AMS 545/CSE 555 Joe Mitchell

# Melkman's Convex Hull Algorithm

We describe an algorithm, due to Melkman (and based on work by many others), which computes the convex hull of a simple polygonal chain (or simple polygon) in linear time.

Let the simple chain be $C = (v_0, v_1, \dots, v_{n-1})$, with vertices $v_i$ and edges $v_i v_{i+1}$, etc. The algorithm uses a *deque* (double-ended queue), $D = \langle d_b, d_{b+1}, \dots, d_t \rangle$, to represent the convex hull, $Q_i = CH(C_i)$, of the subchain, $C_i = (v_0, v_1, \dots, v_i)$. A deque allows one to "push" or "pop" on the *top* of the deque and to "insert" or "remove" from the *bottom* of the deque. Specifically, to *push* $v$ onto the deque means to do ($t \leftarrow t+1$; $d_t \leftarrow v$), to *pop* $d_t$ from the deque means to do ($t \leftarrow t-1$), to *insert* $v$ into the deque means to do ($b \leftarrow b-1$; $d_b \leftarrow b$), and to *remove* $d_b$ means to do ($b \leftarrow b+1$). (A deque is readily implemented using an array.)

We establish the convention that the vertices appear in *counterclockwise* order around the convex hull $Q_i$ at any stage of the algorithm. (Note that Melkman's original paper uses a clockwise ordering.)

It will always be the case that $d_b$ and $d_t$ (the bottom and top elements of the deque) refer to the same vertex of $C$, and this vertex will always be the last one that we added to the convex hull.

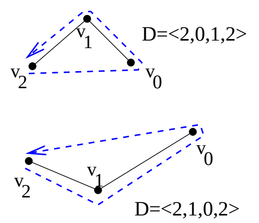
We will assume that the vertices of $C$ are in general position (no three collinear).

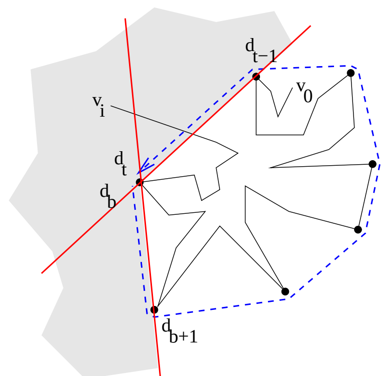
**Algorithm:**

```
(0) Initialize: If Left($v_0, v_1, v_2$), then $D \leftarrow \langle v_2, v_0, v_1, v_2 \rangle$; else, $D \leftarrow \langle v_2, v_1, v_0, v_2 \rangle$.
    $i \leftarrow 3$

(1) While (Left($d_{t-1}, d_t, v_i$) and Left($d_b, d_{b+1}, v_i$)) do
    $i \leftarrow i + 1$
    (We now have a point $v_i$ that is *not* in the convex cone defined by the two hull edges $d_b d_{b+1}$ and $d_t d_{t-1}$ (recall
    that $d_b = d_t$).)

(2) Restore convexity:
    Until Left($d_{t-1}, d_t, v_i$), do pop $d_t$. Push $v_i$.
    Until Left($v_i, d_b, d_{b+1}$), do remove $d_b$. Insert $v_i$.
    $i \leftarrow i + 1$
    Go to (1).
```

In words, the algorithm does the following: At any given stage of the algorithm, we step along the chain, examining vertices $v_i$ in the order they appear along the chain. The deque, $D$, stores the current convex hull. We step along the chain until we find the first vertex $v_i$ that lies outside of the convex hull specified by $D$: The *IMPORTANT* property of the chain being *simple* guarantees that this test (to see if $v_i$ lies inside the current hull or not) can be done very simply in *contant* time ($O(1)$) by checking to see if $v_i$ lies in the exterior (nonconvex) cone determined by the point $d_b = d_t$. (i.e., we are testing to see if $v_i$ lies in the cone (shaded red in the figure below) lying to the right of the ray $d_b d_{b+1}$ and to the left of the ray $d_{t-1} d_t$) The fact that the chain is simple implies that the only way for the chain to emerge from the convex hull $D$ is for the chain to exit the hull through one of the two edge ($d_b d_{b+1}$ or $d_{t-1} d_t$) adjacent to the point $d_b = d_t$ that last was added to $D$. (Exiting through another edge would imply that the chain crossed itself.) This is the crucial place that simplicity helps us: By localizing where the chain can "erupt" from the hull, we have a constant-time method to test if $v_i$ is *in* the hull, and, if it is not, we can do "repairs" to the hull data structure, $D$, by starting at a *known* point, $d_b = d_t$ and popping/removing entries until we restore convexity.

In the figure below, on the left we show the two possible initial conditions and the initialization of the deque, $D$. On the right, we show a generic step of the algorithm, showing the instant when a vertex $v_i$ is found to erupt into the shaded (nonconvex) cone.





**Correctness.** We claim that the algorithm correctly computes the convex hull and does so in linear time. The time bound follows easily from observing that each point $v_i$ is pushed/inserted and popped/removed at most once; thus, we can charge off the work to the vertices.

To prove correctness, we proceed by induction on the step number ($i$) of the algorithm. The induction hypothesis (IH) is: *the deque $D$, considered as a polygon, is convex and contains the subchain $C_i$ (i.e., $D$ represents $Q_i = CH(C_i)$).*

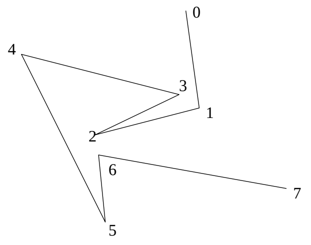
The IH holds at the beginning, when the first three vertices are considered. Now assume that IH holds after $k$ vertices $(v_0, v_1, \ldots, v_{k-1})$ have been considered. Consider now vertex $v_k$.

If vertex $v_k$ is "rejected" (passed over in the While loop), then $v_k$ lies to the left of $d_{t-1}d_t$ and to the left of $d_b d_{b+1}$ (i.e., in the complement of the shaded cone). Could it lie outside of the convex hull $Q_{k-1}$? No, by simplicity of the input chain: The input chain connects $d_{b+1}$ with $d_{t-1}$; this subchain, together with the two edges $d_{t-1}d_t$ and $d_b d_{b+1}$, forms a simple closed curve that must contain $v_i$. (Otherwise, by the Jordan curve theorem, the segment $v_{k-1}v_k$ must cross the boundary of the closed simple curve, which would imply a self-crossing in the input chain.) Thus, if $v_k$ is rejected it must lie within the convex polygon described by $D$.

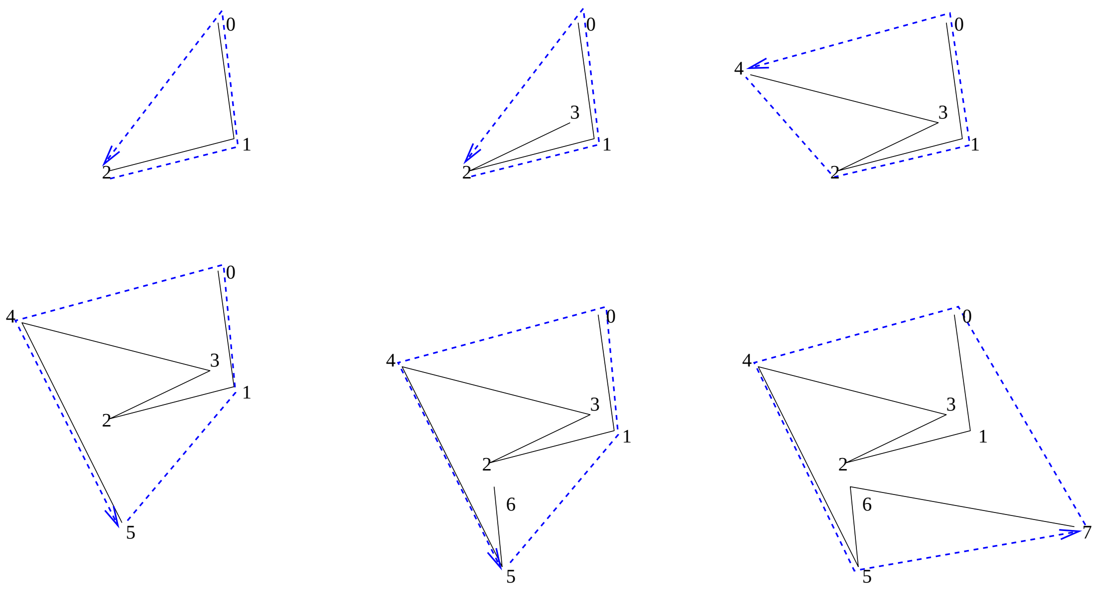
So consider the case that $v_k$ is not rejected (lies in the shaded region). Then, we just have to check that the steps that are done to restore convexity work properly: we do not throw away vertices of the new convex hull, and we do throw away vertices (during popping and removing) that are interior to the new hull. This is easy to check.

(For details, see the paper, "On-line construction of the convex hull of a simple polyline," Avraham A. Melkman, *Information Processing Letters* 25 (1987), 11-12.)

**An Example.** Consider executing Melkman's convex hull algorithm on the vertices of the polygonal chain below, in the order $v_0$, $v_1$, $v_2$, $v_3$, etc. (In the figure, I label $v_i$ with "$i$".) Show the deque, indicating the "top" $d_t$ and "bottom" $d_b$ at the instant just after having computed the hull of the first 6 vertices ($v_0$–$v_5$), the first 7 vertices, and the first 8 vertices.



*Solution:* When we execute Melkman's convex hull algorithm on the vertices of $P$ in the order $v_0$, $v_1$, $v_2$, $v_3$, etc., we get a deque, $D = < d_b, ..., d_t >$. At the beginning (after the first 3 points are considered), the deque is simply $< 2, 1, 0, 2 >$. It then becomes $< 2, 1, 0, 2 >$, then $< 4, 2, 1, 0, 4 >$, then $< 5, 1, 0, 4, 5 >$, then $< 5, 1, 0, 4, 5 >$, then $< 7, 0, 4, 5, 7 >$.


0
1
2
0
3
1
2
4
0
3
1
2
0
4
3
1
2
5
0
4
3
1
2
6
5
0
4
3
1
2
6
7
5