A Variant of Graham's Planar Convex Hull Algorithm

308-250A Fall 1999 **Course Handout** **Prof. D. Avis**

1. Computing the Convex Hull

There are several algorithms for computing the convex hull of a set of n points in the plane in $O(n \log n)$ time. The first published algorithm with this complexity was due to Ron Graham. It has two steps. In the first step, the points are sorted in angular order about some interior point of the convex hull. The second step scans the points in this order and deletes all points interior to the convex hull using a local test. There have since been several variants of this algorithm. Although conceptually very simple, there are certain technical difficulties due to collinearity of points and other "degeneracies". A simple way around this is to assume that the point set does not contain these unpleasant features and leave it to the reader (or programmer) to take care of them. Most programmers complain, however, that it is just the degeneracies and special cases that take up most of the code. Since the convex hull problem is relatively straight forward, we will try to develop as complete a procedure as possible in this section, without making any assumptions about the input data.

We describe a variant of Graham's algorithm in which the points are initially sorted lexicographically according to their first and second coordinates. The points are processed in this order updating the convex hull as each new point is added. We will see that at each step, the new point being considered is a vertex of the updated convex hull. Some other points may no longer be vertices and must be deleted from the list of convex hull vertices. In fact, the heart of the algorithm is in identifying these vertices quickly using some local tests. Figure 1 contains an example of the main step of the procedure. In Figure 1(a) a point p is added to the set of convex hull vertices $P = \{p_1, \ldots, p_7\}$. Supporting lines are drawn from p to P, and the supports *first* and *last* are identified. In case a supporting line contains an edge, as is the case of the line containing $[p_2, p_3]$, the point furthest from p is identified. In Figure 1(b), the chain between *first* and *last* is deleted and replaced by the single vertex p.

We will now describe the theoretical basis of the procedure suggested by the example, then we will discuss the implementation. Suppose we have so far obtained a set $P = \{p_1, \cdots, p_n\}$ of $n \geq 3$ points that are all convex hull vertices of P arranged in sorted clockwise order. This is a circular order, and it simplifies the description if we take indices "mod n" so that we identify p_i with p_{i+n} for all i. In this way we can say that if $n = 7$ vertex p_8 ($= p_1$) follows p_7 in the angular order. Let each point p_i have coordinates $({p_i}^1, {p_i}^2)$. We now define a lexicographic ordering "$\ll$" on P by saying that $p_i \ll p_j$ whenever

$$\text{(i) } p_i^1 < p_j^1 \text{ , or}$$

$$\text{(ii) } {p_i}^1 = {p_j}^1 \text{ and } {p_i}^2 < {p_j}^2.$$

According to this order, we may assume that the points of P have been labelled so that the first point is p_1, i.e. $p_1 \ll p_i$, $i \neq 1$. Let the last point in the order be p_j. This lexicographic ordering is compatible with the angular order of the points in the following sense:

Fact 1: $p_1 \ll p_2 \ll \cdots \ll p_j$ and $p_1 \ll p_n \ll p_{n-1} \ll \cdots \ll p_j$. □

Since we are processing points in lexicographic order, our next point p will satisfy $p \gg p_j$. Thus, for any point $p \gg p_j$ in the plane, let $P' = P \cup \{p\}$. We would like to efficiently compute

the vertices of $CH(P')$ from P, the vertices of $CH(P)$. The following fact is crucial for this computation.

Fact 2: p is a vertex of $CH(P')$.

In order to compute $CH(P')$, we need to compute two **tangents** from p to the convex hull P that we have already computed, as illustrated in Figure 1. The endpoints of the two tangent edges that originate at p indicate a (possibly empty) chain of vertices in P to be deleted. The endpoints of the tangents will be called *first* and *last* as indicated in the figure. Let p_j be defined as in Fact 1 (in Figure 1 it is p_5).

Fact 3:
(a) *first* = p_i if and only if p, p_i, p_{i-1} is a left turn and p, p_i, p_{i+1} is not a right turn.
(b) *last* = p_k if and only if p, p_k, p_{k-1} is a right turn and p, p_k, p_{k+1} is not a left turn.

In proving Fact 3 it is shown that *first* is the unique vertex of $p_1, \ldots, p_j$ satisfying condition (a), and *last* is the unique vertex of $p_j, \ldots, p_n$ satisfying condition (b). Note that conditions are exclusive, so that *first* ≠ *last*.

2. Implementing the Convex Hull Algorithm

The first step in the implementation is to choose an appropriate data structure for the convex hull vertices. From the discussion in the last section, such a data structure should keep the vertices in sorted angular order and allow for traversing the vertices in either direction. A doubly linked circular list is ideal for this purpose. At each node v we store the coordinates of the vertex it represents, along with pointers $pred(v)$ and $next(v)$ to the preceding and succeeding vertices in clockwise order around the convex hull. For simplicity, we use v to represent both the node and the point it represents. We need one additional pointer, *max*, that points to the last vertex inserted into the list. Figure 2 shows the data structure for the example in Figure 1.

We next convert the tests of Fact 3 into left and right turn tests. Let the new point be p and let v initially be set to the vertex pointed to by *max*. The test involves the four points p, v, $pred(v)$ and $next(v)$. The condition in Fact 3(a) is satisfied if and only if

$$left(p, v, pred(v)) \textbf{ and not } right(p, v, next(v)) = \textbf{true}.$$

The two cases handled by this condition are shown in Figure 3(a)(b). Fact 3(b) is handled analogously. The procedure *tangent*(p, *first*, *last*, *max*) implements the test of Fact 3. Note that in the pseudocode, the values of *first* and *last* are returned to the calling routine.

```
procedure tangent(p, first, last, max);
        v:= max;
        while ( not left(p, v, pred(v) )) or right(p, v, next(v))
                v:= pred(v);
        first:= v; v:= max;
        while ( not right(p, v, next(v) )) or left(p, v, pred(v))
                v:= next(v);
        last:= v
return;
```

Any vertices that are both successors of *first* and predecessors of *last* are deleted from the circular list, and p is inserted in this place. Because of the seqential nature of the tests performed, the above code can in fact be simplified: the second part of each "or" condition can be dropped (this is left as an exercise). We get the simplified pseudocode:

```
procedure tangent(p, first, last, max);
        v = max;
        while not left(p, v, pred(v) )
                v = pred(v);
        first = v; v = max;
        while not right(p, v, next(v) )
                v = next(v);
        last = v
return;
```

Putting all of the pieces together we have the following algorithm to find the convex hull.

algorithm *convex hull*;

Step 1. (Sort)
Sort the n input points into increasing lexicographic order and store them in an array $p_1, \ldots, p_n$.

Step 2. (Initialize)
Find the first point p_i that is not collinear with p_1 and p_2. If no such point exists then return the convex hull $\{p_1, p_n\}$. Else, construct the circular linked list for p_1, p_{i-1}, p_i.

Step 3. (Update)
For each $j = i+1, \ldots, n$ call *tangent*(p_j, *first*, *last*, p_{j-1}). Update the circularly linked list by removing all vertices that are successors of *first* and predecessors of *last*. Insert p_j between *first* and *last*.

Extracted from "Keisankikagaku-Risankikagaku", D. Avis, H.Imai and S. Matsunaga, Asakura Press, 1994.