An NC^1 Parallel 3D Convex Hull Algorithm

Nancy M. Amato
Coordinated Science Lab.
University of Illinois
1308 W. Main St.
Urbana, IL 61801
amato@cs.uiuc.edu

Franco P. Preparata*
Dept. Computer Science
Brown University
Box 1910
Providence, RI 02912
franco@cs.brown.edu

Abstract

In this paper we present an $O(\log n)$ time parallel algorithm for computing the convex hull of n points in $\Re^3$. This algorithm uses $O(n^{1+\alpha})$ processors on a CREW PRAM, for any constant $0 < \alpha \leq 1$. So far, all adequately documented parallel algorithms proposed for this problem use time at least $O(\log^2 n)$. In addition, the algorithm presented here is the first parallel algorithm for the three-dimensional convex hull problem that is not based on the serial divide-and-conquer algorithm of Preparata and Hong, whose crucial operation is the merging of the convex hulls of two linearly separated point sets. The contributions of this paper are therefore (i) an $O(\log n)$ time parallel algorithm for the three-dimensional convex hull problem, and (ii) a parallel algorithm for this problem that does not follow the traditional divide-and-conquer paradigm.

1 Introduction

Convex hulls are one of the most fundamental geometric constructs. In addition to being of considerable interest in their own right, convex hulls are often useful in solving apparently unrelated problems in Computational Geometry. Therefore, considerable research effort has focused on developing algorithms, both serial and parallel, for computing convex hulls.

The sequential complexity of computing the convex hull of a set S of n points in $\Re^d$, $d = 2, 3$, is known to be $\Omega(n \log n)$ (see, e.g., [PS85]). Although there exist several optimal serial algorithms for this problem when $d = 2$ and $d = 3$, optimal parallel algorithms are known only for $d = 2$; in general, a parallel algorithm is said to be optimal if the product of the time and the number of processors used (processor-time product) is of the same order as the sequential running time of the problem. When working in $\Re^3$, several algorithms have been proposed: the first due to Chow [C80] required $O(\log^3 n)$ time and $O(n)$ processors, Aggarwal *et al.* [ACGOY88] proposed a new algorithm with these same time and processor bounds, Dadoun and Kirkpatrick [DaK89] implemented the algorithm of Aggarwal *et al.* more efficiently, and more recently Amato and Preparata [AP92] gave an algorithm using $O(\log^2 n)$ time and $O(n)$ processors. All of the above algorithms use the CREW PRAM model of parallel computation (for details of the various PRAM models consult [KR91]). Thus, it remains an important open problem to find an optimal parallel algorithm for computing the convex hull of a point set in $\Re^3$. Although the $O((1/\alpha)\log n)$ time and $O(n^{1+\alpha})$ processor algorithm, for any constant $0 < \alpha \leq 1$, we present here is still sub-optimal, it has the important feature that it achieves $O(\log n)$ time for the three-dimensional convex hull problem, whereas all

*The work of this author was supported in part by NSF Grants CCR-91-96152, CCR-91-96176, and ONR Contract N00014-91-J-4052, ARPA order 8225.

prior parallel algorithms for this problem use at least $O(\log^2 n)$ time.[1]

All traditional parallel algorithms for the three-dimensional convex hull problem are based on the serial divide-and-conquer algorithm of Preparata and Hong [PH77]. Let $CH(A)$ denote the convex hull of the point set A. The serial algorithm for computing the convex hull of a point set S can be outlined as follows: the set S is evenly divided into two sets P and Q such that the z-value of each vertex in P is greater than the z-value of every vertex in Q; $CH(P)$ and $CH(Q)$ are recursively computed; the cycle of supporting faces that are tangent to $CH(P)$ and $CH(Q)$ is computed; finally, $CH(P)$ and $CH(Q)$ are merged along the cycle of supporting faces just computed to form $CH(S) = CH(P \cup Q)$. Note that the edges of $CH(P)$ ($CH(Q)$) that are incident to the cycle of supporting faces of $CH(P)$ and $CH(Q)$ will form a circuit in which vertices may be visited more than once and the same edge may occur with both orientations along the circuit; this circuit is referred to as the *upper seam* (*lower seam*).

Any algorithm inspired by the above paradigm (referred to here as "bisect-and-conquer", to stress the subdivision into *two* subproblems) necessarily consists of $O(\log n)$ merge phases. Since it seems unlikely that two linearly separated convex hulls can be merged in constant time, any parallel algorithm based upon this approach seems doomed to require time $\omega(\log n)$ (i.e., greater than logarithmic). Thus, for some time it has been suggested in the research community that only a finer subdivision than bisection is likely to improve the time performance; our algorithm is the first to achieve such an objective. In particular, the algorithm presented in this paper avoids the above drawback by departing from the bisect-and-conquer paradigm as follows: instead of dividing the point set S into two subsets, we partition it into $O(n^\alpha)$ subsets, each of size $O(n^{1-\alpha})$, recursively construct the convex hull of each subset, and then merge the resulting $O(n^\alpha)$ convex hulls to form $CH(S)$, $0 < \alpha \leq 1$. However, our approach greatly alters the overall strategy of the merging phase of the algorithm as follows. Note that in the traditional bisect-and-conquer approach, each face of the merged hull contains at least one edge from one of the subhulls; indeed, since the surface of the convex hull is assumed to be triangulated, a face of the merged hull is either a face of one of the subhulls, or it contains a seam edge of one of the subhulls and a seam vertex of the other subhull. Thus, the merging process in the bisect-and-conquer approach can be accomplished by classifying each edge (or face) of the subhulls as either external or internal to the merged hull. However, in our case, a face of the merged hull need not contain an edge of any of the subhulls, and thus the merging process must classify each *vertex* of one of the subhulls as either internal or external to the final merged hull, i.e., the traditional edge classification must be replaced by vertex classification.

Thus, the contributions of this paper to the understanding of the parallel three-dimensional convex hull problem are as follows. First, we depart from the bisect-and-conquer paradigm that has dominated all previous parallel algorithms for this problem; a necessary consequence of our new approach is the development of a criterion for classifying vertices, rather than edges or faces, as either internal or external to the convex hull. Secondly, and perhaps more importantly, we have shown that it is indeed possible to compute the convex hull of a point set in $\Re^3$ in $O(\log n)$ time with a relatively small number of processors.

We finally note here that there exists a trivial algorithm for computing the convex hull of a three-dimensional point set S in $O(\log n)$ time using $O(n^4)$ processors on a CREW PRAM, where $|S| = n$. This algorithm is face-based, rather than edge- or vertex-based. We determine the faces of $CH(S)$ as follows. For each of the $\binom{n}{3} = O(n^3)$ subsets $S' \subset S$ of cardinality three, determine if they form a face of $CH(S)$ by checking to see if all points of S lie on one side of the plane containing S'. This takes $O(\log n)$ time using $O(n)$ CREW PRAM processors for each subset S'. Then, the order of the faces around each vertex of $CH(S)$

[1]Related independent and concomitant research has been reported in [PDW92], which outlines a (substantially different) technique claimed to achieve the same time and processor complexities as the one we present here.

can be found in $O(\log n)$ time using $O(n)$ processors. Thus, the entire process takes time $O(\log n)$ using $O(n^4)$ processors on a CREW PRAM.

2 Breaking the $O(\log^2 n)$ Barrier

In this section we present a relatively simple algorithm that computes the convex hull of n points in $O(\log n)$ time using $O(n^2)$ processors on a CREW PRAM. Although this algorithm does much more work than the algorithm we present in the next section, it allows us to introduce some useful notation and illustrate techniques that are also used in the more efficient algorithm we will present later. The following definition formalizes the concepts of internal and external.

Definition: Let S denote a point set in $\Re^3$. A point $v \in S$ is said to be *external* if it is a vertex of $CH(S)$; similarly, a point $v \in S$ is said to be *internal* if it is not external, i.e., it is contained in the interior of $CH(S)$ (or, equivalently, there are four points in $S - \{v\}$ such that v is internal to their determined tetrahedron). If v is external to $CH(S)$, then the other external vertices that are incident to v on $CH(S)$ will be referred to as v's *neighborhood*, and will be denoted by $n(v)$; the set $n(v) = \{n_0, n_1, \dots, n_{k-1}\}$ is assumed to be ordered so that the vertices v, n_i, and $n_{(i+1) \bmod k}$ determine a face of $CH(S)$, $0 \le i < k$.

In order to compute the convex hull of a point set S in $\Re^3$, $|S| = n$, we note that it is sufficient to classify each $v \in S$ as either internal or external to $CH(S)$, and if v is external to compute the sequence $n(v)$. Once we have determined the convex hull vertices, and their neighborhoods, it is a simple matter to form the standard doubly connected edge list (DCEL) representation of the convex hull; recall that in a DCEL we record, for each edge, its two endpoints v_1 and v_2, the two faces incident to it, and the edges which follow it in a clockwise traversal around the edges incident to v_1 and v_2 on $CH(S)$ (see, e.g., [PS85]). Thus, in order to show that $CH(S)$ can be computed in $O(\log n)$ time using $O(n^2)$ processors on a CREW PRAM, it is sufficient to show that, using $O(\log n)$ time and $O(n)$ processors, each point v of S can be classified as either internal or external, and that the ordered set $n(v)$ can be computed if v is external.

We now discuss how a vertex $v \in S$ can be classified as either internal or external, and how the set $n(v)$ can be found if v is external. Let $v \in S$, and define $r(v,p)$ as the ray originating at v and passing by some other point $p \in S$. Next, define the set of rays $R_v = \{r(v,p) \mid p \in S - \{v\}\}$. The following simple lemma is the geometric basis of our algorithm.

Lemma 1: A vertex $v \in S$ is external to $CH(S)$ if and only if v is external to $CH(R_v)$. [In particular, if v is internal to $CH(S)$, then $CH(R_v) = \Re^3$, and if v is external to $CH(S)$, then $CH(R_v) \neq \Re^3$ and v's neighborhood (the set $n(v)$) lies on the boundary edges (rays) of $CH(R_v)$.]

Proof: Note that $CH(R_v)$ either has the single finite vertex v or it coincides with $\Re^3$. Assume that $v \in S$ is internal to $CH(S)$. Then there are four points of S, say p_1, p_2, p_3, and p_4, whose tetrahedron contains v in its interior; it follows that the convex combination of the four rays $\{r(v,p_i) \mid i = 1,2,3,4\}$ coincides with $\Re^3$, i.e., v is internal to $CH(R_v)$. Conversely, assume that v is external to $CH(R_v)$. Then, for any four other points of S, their determined tetrahedron does not contain v, thus proving that v is also external to $CH(S)$. □

Thus, in order to classify $v \in S$ as either internal or external to $CH(S)$, it is enough to find $CH(R_v)$, and moreover, if v is external, then $CH(R_v)$ will yield the ordered set $n(v)$. Although the problem of computing $CH(R_v)$ may seem to be as difficult as computing $CH(S)$, as we will see below, the fact that all rays in R_v originate at v greatly simplifies matters. The above discussion yields the following algorithm for classifying each $v \in S$ as either internal or external to $CH(S)$.

```
Algorithm: CLASSIFY($v, S$)
  1. Construct $R_v = \{r(v,p) \mid p \in S - \{v\}\}$.
  2. Compute $CH(R_v)$.
  3. If ($CH(R_v) = \Re^3$)
       then return (internal)
       else return (external and $n(v)$)
```

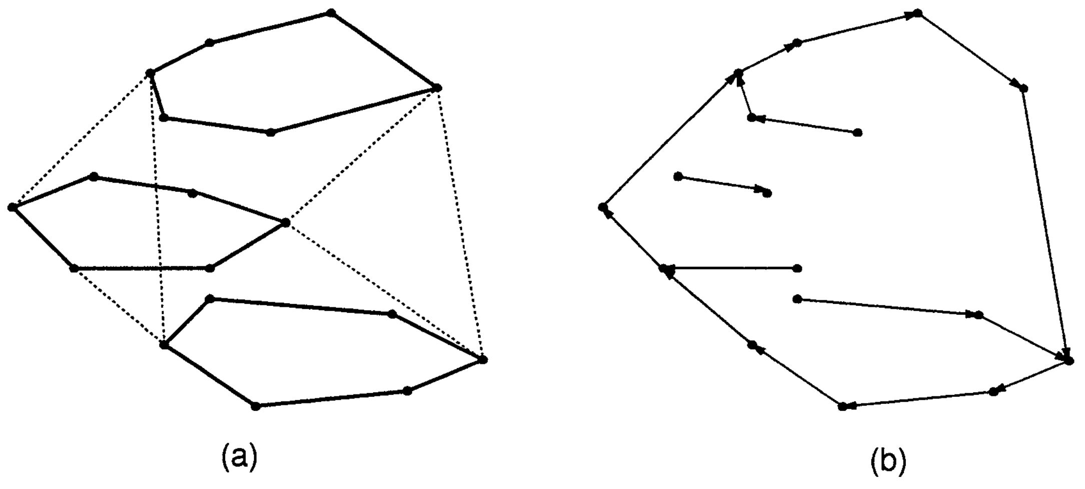

Figure 1: (a) Three subhulls and their common supporting segments. (b) The directed graph corresponding to the subhulls and their supporting segments.

The correctness of CLASSIFY follows directly from Lemma 1, and thus we need only determine its complexity. We let $|S| = m$. We first note that Steps 1 and 3 can be implemented in $O(1)$ time using $O(m)$ processors.

As will be fully explained below, the problem of constructing $CH(R_v)$ (Step 2) is closely related to the problem of computing the convex hull of a planar point set. For this reason, we sketch for convenience a slight variant of the algorithm of Aggarwal *et al.* [ACGOY88] for computing the convex hull of a set A of k points in the plane in $O(\log k)$ time using $O(k)$ processors on a CREW PRAM. The set A is divided (by increasing y-coordinate) into $\sqrt{k}$ subsets, $A_1, A_2, \ldots, A_{\sqrt{k}}$, each of $\sqrt{k}$ points. Then, each $CH(A_i)$, $1 \leq i \leq \sqrt{k}$, is recursively computed and the resulting $\sqrt{k}$ subhulls are merged to form $CH(A)$. The merging process proceeds as follows. First, the two common supporting segments for all pairs of subhulls (there are $O(k)$ such pairs) are computed in $O(\log k)$ time using $O(k)$ processors (one processor per pair) by the sequential technique of Dobkin and Kirkpatrick [DK90] (see Fig. 1(a)). Next, all segments incident on each point $p \in A$ (including both the supporting segments and the edges of the subhulls) are sorted by slope in $O(\log k)$ time using $O(k)$ processors [Co88]: if two consecutive segments form an angle $> \pi$, then p is a potential convex hull vertex, and if no two consecutive segments form an angle $> \pi$, then p is internal. Next, a directed graph (digraph) is constructed from some of these segments so that the only cycle in the digraph gives $CH(A)$. Specifically, the vertices of the digraph, V_d, are the potential hull vertices identified above; for each vertex $v \in V_d$, let a_v denote the potential convex hull edge originating at v on a clockwise traversal of the convex hull. The arcs of the digraph are those arcs a_v, $v \in V_d$, such that the terminus of a_v is also in V_d, i.e., it is also a potential hull vertex. (See Fig. 1(b).) Note that since only one edge exits from any vertex, the digraph can only contain one cycle, and that cycle must be the desired convex hull. Finally, a list ranking operation on the digraph identifies the cycle, and thus completes the merging process in time $O(\log k)$ using $O(k)$ processors [KR91]. Thus, the running time of the algorithm satisfies the recurrence $T(k) = T(\sqrt{k}) + O(\log k) = O(\log k)$ using $O(k)$ processors on a CREW PRAM. (The algorithm of [ACGOY88] differs from the one sketched here in that it uses a different technique of computing the pairwise supporting tangents, and constructs the upper and lower hulls separately. We

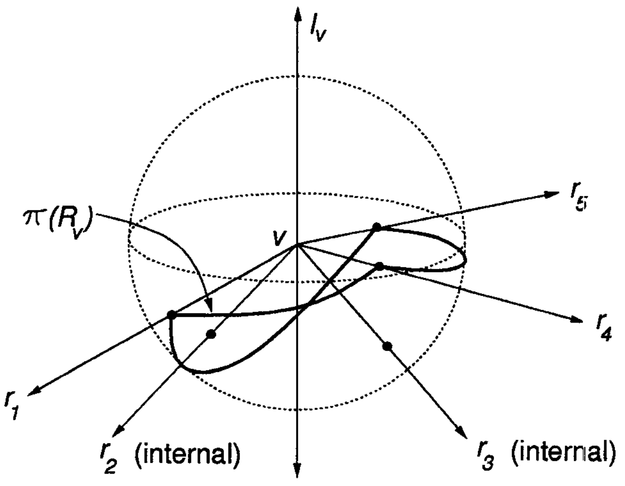

Figure 2: The spherical polygon $\pi(R_v)$ determined by rays r_1, r_2, r_3, r_4, and r_5.

have chosen the variant described above because it generalizes more naturally to our three-dimensional application.)

We now return to the problem of constructing $CH(R_v)$. If $CH(R_v) \neq \Re^3$, any plane H intersecting $CH(R_v)$, intersects it in a (possibly unbounded) convex polygon. In particular, consider a ray $r \in R_v \cap CH(R_v)$, and let $u(r)$ be the point of r at unit distance from v. Then the plane $H(r)$ orthogonal to r and containing $u(r)$ intersects $CH(R_v)$ in a convex polygon $P(r)$. The edges of $P(r)$ incident on $u(r)$ belong to lines tangent to the unit sphere Σ centered at v. It is realized that, when $CH(R_v) \neq \Re^3$, the intersection of $CH(R_v)$ with Σ is a spherical polygon $\pi(R_v)$, whose edges are arcs of Σ's great circles, so that their tangents at $u(r)$ contain the edges incident upon $u(r)$ in $P(r)$ (see Fig. 2). We now explain how the above described planar convex hull algorithm can be adapted to construct the spherical polygon $\pi(R_v)$.

The two portions of the planar algorithm that must be reinterpreted are: (i) the method of partitioning the input set, and (ii) the mechanism by which subhulls are merged. The natural solution to the latter is to use the slopes of tangents incident on $u(r)$ with respect to a standard reference on a plane tangent to Σ at $u(r)$ (such as parallel-meridian system) as the slopes of segments tangent to a point were used in the original algorithm. For the former, recall that in the plane the point set is partitioned by one of the two coordinates. The natural generalization of this when computing the spherical polygon $\pi(R_v)$ is to partition the set of rays R_v with respect to a system of polar angles. Such a system can be devised as follows. (We assume here that $|R_v| \geq 3$, else $CH(R_v)$ can be obtained by direct computation.) Let l_v be any line passing by v, let T_v be any plane containing l_v, and let H_v be either halfplane of T_v defined by l_v; selection of H_v determines a system of polar angles around l_v.

The computation of $\pi(R_v)$, and thus $CH(R_v)$, is now described. Let $(r_1, r_2, \ldots, r_m)$ denote the cyclic list of rays in R_v (ordered by polar angle around l_v); this ordered list can be constructed by sorting the m rays in R_v by angular order in $O(\log m)$ time using $O(m)$ processors [Co88]. We partition this list into $\sqrt{m}$ sublists, $R_1, R_2, \ldots, R_{\sqrt{m}}$, each of size $\sqrt{m}$, so that R_1 follows $R_{\sqrt{m}}$ in the chosen order. We recursively construct the polygons $\{\pi(R_j) | j = 1, \ldots, \sqrt{m}\}$, where $\pi(R_j) = CH(R_j) \cap \Sigma$. Note that if, for any $1 \leq j \leq \sqrt{m}$, the point set $R_j \cap \Sigma$ is not contained in any hemisphere of Σ, then $CH(R_j) = \Re^3$, and thus $CH(R_v) = \Re^3$, i.e., v is internal and the computation can stop; this condition will be detected by the recursive application of the merging process as described below.

At the last stage of the construction, we perform the pairwise merges of the polygons $\{\pi(R_j) | j = 1, \ldots, \sqrt{m}\}$. In general, each pairwise merge is accomplished by constructing two common supporting great circles for the pair of polygons, where each great circle is centered at v and passes by a point on the boundary of each polygon; we note here that it is easily shown that the task of computing the supporting great circles is nearly as simple as computing the common supporting lines in the plane case. Recall that in the plane case, two common supporting lines were required for each pairwise merge of subhulls, and the convex hull of two subhulls was always a bounded convex set. This situation is somewhat altered in the present setting since if the two subhulls are not contained in a single hemisphere of Σ, then the convex span of their corresponding rays is $\Re^3$. Otherwise, as in the plane case, two supporting arcs (and two great circles) are needed to complete the merge. A useful feature of the technique of Dobkin and Kirkpatrick [DK90] (and the reason for adopting it here), as generalized to spherical polygons, is that it can easily distinguish between the above cases; initially it is assumed that two supporting great circles are required, but if at any point in the computation a supporting great circle cannot be computed, then the two subhulls are not contained in a single hemisphere, i.e., v is internal.

After the pairwise merges are done, all tangents incident on a given point $u(r)$ on Σ are sorted by slope: if no two consecutive tangents form an angle $> \pi$, then $u(r)$ corresponds to a ray r that is internal. If all rays are found to be internal, then v is also internal. If v is external, then, as in the planar algorithm, a standard list ranking operation constructs the neighborhood of v. From the preceding discussion, we conclude that the final merge step is implementable in $O(\log m)$ time with $O(m)$ CREW PRAM processors.

Therefore, the time complexity of CLASSIFY(v, S) is $O(\log m)$ using $O(m)$ processors on a CREW PRAM, where $|S| = m$. Thus, we have the following theorem.

Theorem 1: The convex hull of n points in $\Re^3$ can be found in $O(\log n)$ time using $O(n^2)$ processors on a CREW PRAM.

Proof: For a given point $v \in S$, the algorithm CLASSIFY(v, S) will determine whether or not v is external to $CH(S)$, and moreover, if v is external, will compute v's neighborhood on $CH(S)$. CLASSIFY(v, S) requires $O(\log n)$ time and $O(n)$ processors on a CREW PRAM. Therefore, performing CLASSIFY for each $v \in S$ will require $O(\log n)$ and $O(n^2)$ processors on a CREW PRAM. $\square$

We end this section with a brief sketch of an alternative method for determining whether a particular point v is internal to $CH(S)$. We consider the set $R_v = \{r(v,p) \mid p \in S - \{v\}\}$ and the two planes T_1 and T_2, whose respective equations are $z = z(v) - 1$ and $z = z(v) + 1$. We let $R_v^{(j)} = \{r \mid r \in R_v \text{ and } r \cap T_j \neq \emptyset\}$, $j = 1, 2$. Using the algorithm of [ACGOY88] we construct, in time $O(\log n)$ with $O(n)$ processors, the convex polygons $P^{(j)} = CH(R_v^{(j)} \cap T_j)$, $j = 1, 2$. It is readily shown that v is external if and only if $P^{(1)} \cap P^{(2)'} = \emptyset$, where $P^{(2)'}$ is the projection through v of $P^{(2)}$ on T_1. This can be determined in time $O(\log n)$ with a single processor [DK90]. If v is external, then $CH(R_v)$ can be constructed as follows. In time $O(1)$ with a single processor we determine a plane H intersecting all members of both $R_v^{(1)}$ and $R_v^{(2)}$. (H is parallel to the plane determined by v and the two points of support of a segment separating $P^{(1)}$ and $P^{(2)'}$, which was computed when testing if $P^{(1)} \cap P^{(2)'} = \emptyset$.) The central projections (from v) of $P^{(1)}$ and $P^{(2)}$ on H are also obtained in $O(1)$ time using $O(n)$ processors, and can be merged into a single polygon - the desired convex hull - in $O(\log n)$ time with one processor. This alternative technique also yields $CH(R_v)$ in time $O(\log n)$ with $O(n)$ processors.

3 A More Efficient Algorithm

In this section we present our algorithm for constructing the convex hull of n points in $\Re^3$ in $O((1/\alpha)\log n)$ time using $O(n^{1+\alpha})$ processors on a CREW PRAM, for any constant $0 < \alpha \leq 1$. This algorithm utilizes the CLASSIFY procedure developed in the previous section; recall that we performed one CLASSIFY operation for each $v \in S$, and that the entire point set was used in each such invocation. Although the more efficient algorithm presented below may still perform a CLASSIFY operation for each $v \in S$, the work done by all the CLASSIFY operations in the aggregate will be less, i.e., although $O(n)$ points may still be required in a few CLASSIFY operations, in total only $O(n^{1+\alpha})$ points will be used, rather than the $O(n^2)$ that were used in the previous algorithm. Thus, the trick to reducing the complexity of our initial algorithm is to select a subset of the original points that is sufficient to allow CLASSIFY to determine whether a particular vertex v is internal or external to $CH(S)$. We begin by giving an overview of the algorithm.

Algorithm: 3D CONVEX HULL(S, n, α)

1. Partition S, by z-coordinate, into n^α groups, each of size $n^{1-\alpha}$; let $n^\alpha = m$ and denote the ith such group by S_i.

2. Recursively compute $CH(S_i)$, $1 \le i \le m$.

3. Merge $CH(S_i)$, $1 \le i \le m$, to form $CH(S) = CH(\cup_{1\le i\le m} CH(S_i))$.

Recall that we can sort all points in S by z-coordinate in $O(\log n)$ time with $O(n)$ processors on an EREW PRAM using Cole's parallel merge sort [Co88]. Thus, the time complexity of the above algorithm will satisfy the recurrence:

$$T(n) = O(\log n) + T(n^{1-\alpha}) + M(n^\alpha, n^{1-\alpha}) \quad (1)$$

where $M(n^\alpha, n^{1-\alpha})$ is the time required to merge n^α linearly separated convex hulls, each of size $n^{1-\alpha}$. In the remainder of this section we will show that $M(n^\alpha, n^{1-\alpha}) = O(\log n^{1+\alpha})$, so that $T(n) = O((1/\alpha)\log n)$; the algorithm uses $O(n^{1+\alpha})$ processors on a CREW PRAM.

We begin with some useful definitions. Consider two separable convex hulls $CH(P)$ and $CH(Q)$ and the convex hull of their union $CH(P \cup Q)$, where $|P| = |Q| = O(n)$. As before, a vertex $v \in CH(P)$ is *external* if it is a vertex of $CH(P \cup Q)$, and otherwise it is *internal*, and if v is external, the set of vertices that are incident to v on $CH(P \cup Q)$ is v's *neighborhood* and will be denoted by the ordered set $n(v)$. In addition, a vertex $v \in CH(P)$ is said to be a *seam* vertex if it is an external vertex and it is incident to some vertex $w \in CH(Q)$ on $CH(P\cup Q)$, i.e., one of its neighbors on $CH(P\cup Q)$ is a vertex of $CH(Q)$. External vertices that are not seam vertices will be referred to as *e-external* vertices, and external vertices that are seam vertices will be referred to as *s-external* vertices.

We now consider the problem of merging $O(m)$ convex hulls $CH(S_i)$, $1 \le i \le m$, each of size $O(n/m)$. We first describe the merging process, and then prove the correctness of the technique and analyze its complexity.

Algorithm: MERGE
input: $CH(S_i)$, $1 \le i \le m$, where $|S_i| = n/m$
output: $CH(S) = CH(\cup_{1\le i\le m} CH(S_i))$

1. For all $1 \le i < j \le m$ compute $CH(CH(S_i) \cup CH(S_j))$. [There are $\binom{m}{2} = O(m^2)$ such pairwise merges.]

2. Let S^* denote the set of all vertices that resulted external in all pairwise merges in Step 1, and s-external in at least one pairwise merge. For each $v \in S^*$, construct the set A_v, which consists of all vertices that were adjacent s-external vertices to v in some pairwise merge, i.e., if $v \in CH(S_i)$ and $w \in A_v$, then w belongs to v's neighborhood on $CH(CH(S_i) \cup CH(S_j))$, where $w \in CH(S_j)$ for some $1 \le j \le m$ and $j \ne i$.

3. Consider vertex $v \in S^*$; assume that $v \in CH(S_i)$ and let $n_i(v)$ denote v's neighborhood on $CH(S_i)$. For each such vertex v, determine if v is external or internal to $CH(S)$ by performing the operation CLASSIFY$(v, A_v \cup n_i(v))$.

4. Adjoin to the set $\{v \mid v \in S^*$ and v has been classified external in Step 3 $\}$ the set of points that resulted e-external in all pairwise merges; these are the vertices of $CH(S)$. Form a doubly connected edge list (DCEL) representation of $CH(S)$, and then build a hierarchical representation of $CH(S)$. [The hierarchical representation data structure [DK90] is necessary for the enclosing recursive pairwise merges contemplated in Step 1.]

Before analyzing the complexity of MERGE, we establish its correctness. We begin by stating the following simple facts. For convenience we will denote the set of convex hulls resulting from the pairwise merges in Step 1 as $C = \{C_{i,j} | 1 \le i < j \le m\}$, where $C_{i,j} = CH(CH(S_i) \cup CH(S_j))$.

Fact 1: If a vertex v of $CH(S_i)$ or $CH(S_j)$ is internal to $C_{i,j}$, for some $1 \le i < j \le m$, then it is also internal to $CH(S)$.

Fact 2: If a vertex $v \in CH(S_i)$ is e-external to $C_{i,j}$ and $C_{j',i}$, for all $1 \le j' < i < j \le m$, then v is also external to $CH(S)$; moreover, the vertices incident to such a vertex v on $CH(S)$ will be the vertices that are incident to v on $CH(S_i)$, $1 \leq i \leq m$, i.e., $n(v) = n_i(v)$.

Fact 3: If a vertex $v \in CH(S_i)$ is s-external to at least one $C_{i,j}$ or $C_{j',i}$, and is not internal to any $C_{i,j}$ or $C_{j',i}$, for all $1 \leq j' < i < j \leq m$, then v could be internal or external to $CH(S)$.

The above facts imply that, after Step 1 of the merging process, the points in S can be naturally partitioned into the three following classes: I^* (Fact 1), E^* (Fact 2), and S^* (Fact 3), where I^* consists of those vertices that are known to be internal to $CH(S)$ (disregarded in subsequent computations), E^* consists of those vertices that are known to be external to $CH(S)$ (their neighborhoods in $CH(S)$ are already known), and S^* consists of those vertices whose status with respect to $CH(S)$ remains unknown (this is the set identified in Step 2). Therefore, we now argue that the remaining steps (2-4) of the merging process correctly characterize all S^* vertices with respect to $CH(S)$.

The following lemma establishes that the point set S is contained in $CH(R_v^*)$, where $v \in S^* \cap CH(S_i)$ and $R_v^* = \{r(v,p) | \, p \in A_v \cup n_i(v)\}$. It then follows that $CH(S) \subset CH(R_v^*)$, which implies that (i) v is external to $CH(S)$ if and only if it is external to $CH(R_v^*)$, (or equivalently, v is internal to $CH(S)$ if and only if it is internal to $CH(R_v^*)$), and (ii) if v is external, then v's neighborhood on $CH(S)$ must lie on the boundary rays of $CH(R_v^*)$, i.e., $CH(R_v^*)$ identifies the ordered set $n(v)$.

Lemma 2: If $v \in S^* \cap CH(S_i)$, then $S \subset CH(R_v^*)$, where $R_v^* = \{r(v,p) | \, p \in A_v \cup n_i(v)\}$.

Proof: Consider some $v \in S^* \cap CH(S_i)$, where $R_v^* = \{r(v,p) | \, p \in A_v^*\}$, and $A_v^* = A_v \cup n_i(v)$. To obtain a contradiction, assume there is some $w \in S$ such that $w \notin CH(R_v^*)$. It is easy to see that $w \notin S_i$; indeed, since $n_i(v) \subset A_v^*$ it must be that $CH(S_i) \subset CH(R_v^*)$, i.e., $CH(S_i) \subset CH(R_i) \subset CH(R_v^*)$, where $R_i = \{r(v,p) | \, p \in n_i(v)\}$. Thus, it must be that $w \in S_j$, for some $j \neq i$. Consider $C_{i,j} = CH(CH(S_i) \cup CH(S_j))$; clearly v is external to $C_{i,j}$, because otherwise v would be a vertex of I^* and not a vertex of S^*. Let $n_{i,j}$ denote v's neighborhood on $C_{i,j}$, and let $R_{i,j}$ denote the set of rays $\{r(v,p) | \, p \in n_{i,j}\}$. Note that $CH(S_j) \subset C_{i,j} \subset CH(R_{i,j})$, i.e., $w \in CH(R_{i,j})$. However, this implies that $w \in CH(R_v^*)$, since by definition of A_v^* we have $n_{i,j} \in A_v^*$, which contradicts our assumption that $w \notin CH(R_v^*)$. □

Lemma 2 establishes the correctness of MERGE, and thus, we now turn our attention to its complexity. We recall that the $O(\log^2 n)$ time, $O(n)$ processor CREW PRAM algorithm of [AP92] for computing the convex hull of an arbitrary point set in $\Re^3$, uses a bisect-and-conquer strategy in which, at each of $O(\log n)$ stages, two separable convex hulls, $CH(P)$ and $CH(Q)$, are merged to form $CH(S) = CH(P) \cup CH(Q)$. Providing that *hierarchical representations* [DK90] of the separable convex hulls $CH(P)$ and $CH(Q)$ are available (we will address this in Step 4), the algorithm of [AP92] gives an $O(\log n)$ time method of merging $CH(P)$ and $CH(Q)$, using $O(n)$ processors, where $|CH(P)| = |CH(Q)| = O(n)$. In addition, the technique used to accomplish the merging process in [AP92] determines, for each edge $e \in CH(P) \cup CH(Q)$, whether or not e is an internal, external, or seam edge of $CH(CH(P) \cup CH(Q))$; these classifications can clearly be used to determine whether or not a vertex is internal, e-external or s-external to the merged hull. Thus, the merging process of [AP92] can be used to implement Step 1 of MERGE; since there are $O(m^2)$ pairwise merges of convex hulls each of size $O(n/m)$, Step 1 requires $O(\log(n/m))$ time using $O(nm)$ processors.

Since a polytope resulting from each merge in Step 1 can have size $O(n/m)$, and there are $O(m^2)$ such merges, the number of vertices in the set $A = \cup_{v \in S^*} A_v$ can be as large as $O(nm)$, where the sets A_v are as defined in Step 2; the sets A_v can be formed by a simple sorting process in time $O(\log nm)$ using $O(nm)$ processors [Co88]. Although a single set A_v can be of size $O(n)$, the CLASSIFY operations of Step 3 will require $O(\log n)$ time and $O(nm)$ processors in the aggregate because $|A| = O(nm)$. Since the neighborhood of any vertex external to $CH(S)$ was identified in Step 3, it is a trivial matter to verify that in

Step 4 a doubly connected edge list (DCEL) representation of $CH(S)$ can be constructed in constant time using $O(n)$ processors, and then a hierarchical representation of $CH(S)$ can be built optimally in $O(\log n)$ time using $O(n/\log n)$ processors on a CREW PRAM using the technique of Cole and Zajicek [CZ90].

Thus, the total complexity of the merging process is $O(\log nm)$ time using $O(nm)$ processors on a CREW PRAM, i.e., $M(m, n/m) = O(\log nm)$ or $M(n^{\alpha}, n^{1-\alpha}) = O(\log n^{1+\alpha})$ using $O(n^{1+\alpha})$ processors. Plugging this value into Equation (1) yields $T(n) = O(\log n) + O(\log n^{1+\alpha}) + T(n^{1-\alpha}) = O((1/\alpha)\log n)$ using $O(n^{1+\alpha})$ processors on a CREW PRAM, which establishes the following theorem.

Theorem 2: The convex hull of a set of n points in three-dimensional space can be computed in $O((1/\alpha)\log n)$ time using $O(n^{1+\alpha})$ processors on a CREW PRAM, for any constant $0 < \alpha \leq 1$.

4 Acknowledgement

We are grateful to Guenter Rote for his helpful comments which greatly improved the presentation of the CLASSIFY algorithm.

References

[ACGOY88] A. Aggarwal, B. Chazelle, L. Guibas, C. Ó'Dúnlaing, C. Yap, Parallel Computational Geometry, *Algorithmica* **3** (1988), pp. 293-327.

[AP92] N. Amato and F. Preparata, The Parallel 3D Convex Hull Problem Revisited, *International Journal of Computational Geometry & Applications* **2**(2) (1992), pp. 163-174.

[Co88] R. Cole, Parallel Merge Sort, *SIAM J. Comput.* **17**(4) (1988), pp. 770-785.

[CZ90] R. Cole and O. Zajicek, An Optimal Parallel Algorithm for Building A Data Structure for Planar Point Location, *Journal of Parallel and Distributed Computing* **8** (1990), pp. 280-285.

[C80] A. Chow, Parallel Algorithms for Geometric Problems, Ph.D Dissertation, Dept. of Computer Science, University of Illinois, Urbana, Illinois, (1980).

[DaK89] N. Dadoun and D. Kirkpatrick, Parallel Construction of Subdivision Hierarchies, *Journal of Computer and System Sciences* **39** (1989), pp. 153-165.

[DK90] D. Dobkin and D. Kirkpatrick, Determining the Separation of Preprocessed Polyhedra - A Unified Approach, *ICALP* (1990), pp. 400-413.

[KR91] R. Karp and V. Ramachandran, Parallel Algorithms for Shared-Memory Machines, *Handbook of Theoretical Computer Science - Volume A*, Ed. J. Van Leeuwen, Elsevier Science Publishers/The MIT Press, Amsterdam (1990), pp. 869-941.

[PDW92] W. Preilowski, E. Dahlhaus, and G. Wechsung, New Parallel Algorithms for Convex Hull and Triangulation in 3-Dimensional Space, *Proceedings MFCS*, Springer Lecture Notes in Computer Science, **629** (1992), pp. 442-450.

[PH77] F. Preparata and S. J. Hong, Convex Hulls of Finite Sets of Points in Two and Three Dimensions, *Comm. ACM* **20** (1977), pp. 87-93.

[PS85] F. Preparata and M. Shamos, *Computational Geometry*, (Springer, New York, 1985).