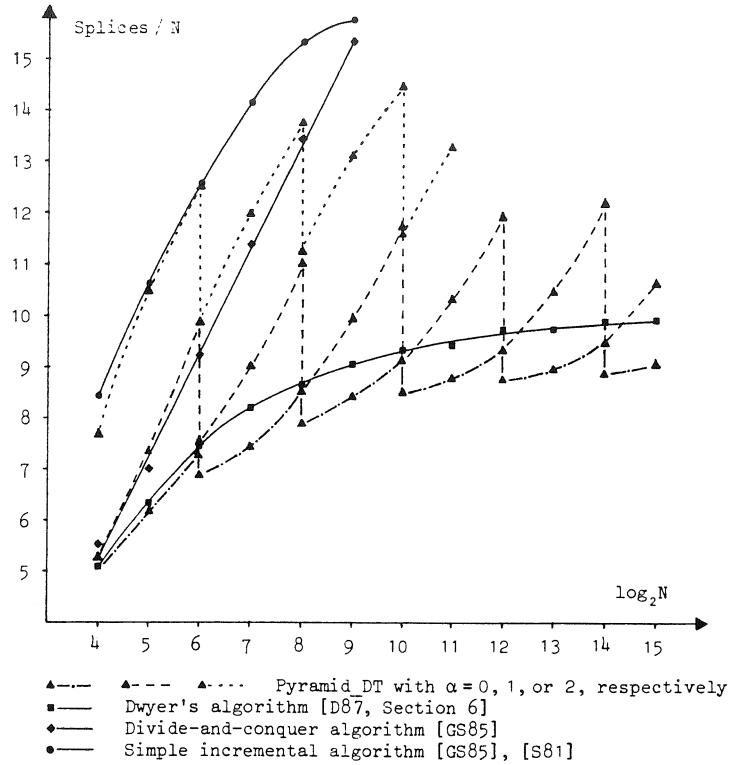

FIGURE 3
Mean number of splices performed by the different algorithms for uniformly distributed sites in the unit square.

5. CONCLUDING REMARKS

We have presented and analysed a Delaunay triangulation algorithm that runs in linear expected time and $\Theta(N \log N)$ worst-case time. It uses buckets of fixed sizes, and can be regarded as a two-dimensional counterpart of the distributive sorting algorithms for real numbers (see for example [MA80], [DK81], [D86]). However, the merge operation forms a fundamental part of our algorithm, whereas simple concatenation of lists suffices in sorting. The pyramid and the Morton merging orders seem to be very efficient, which is due to the fact that during the merges they limit the amount of re-modification of the diagram.

We close this paper by drawing attention to some open problems.

1) We have considered in this work the expected time only for quasi-uniformly distributed sites. It is natural to ask how efficient our algorithm is when the distribution of the sites is non-uniform.

2) The experiments indicate that, at least for uniformly distributed sites, the average-case behaviour of our algorithm and Dwyer's algorithm [D87, Section 6] is almost equal. Can one prove the linear average-case running time for the two-directional divide-and-conquer algorithm which was Dwyer's actual implementation? Observe here also the close analogy between the two-directional divide-and-conquer and the Morton merge algorithms.

3) Can one prove linearity for the algorithm which performs the merging of the cells in row and column order as proposed by Dwyer? A straightforward application of Corollary 3.9 yields an $O(N\sqrt{\log N})$ upper bound which is worse than that obtained by Dwyer!

4) It is obvious that the algorithm and its analysis can be extended to L_p-metrics, $1<p<\infty$, without any significant changes. The L_1 and L_∞ metrics and more general metrics are more difficult. We hope to return to this subject in another paper.

5) It would also be interesting to know whether quadtree merging could be used to solve other geometrical-computational problems in linear expected time.

ACKNOWLEDGEMENTS

We wish to express our sincere thanks to Rex Dwyer for his kind co-operation; especially we are very grateful to him and Jon Webb for allowing us to use their programs.

REFERENCES

[AB83] D. AVIS, B.K. BHATTACHARYA (1983). Algorithms for computing d-dimensional Voronoi diagrams and their duals, *Advances in Computing Research, Vol. 1: Computational Geometry*, F.P. PREPARATA (ed.), JAI Press, 159-180.

[BWY80] J.L. BENTLEY, B.W. WEIDE, A.C. YAO (1980). Optimal expected-time algorithms for closest point problems, *ACM Transactions on Mathematical Software* **6**, 563-580

[D85] L. DEVROYE (1985). Expected time analysis of algorithms in computational geometry, *Machine Intelligence and Pattern Recognition , Vol. 2: Computational Geometry*, G.T. TOUSSAINT (ed.), North-Holland, 135-151.

[D86] L. DEVROYE (1986). *Lecture Notes on Bucket Algorithms*, Birkhäuser.

[DK81] L. DEVROYE, T.KLINCSEK (1981). Average time behavior of distributive sorting algorithms, *Computing* **26**, 1-7.

[D87] R.A. DWYER (1987). A faster divide-and-conquer algorithm for constructing Delaunay triangulations, *Algorithmica* **2**, 137-151.

[F87] S. FORTUNE (1987). A sweepline algorithm for Voronoi diagrams, *Algorithmica* **2**, 153-174.

[GS77] P.J. GREEN, R. SIBSON (1977). Computing Dirichlet tessellations in the plane, *The Computer Journal* **21**, 168-173.

[GS85] L.J. GUIBAS, J. STOLFI (1985). Primitives for the manipulation of general subdivisions and the computation of Voronoi diagrams, *ACM Transactions on Graphics* **4**, 74-123.

[L77] C.L. LAWSON (1977). Software for C^1 surface interpolation, *Mathematical Software III*, J. RICE (ed.), Academic Press, 161-193.

[LS80] D.T. LEE, B.J. SCHACHTER (1980). Two algorithms for constructing a Delaunay triangulation, *International Journal of Computer and Information Sciences* **9**, 219-242.

[M84] A. MAUS (1984). Delaunay triangulation and the convex hull of n points in expected linear time, *BIT* **24**, 151-163.

[MA80] H. MEIJER, S.G. AKL (1980). The design and analysis of a new hybrid sorting algorithm, *Information Processing Letters* **10**, 213-218.

[OIM84] T. OHYA, M. IRI, K. MUROTA (1984). Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms, *Journal of the Operations Research Society of Japan* **27**, 306-337.

[PS85] F.P. PREPARATA, M.I. SHAMOS (1985). *Computational Geometry: An Introduction*, Springer-Verlag.

[S84] H. SAMET (1984). The quadtree and related hierarchical data structures, *Computing Surveys* **16**, 187-260.

[S78] M.I. SHAMOS (1978). *Computational geometry*, Ph.D. Thesis, Department of Computer Science, Yale University, New Haven, Conn.

[S81] M. SHAPIRO (1981). A note on Lee and Schachter's algorithm for Delaunay triangulation, *International Journal of Computer and Information Sciences* **10**, 413-418.