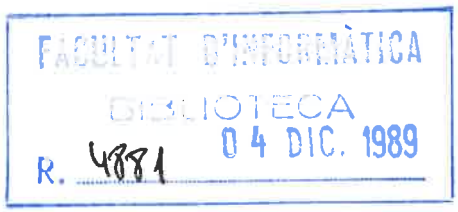
Parallel tetrahedization of a set of points in 3D

R.M. Giménez

Report LSI–88–10



Abstract: This paper provides a parallel algorithm that compute a tetrahedrization of a set of points. The algorithm works in $O(\log^2 n \log^* n)$ parallel time using $O(n)$ processors.

Abstract: Aquest paper presenta un algorisme paral.lel per construir una descomposició en tetraèdres d'un conjunt de punts. El algorisme treballa en temps $O(\log^2 n \log^* n)$ i fa servir $O(n)$ processadors.

PARALLEL TETRAHEDRIZATION OF A SET OF POINTS IN 3D

Rosa M. Gimenez*
Departament de Llenguatges i Sistemes Informàtics
Facultat d'Informàtica U.P.C.
08028 Barcelona, SPAIN

Keywords: Computational geometry, Tetrahedrization, Parallel algorithms, Complexity.

Abstract: This paper provides a parallel algorithm that compute a tetrahedrization of a set of points. The algorithm works in $O(\log^2 n \log^* n)$ parallel time using $O(n)$ processors.

1. Introduction

The algorithm presented in this paper constructs a tetrahedrization of a set of points in the three-dimensional Euclidean space. The problem is defined as the descomposition of the convex hull of the point set in tetrahedra such that 1) every vertex of the tetrahedra is an element in the point set, 2) every point in the set is a vertex in the tetrahedrization and 3) the intersection of the interior of two tetrahedra is always empty. We assume that the points are in general position.

The parallel running time in worst case is $O(\log^2 n \log^* n)$ using $O(n)$ processors, where $\log^* n$ is defined as the number of applications of the log function required to reduce n to a constant value. This complexity is due to the fact that the most expensive part of the algorithm is to compute the convex hull of the point set, shown to be $O(\log^2 n \log^* n)$ in [3].

The algorithm is based on the sequential process described in [2].

A concurrent read but exclusive write (CREW) memory model is assumed.

2. The algorithm

In this section, we give an outline of the algorithm for constructing a tetrahedrization of a set of points $\mathcal{P}$ in E^3.

* This research was partially supported by CIRIT EE87-1

Initial step:

Construct the convex hull of the point set $\mathcal{P}$: $CH(\mathcal{P})$. **(2.1)**

Tetrahedrize the convex hull obtained. **(2.2)**

In parallel for each point $p \in \mathcal{P}$ s.t $p \notin CH(\mathcal{P})$ **(2.3)**

Compute the tetrahedron τ, in which p lies

Iteration:

While there exists a non empty $\mathcal{P} \cap int(\tau)$ (int(τ) means interior(τ) **do**

In parallel for each tetrahedron τ s.t $\mathcal{P} \cap int(\tau) \neq \emptyset$:

Choose a point p in $\mathcal{P} \cap int(\tau)$ such that partitioning τ into four tetrahedra by spanning by p and the faces of τ, each one of the new tetrahedron contains a balanced number of points in its interior. **(2.4)**

Construct these four new tetrahedra

endwhile

3. Analysis

(2.1) Construct the convex hull: As it has been pointed out in the introduction, in [3] is shown how to construct the convex hull of a point set in E^3 in $O(\log^2 n \log^* n)$ parallel time, using $O(n)$ processors. The technique is the hierarchical representation. Following the notation used in [2], the number of points in the boundary of the convex hull will be called n' and the number of points in the interior n''.

(2.2) Tetrahedrize the convex hull: This can be easily done in $O(\log n')$ parallel time using $O(n')$ processors. The process is to choose a vertex v in the convex hull, then assigning one processor to each face that does not contain the point v, construct one tetrahedron for each face with the point v (the faces of the convex hull constructed above are triangulated).

(2.3) The purpose of this point of the algorithm is to compute $\mathcal{P} \cap int(\tau)$ for each tetrahedron τ: This problem can be reduced to the planar point location search. The subdivision hierarchy technique uses $O(\log n \log^* n)$ parallel preprocessing time and $O(\log n)$ sequential query time to perform planar point location. This technique can also be used in 3 dimensional applications due to the fact that every convex subdivision is equivalent to a bounded planar subdivision [3]. Therefore, assigning one processor to each interior point in the convex hull, we have a total query time of $O(\log n'')$. Thus, in $O(\log n' \log^* n') \cup O(\log n'')$ parallel time we have computed $\mathcal{P} \cap int(\tau)$ for every τ, in the actual tetrahedrization.

(2.4) Iteration: Complete the tetrahedrization. In [2] is shown the following result: Let S be a simplex with vertex set V, and let P be a set of n points in the interior of S such that no four points of $P \cup V$ are coplanar. Then there is a linear time algorithm that picks a point p of P such that each simplex spanned by p and a facet of S contains at most $\beta_d(n)$ points of P, where $\beta_d(n) = \lfloor \frac{d.n}{d+1} \rfloor$ and d is the dimension of

the Euclidean space.

In our case, for each tetrahedron τ, we have to find a point $p \in \mathcal{P} \cap int(\tau)$ such that for each new tetrahedron τ', $\mathcal{P} \cap int(\tau')$ is at most $\beta_3(m)$, where $\beta_3(m) = \lfloor \frac{3m}{4} \rfloor$ and $m = |\mathcal{P} \cap int(\tau)|$. F_i is defined as the facet of τ that does not contain the vertex v_i of τ.

A parallel algorithm to find such a point p, can be the following:

In parallel for each τ that $\mathcal{P} \cap int(\tau) \neq \emptyset$

  In parallel for each F_i, $1 \leq i \leq d+1$

    Choose an edge in F_i and call it F_i'

  In parallel for each F_i' and for each $m \in \mathcal{P} \cap int(\tau)$

    Construct the hyperplane passing through F_i' and m

  In parallel for each F_i' **(3.1)**

    Choose the hyperplane that has exactly $\lceil \frac{m}{d+1} \rceil - 1$ points
    of $\mathcal{P} \cap int(\tau)$ on the opposite side of it from F_i
    Mark these $\lceil \frac{m}{d+1} \rceil - 1$ points

  Choose one point unmarked and construct the four tetrahedra associated with it.

Notice that if we tetrahedrize throgh one of the unmarked points, then the maximum number of points that a new tetrahedron can have in its interior is $\lceil m - \frac{m}{d+1} \rceil$, this is because each new tetrahedron τ_i' constructed with the face F_i and the point p, does not contain any of the points marked in the process (3.1) associated with the face F_i. For more details see [2].

(3.1) One way to do this could be to sort in decreasing order the hyperplanes in the order defined by the angle α between F_i and the hyperplane, and then mark the $\lceil \frac{m}{d+1} \rceil - 1$ points of the hyperplanes in the $\lceil \frac{m}{d+1} \rceil - 1$ first positions; but there is a more efficient way to do it: In [1], it is shown how to select the K^{th} item between a set of n elements in $O(\log \log n)$ parallel time with $O(n)$ processors, without sorting the elements; so the only we have to do is to select the hyperplane in the $\lceil \frac{m}{d+1} \rceil - 1$ position and then assigning one processor to each point $p \in \mathcal{P} \cap int(\tau)$, determine in which side of the hyperplane lies the point p. Doing this, we know the points to be marked.

After this process, we have chosen a point p for each tetrahedron τ with $\mathcal{P} \cap int(\tau) \neq \emptyset$. Thus, if we tetrahedrize through this point, we obtain four more tetrahedra with at most $\frac{3}{4}m$ points each one. If we repeat the process with each one of the new tetrahedra then we will obtain 16 tetrahedra with at most $(\frac{3}{4})^2 m$ points each one, and so on. Since the sequence of $|\mathcal{P} \cap int(\tau)|$ forms a decreasing geometric series, the total number of steps is $O(\log n)$ and since each execution of the iteration costs $O(\log \log n)$, the total time is $O(\log n \log \log n)$. We can conclude that the total complexity of the algorithm is:

In the initial step:

	time	**processors**
Convex hull	$O(\log^2 n \log^* n)$	$O(n)$
Initial tetrahedrization	$O(\log n')$	$O(n')$
Construct $\mathcal{P} \cap int(\tau), \forall \tau$	$O(\log n' \log^* n') \cup O(\log n'')$	$O(n') \cup O(n'')$

and in the Iteration:

$O(\log n'' \log\log n'')$ $O(n'')$

Therefore, the total parallel running time of the algorithm is $O(log^2 n \log^* n)$ and the number of processors required is $O(n)$.

References

[1] M. Ajtai, J. Komlos, W.L. Steiger and E. Szemeredi, Deterministic selection in $O(\log\log n)$ parallel time, FOCS 27th, 1986.

[2] H.Edelsbrunner, F.P.Preparata and D.B.West, Tetrahedrizing points sets in three dimensions, Tech.Rept. Dept. of Computer Science, Univ. of Illinois at Urbana-Champaign, UIUCDCS-R-86-1310, 1986.

[3] N.Dadoun and D.G.Kirkpatrick, Parallel Processing for efficient Subdivision Search, Proc.3rd ACM Computational Geopmetry Conf. (1987) 205-214.

[4] F.P. Preparata and M.I. Shamos, Computational Geometry, Springer-Verlag, New York, 1985