Moldable and Castable Polygons*

Arnold Rosenbloom and David Rappaport†

December 21, 1993

Abstract

This paper introduces the concepts of Moldability and Castability of simple polygons and relates Moldability to Monotonicity. We detail a $\theta(n)$ algorithm for determining all n forward maximal monotone chains of a simple polygon and apply this algorithm to the problems of determining 2-Moldability, 2-Castability and the minimum monotone decomposition of a simple polygon [9]. Our results include a simple optimal algorithm solving the minimum monotone decomposition problem, an optimal algorithm to determine 2-Moldability and an $O(n \log n)$ algorithm to determine 2-Castability.

Keywords: monotonicity, decomposition, moldability, castability

1 Introduction

A statue can be created by pouring wet cement into a mold, allowing the cement to harden, then removing the mold. If as well, the two pieces of the mold can be removed by translation then the resulting statue is 2-Moldable. Alternatively, we can form the statue by taking two cast pieces, lie them on their sides, fill them up with cement and after the cement hardens glue the pieces together along the flat sides. Of course we still want to remove the cast pieces without breaking them. A statue that can be created in such a way is a 2-Castable statue.

This paper introduces the notions of Moldability and Castability of simple polygons and investigates the 2-Moldability/2-Castability of simple polygons. Our results include theory relating 2-Moldability to 2-Monotonicity and a $\theta(n)$ time and space algorithm for determining all forward maximal monotone chains of a simple polygon. Two immediate applications of this algorithm are determining the 2-Moldability of a polygon and a minimum monotone decomposition[9] for a polygon. In both cases we exhibit a $\theta(n)$ algorithm. Finally we use the results of [3] to obtain an $O(n \log n)$ algorithm for determining 2-Castability.

*This research was supported in part by the Natural Sciences and Engineering Research Council of Canada under grant A-9204.

†Computing and Information Science, Queens University, Kingston Ontario K7L 3N6 email: arnold@cs.toronto.edu daver@qucis.queensu.ca

The concept of monotonicity of chains of a polygon will play an important role in the development of algorithms for recognizing moldable sets. In [6] an $\theta(n)$ time and space algorithm is given for determining if a polygon is monotone. In [5] a simple $\theta(n)$ algorithm is given which triangulates a monotone polygon. In [9] an $\theta(n)$ time and space algorithm is given (using the results of [4]) which determines the minimum number of monotone chains into which a given polygon can be decomposed. We provide a straightforward algorithm which solves the same problem in the same time and space bounds.

2 Notation

By a direction in $\mathbb{R}^2$ we mean a non-origin point which will be used to determine changes in position. By $\overset{d}{\overrightarrow{z}}$ we mean the ray originating at z and parallel to the direction d. Given 2 points p, q by the ray $\overrightarrow{pq}$ we mean the set of points on the half line based on p and including q. For any $X \subseteq \mathbb{R}^2$ $\partial X, X^o, X^c, \overline{X}$ will denote the boundary of X, the interior of X, the complement of X and the closure of X. By an ϵ neighborhood of z we mean the set of points less than ϵ distance from z. $\mathbb{Z}_n$ denotes the integers modulo n and $\mathbb{Z}_n^{i,j}$ denotes the set of equivalence classes from i to j modulo n (ie. $\{i \bmod n, (i+1) \bmod n, \ldots, j \bmod n\}$). A **polygon** is a finite sequence of segments (edges) which intersect only at their end points (the vertices) and only consecutive edges intersect.

> **Polygon Convention :** The vertices are ordered so that the interior of the polygon lies to the left of the edges.

That is, if z is internal to some edge of the polygon P then there is a neighborhood of z so that all points in the neighborhood and to the left of the edge are contained in P^o.

Given 2 points $p, q \in \partial P$ by $(p \ldots q)$ we mean the set of boundary points of P between p and q in the natural direction. $[p \ldots q] = (p \ldots q) \cup \{p, q\}$. If v and v' (appearing in this order) are sequential vertices of the chain $[p \ldots q]$ then $v' - v$ is a vector induced by $[p \ldots q]$. Note that we consider p and q vertices of $[p \ldots q]$.

2-Moldability will be in some sense the most general type of moldability. It corresponds to being able to fill up a set while a mold is in place and then being able to remove the mold by translation without breaking it.

We will say that the chain $[p \ldots q]$ of polygon P is **removable** in direction d precisely when $\forall z \in [p \ldots q]$ $\overset{d}{\overrightarrow{z}} \cap P^o = \emptyset$.

> **Definition :** A polygon P is **2-Moldable** when there are chains $[p \ldots q], [q \ldots p]$ and directions d_{pq}, d_{qp} such that $[p \ldots q]$ is removable in direction d_{pq} and $[q \ldots p]$ is removable in direction d_{qp}.

An obvious extension of the above definition is to the definition of k-Moldability. Polygon P is **k-Moldable** if there is some partition of the boundary into k pieces such that each piece is removable in some direction.

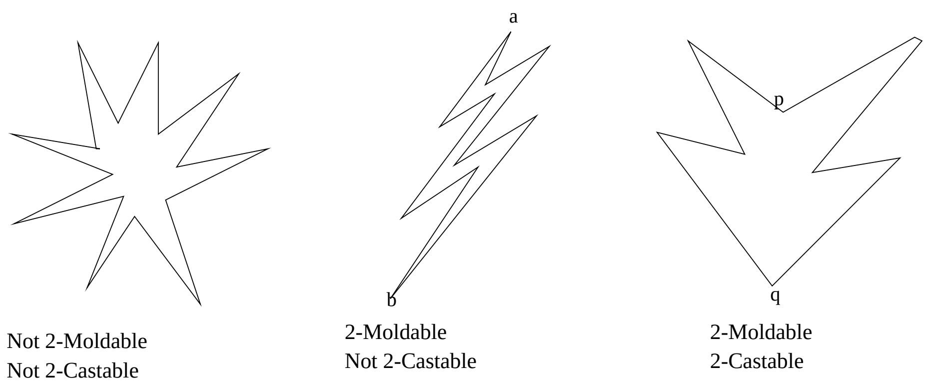

Figure 1: Some example polygons

3 Polygons and Monotonicity

A chain $[p \dots q]$ of a polygon P is **monotone with respect to** L if there is some orientation of L so that the ordering of the projection of the vertices in $[p \dots q]$ on L agrees with the ordering of the vertices themselves. The chain $[p \dots q]$ is **monotone** if there is some line L so that $[p \dots q]$ is monotone with respect to L. A **monotone decomposition** of a polygon is a decomposition of its boundary into disjoint (except at the endpoints) monotone chains. A polygon is k-Monotone if there is a monotone decomposition of the polygon consisting of k chains.

It should be obvious that a k-Moldable polygon is also a k-Monotone polygon, but the converse is not in general true. The polygon in figure 2 is 3-monotone but is not k-moldable for any k. To see this, note that the segment $\overline{pq}$ can not be translated indefinitely in any direction without intersecting the interior.

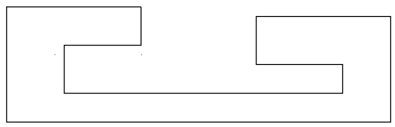

Figure 2: A 3-Monotone polygon which is not k-Moldable

4 Towards a Computable Characterization of 2-Moldability

Although general k-moldability and k-monotonicity are not equivalent, we can establish a relationship for the special case $k = 2$.

Lemma 1 : If $[p \dots q]$ is removable in direction d then $[p \dots q]$ is monotone with respect to a line L with L perpendicular to d.

Corollary 2 : If polygon P is k-Moldable then it is k-Monotone.

Lemma 3 : If $[p \dots q]$ is monotone with respect to a line L_p and $[q \dots p]$ is monotone with respect to a line L_q then there are directions d_{pq} and d_{qp}, perpendicular to L_p and L_q respectively, so that $[p \dots q]$ is removable in direction d_{pq} and $[q \dots p]$ is removable in direction d_{qp}.

Proof: Without loss of generality let p and q lie on a vertical line L with q above p. We are free to orient the directions d_{pq} and d_{qp} so that d_{pq} points to the right and d_{qp} points to the left of L. Since the chain $[p \dots q]$ is monotone with respect to a line perpendicular to d_{pq}, it follows that $[p \dots q]$ is constrained to lie within a strip bounded by lines through $\overset{d_{pq}\rightarrow}{p}$ and $\overset{d_{pq}\rightarrow}{q}$. Similarly $[q \dots p]$ is constrained to lie within a strip bounded by the lines through $\overset{d_{qp}\rightarrow}{p}$ and $\overset{d_{qp}\rightarrow}{q}$. As a result no part of P can cross L below the point p or above the point q.

We show that $\overset{d_{pq}\rightarrow}{z}$ does not intersect P^o for all z in $[p \dots q]$. For suppose that $\overset{d_{pq}\rightarrow}{z}$ does intersect P^o. If $\overset{d_{pq}\rightarrow}{z}$ intersects P^o then $\overset{d_{pq}\rightarrow}{z}$ must also intersect $(q \dots p)$ because $[p \dots q]$ is monotone with respect to a line perpendicular to direction d_{pq}. Let x denote a point on $\overset{d_{pq}\rightarrow}{z}$ intersecting $(q \dots p)$. There is a continuous path from x to p in $[q \dots p]$ and a continuous path from x to q in $[q \dots p]$. However, one of these paths must either cross L below p or above q, a contradiction. Therefore, $\overset{d_{pq}\rightarrow}{z}$ cannot intersect $(q \dots p)$ and we conclude that $[p \dots q]$ is removable in direction d_{pq}. A similar symmetric argument can be used to show that $[q \dots p]$ is removable in direction d_{qp}. $\square$

Note that if a polygon is 2-Monotone (and so 2-Moldable) then there are a pair of vertices which determine a 2-Monotone (and hence a 2-Moldable) decomposition for the polygon. So in order to determine the 2-Moldability of a polygon we can examine vertex pairs v_i, v_j to see if both $[v_i \dots v_j]$ and $[v_j \dots v_i]$ are monotone. Unfortunately, this naive algorithm has complexity at least $O(n^2)$. We develop a linear time algorithm in the next section.

5 Computing Forward Maximal Monotone Chains

This section details a $\theta(n)$ time and space algorithm for determining all forward maximal monotone chains for a given polygon. As well as being of interest in its own right, this algorithm will be used to develop $\theta(n)$ algorithms for determining 2-Monotonicity, 2-Moldability and a minimum monotone decomposition of a simple polygon. A further application of our work here will be in establishing a $O(n \log n)$ algorithm for determining 2-Castability.

5.1 Computing $C[]$

A chain $[p \dots q]$ is **forward maximally monotone** if it is monotone but any extension of the chain (at the q end) produces a non-monotone chain. Our efforts throughout this section will be in detailing an algorithm which, given a polygon as a sequence of its n vertices, determines the array $C[]$ which describes all the forward maximal monotone chains for the polygon. That is, $C[]$ satisfies $C[i] = j$ if and only if $[v_i \dots v_j]$

is a forward maximal monotone chain. We arrive at the algorithm **Compute C[]** by noticing that there is an intimate connection between the monotonicity of a chain $[p \ldots q]$ and the directions determined by edges in the chains. The connection allows us to rephrase our problem of determining all forward maximal monotone chains to a problem about points on a circle. An optimal solution to this final problem is obtained by noticing some properties of the problem structure which allow a simple linear time algorithm.

> **Lemma 4 :** A chain $[p \ldots q]$ of P is Monotone if and only if all vectors induced by the chain lie in some closed half-plane through the origin.

By the **unit circle** $\bigcirc$ we will mean $\{(\cos(\theta), \sin(\theta)) | \theta \in \mathbb{R}\}$. For $a, b \in \bigcirc$, by the **angle from** a **to** b, written $\angle ab$, we will mean the set of points of $\bigcirc$ from a counterclockwise to b. The **acute angle between** a **and** b, written $\angle\angle ab$, will be the angle created by the smaller of $\angle ab$ and $\angle ba$. We will denote the size of $\angle ab$ by $|\angle ab|$. The reader should realize that there is a natural mapping between the real interval $[0, 2\pi)$ and points on the unit circle. Either representation may be used but our algorithms will assume the $[0, 2\pi)$ representation. The array $C[]$ is computed by mapping edges $\overline{v_i v_{i+1}}$ to points $c_i = \overrightarrow{0 d_i} \cap \bigcirc$ (where $d_i = v_{i+1} - v_i$) and solving the equivalent problem.

Instance : A set of points $\{c_0, \ldots, c_{n-1}\}$ on the unit circle which satisfies

1. $0 < |\angle\angle c_i c_{i+1}| < \pi$
2. The complete set $\{c_0, \ldots, c_{n-1}\}$ does not fit in any angle of size π

Problem: For each $i \in 0, \ldots, n-1$ determine $P[i]$ which satisfies:

1. The set of points $\{c_i, \ldots, c_{P[i]}\}$ fits in some angle of size π.
2. The set of points $\{c_i, \ldots, c_{P[i]}, c_{P[i]+1}\}$ does not fit in any angle of size π.

The above restrictions on the instance arise as a result of the mapping from sequential vertex differences to points on the unit circle.

$\mathbb{Z}_n$ denotes the integers modulo n and $\mathbb{Z}_n^{i,j}$ denotes the set of equivalence classes from i to j modulo n (ie. $\{i \bmod n, (i+1) \bmod n, \ldots, j \bmod n\}$). In what follows $i, j, a, b \in \mathbb{Z}_n$. For $a \neq b$ $\angle_a^b = \cup_{k \in \mathbb{Z}_n^{a,b-1}} \angle\angle c_k c_{k+1}$ and $U_a^b = \{c_k | k \in \mathbb{Z}_n^{a,b}\}$. It should be clear that U_a^b fits in an angle of size π if and only if $\angle_a^b$ does and $\angle_a^b$ is the smallest angle containing U_a^b. An easy induction establishes that $\angle_a^b \neq \bigcirc \Rightarrow \angle_a^b = \angle cc'$ for some $c, c' \in U_a^b$.

It is important to note that we are distinguishing points c_i, c_j not only as points of $\bigcirc$ but also by their indices. So for example, if $i \neq j$ then c_i and c_j might represent the same point of $\bigcirc$ but we will say they are different. To be precise, one should think of each c_i as an ordered pair (θ_i, i) where θ_i is a point on $\bigcirc$ and $i \in \mathbb{Z}_n$. We will say that c_j **dominates** c_i with respect to $\mathbb{Z}_n^{a,b}$ if j appears after i in the sequence $< a, a+1, \ldots, b >$.

Finally, we define the **first** and **last** members of a subset S of U_a^b as follows. $first(S)$ is the most dominant member of the set $\{c \in S | \forall d \in S \angle cd \subseteq \angle_a^b\}$. Similarly, $last(S)$ is the most dominant member of $\{c' \in S | \forall d \in S \angle dc' \subseteq \angle_a^b\}$. Thus $\angle first(U_a^b) last(U_a^b) = \angle_a^b$.

Our algorithm uses the quantities $first$ and $last$ to keep track of $\angle_a^b$. For the sake of algorithmic efficiency, we use a subset $\mathcal{L}_a^b \subseteq U_a^b$, so that $first(U_a^b) = first(\mathcal{L}_a^b)$ and $last(U_a^b) = last(\mathcal{L}_a^b)$ as points on the circle.

More formally

ALGORITHM : Compute $P[]$

$\mathcal{L} \leftarrow \{c_0, c_1\}$
$i \leftarrow 0$ $j \leftarrow 1$
while ($i < n$) **do**
  while ($|\angle first(\mathcal{L}) last(\mathcal{L})| \leq \pi$) **do**
    $j \leftarrow j + 1 \bmod n$
    $\mathcal{S} \leftarrow \{c_k \in \mathcal{L} : c_k \in \angle c_{j-1} c_j\}$
    $\mathcal{L} \leftarrow \mathcal{L} - \mathcal{S} \cup \{c_j\}$
  end while
  if ($first(\mathcal{L}) = c_j$) **then** $k \leftarrow index(last(\mathcal{L}))$ **else** $k \leftarrow index(first(\mathcal{L}))$
  while ($i \leq k$) **do**
    $P[i] \leftarrow j - 1$
    $\mathcal{L} \leftarrow \mathcal{L} - \{c_i\}$
    $i \leftarrow i + 1$
  end while
end while
end Compute

5.1.1 Analysis and Correctness

We begin by proving the correctnes of Algorithm Compute P. We will use the notation $\mathcal{L}_a^b$ to denote the state of the set $\mathcal{L}$ when $i = a$ and $j = b$.

We must show that $\angle_a^b = \angle first(\mathcal{L}_a^b) last(\mathcal{L}_a^b)$.

Lemma 5 : If $\angle first(\mathcal{L}_a^b) last(\mathcal{L}_a^b) = \angle first(U_a^b) last(U_a^b) = \angle_a^b$ then $\angle first(\mathcal{L}_a^{b+1}) last(\mathcal{L}_a^{b+1}) = \angle first(U_a^{b+1}) last(U_a^{b+1}) = \angle_a^{b+1}$.

Proof: Recall that $\mathcal{L}_a^{b+1} = \mathcal{L}_a^b - \{c_k \in \mathcal{L}_a^b : c_k \in \angle c_{b-1} c_b\} + \{c_{b+1}\}$. We consider two cases. Suppose $c_{b+1} \in \angle_a^b$ then $first(\mathcal{L}_a^{b+1}) = first(\mathcal{L}_a^b) = first(U_a^b) = first(U_a^{b+1})$ and $last(\mathcal{L}_a^{b+1}) = last(\mathcal{L}_a^b) = last(U_a^b) = last(U_a^{b+1})$

and we are done. On the other hand suppose that $c_{b+1} \notin \angle_a^b$. In this case c_{b+1} either becomes the new value for $first(\mathcal{L}_a^{b+1})$ or $last(\mathcal{L}_a^{b+1})$. Either way we maintain $first(\mathcal{L}_a^{b+1}) = first(U_a^{b+1})$ and $last(\mathcal{L}_a^{b+1}) = last(U_a^{b+1})$. □

Lemma 6 : If $\angle first(\mathcal{L}_a^b) last(\mathcal{L}_a^b) = \angle first(U_a^b) last(U_a^b) = \angle_a^b$ then $\angle first(\mathcal{L}_{a+1}^b) last(\mathcal{L}_{a+1}^b) = \angle first(U_{a+1}^b) last(U_{a+1}^b) = \angle_{a+1}^b$.

Proof: Recall that $\mathcal{L}_{a+1}^b = \mathcal{L}_a^b - \{c_a\}$. There are two cases to consider. If $c_a \in \angle_a^b$ then we are done. Otherwise when we remove c_a from $\mathcal{L}_a^b$ we will obtain a new value for $first(\mathcal{L}_{a+1}^b)$ or $last(\mathcal{L}_{a+1}^b)$. We first consider the case when $c_a = first(\mathcal{L}_a^b) = first(U_a^b)$. Use c_x to denote the value of $first(U_{a+1}^b)$. We must demonstrate that $c_x \in \mathcal{L}_a^b$. Any element c_k an element of U_a^b and not an element of $\mathcal{L}_a^b$ has the property that $c_k \in \angle c_{j-1}c_j$, and c_k is encountered prior to c_{j-1} in the while loop of algorithm Compute $P[]$. Since c_x is adjacent to c_a the only pair that contains c_x in U_a^b is $\angle c_a c_{a+1}$, but $x \notin \mathbb{Z}_n^{1,a}$ so c_x was not encountered prior to c_a in the while loop of algorithm Compute $P[]$. Therefore, c_x must also be a member of $\mathcal{L}_a^b$. We conclude that $c_x \in \mathcal{L}_{a+1}^b$ and that $first(\mathcal{L}_{a+1}^b) = first(U_{a+1}^b)$. A similar symmetric argument can be made to show that $last(\mathcal{L}_{a+1}^b) = last(U_{a+1}^b)$. □

Consider the initial conditions of $a = 0$ and $b = 1$, and observe that $\angle_a^b = \angle first(\mathcal{L}_a^b) last(\mathcal{L}_a^b)$. Thus we can bootstrap an inductive arguement based on the previous two lemmas to prove the correctness of algorithm Compute P[]. We now discuss the complexity of this algorithm.

Lemma 7 : Algorithm Compute P[] can be implemented in $O(n)$ time and space.

Proof: The key to an efficient implementation of Compute P is to use a suitable data structure to represent the set $\mathcal{L}$. We maintain the values of $\mathcal{L}$ in a doubly linked list sorted according to their $\angle$ values. That is, following an elements forward link leads to that elements nearest counterclockwise neighbour. The list is also maintained so that $first(\mathcal{L})$ and $last(\mathcal{L})$ appear as the first and last element of the list. The principle difficulty in obtaining the linear running time is in updating the representation of $\mathcal{L}_i^j$ to $\mathcal{L}_i^{j+1}$, and updating $\mathcal{L}_i^j$ to $\mathcal{L}_{i+1}^j$. We address the problem of updating $\mathcal{L}_i^j$ to $\mathcal{L}_i^{j+1}$ first. To remove the set $\{c_k \in \mathcal{L} : c_k \in \angle c_j c_{j+1}\}$ we traverse the list from the location of c_j to the location of c_{j+1} deleting every element we encounter. Furthermore we keep track of the location of the most recent addition to $\mathcal{L}$. Since any time we traverse an element we also delete it, the accumulated effort throughout the life of the algorithm is $O(n)$.

In updating $\mathcal{L}_i^j$ to $\mathcal{L}_{i+1}^j$ we must determine the location of c_i in $\mathcal{L}$. We claim that c_i is either $first(\mathcal{L})$, $last(\mathcal{L})$, or not in $\mathcal{L}$. To see why this so, consider c_i not first or last such that $c_x = first(\mathcal{L})$ and $c_y = last(\mathcal{L})$ and $i < x < y$. (Note we are only concerned with the fact that $i < x$ and that $i < y$, the relative ordering of x and y is not important, we have picked arbitrarily.) Since $x < y$ then at some time in the process of updating the set $\mathcal{L}$ we scanned the circular list from c_x to c_y. Thus for all elements c_i such that $i < x < y$, we know that c_i cannot be in $\mathcal{L}$. Therefore, the cost that has been incurred is bounded from above by a constant.

Thus both updates are taken care of. We are obviously within the $O(n)$ bound for space complexity. Thus we conclude that algorithm Compute P[] has a time and space complexity in $O(n)$. □

Returning to our original problem of computing maximal monotone chains observe that it follows from lemma 4 that C[i] = P[i]+1. Coupled with our algorithm analysis the previous statement leads to the main result of this section.

> **Theorem 8 :** Algorithm Compute P[] can be used to compute maximal monotone chains in $O(n)$ time and space.

We will refer to the algorithm in the above theorem as **Compute** $C[]$.

5.2 Application 1: Determining 2-Moldability of a Polygon

We can now use **Compute** $C[]$ to determine if a polygon is 2-Moldable/2-Monotone.

ALGORITHM: Is 2-Moldable?

for $i \in < 0, \ldots, n-1 >$

  if $C[C[i]]$ appears in $i, i+1, \ldots, C[i]$(all numbers taken modulo n) then result(Moldable)

end for

result(NotMoldable)

> **Theorem 9 :** There is a linear time and space algorithm which, given a polygon P as an array $v[]$ of n vertices, determines if P is 2-Moldable and if so returns a 2-Mold for P.

5.3 Application 2: Minimum Monotone Decomposition

We can also apply our $C[]$ algorithm to the problem of [9] and solve the minimum monotone decomposition problem. That is, given a polygon as a sequence of vertices, determine its minimum (in number of chains) monotone decomposition. The algorithm presented below as well as our Compute P[] algorithm first appeared in [7][1] and was later adopted as an improvement to [8] in the paper [9].

Consider the problem of finding a minimum cover for a circle given the set of n arcs $\angle z_i z_{C[i]}$ where $z_i \in \bigcirc$ has coordinates $(\cos(i2\pi/n), \sin(i2\pi/n))$. We can use a queue to provide the algorithm of [4] with an ordered sequence of arc endpoints and find a minimum cover $\angle z_{i_1} z_{C[i_1]}, \ldots, \angle z_{i_m} z_{C[i_m]}$ for $\bigcirc$. We claim that this minimum circle cover corresponds to a minimum monotone decomposition of P and that the algorithm takes linear time and space, the same bounds as [9]. It should be noted that some variant of the algorithm of [9] might be used to decide 2-Moldability.

We compare our results with the results of [9] by comparing both as algorithms which solve the Minimum Monotone Decomposition problem. The approach in [9] is to suggest that once all maximal monotone chains are obtained in sorted order then the algorithm of [4] solves the decomposition problem in linear time. They then break the universe of all polygons into 2 disjoint sets. The first admits a simple greedy solution to the decomposition problem. For the second set, they supply an algorithm (which applies only to this set) for determining all maximal monotone chains in sorted order. Once this is established, the comments regarding

[9] implies the linear time result for polygons in the second set. These two results, together with an algorithm for determining membership in the sets, form their decomposition algorithm.

Our approach does not distinguish polygons. For any polygon, we compute all maximal monotone chains and then use the algorithm of [4] to supply the minimum decomposition.

6 2-Castability

Another way to create a statue is to lie two mold pieces on their sides, fill the pieces with cement then, after the cement hardens, pull off the mold and glue the two pieces together on their flat sides. In two dimensions the above intuition implies that we should classify a polygon as 2-Castable if it is 2-Moldable with mold pieces $[p \ldots q]$ and $[q \ldots p]$ and the segment $\overline{pq}$ stays strictly inside the polygon. More formally

> **Definition :** A polygon P is **2-Castable** if $\exists p, q \in \partial P$ such that $[p \ldots q]$ and $[q \ldots p]$ are removable and $\overline{pq} - \{p, q\} \subset P^o$.

> **Definition :** For $p, q \in P$ we say that p **sees** q whenever $\overline{pq} - \{p, q\} \subset P^o$.

It should be noted that while any 2-Moldable polygon has a 2-Mold which is determined by a pair of vertices, the same can't be said of 2-Castable polygons. That is, there are 2-Castable polygons for which no pair of vertices determine a 2-Cast.

6.1 Approach

Our purpose here is to establish an $O(n \log n)$ time and $O(n)$ space algorithm which decides the 2-Castability of a polygon. Briefly, the algorithm proceeds by walking 2 chains around P. At each main step of the algorithm, it is determined whether the 2 chains determine a 2-Mold for the polygon (ie. they in some way decompose the polygon into a pair of monotone chains). If this is the case then the algorithm determines whether the chains (in some way) determine a 2-Cast. Determining the 2-Castability of the polygon at this stage amounts to determining the visibility status of certain sets of points on the boundary of the polygon. To this end, the algorithm maintains a pair of convex hulls associated with the chains and ascertains the visibility status of the points in questions by determining the intersection status of the convex hulls.

6.2 Chain Definition and 2-Moldability

A chain $[p \ldots q]$ is **backward maximally monotone** if it is monotone but any extension at the p end produces a non-monotone chain For a given polygon P, the array $B[]$ satisfies $[v_{B[i]} \ldots v_i]$ is a backwards maximal monotone chain. We will refer to $[v_{B[j]} \ldots v_j]$ and $[v_i \ldots v_{C[i]}]$ as **the back chain at** v_j and **the forward chain at** v_i respectively.

Given a polygon $P =< v_0, \ldots, v_{n-1} >$, we can reverse the roles of left and right and run algorithm **Compute** $C[]$ on the sequence $< v_{n-1}, v_{n-2}, \ldots, v_0 >$. The result of such a process is the array $B[]$. The time and space bounds remain unchanged.

We characterize 2-Moldability in terms of the chains defined above.

Lemma 10 : $\forall p, q \in \partial P$ we have

$$p, q \text{ determine a 2-Mold for } P$$

$$\Leftrightarrow$$

$$\exists i \in \mathbb{Z}_n \text{ such that at least one of the following holds}$$

$$p \in \overline{v_i v_{i+1}} - \{v_i, v_{i+1}\}, \partial P = [v_i \ldots v_{C[i]}] \cup [v_{B[i+1]} \ldots v_{i+1}], q \in [v_{B[i+1]} \ldots v_{C[i]}]$$

$$\text{or}$$

$$p = v_i, \partial P = [v_i \ldots v_{C[i]}] \cup [v_{B[i]} \ldots v_i], q \in [v_{B[i]} \ldots v_{C[i]}]$$

Proof: We will establish the result in the case $p = v_i$, the other case has a similar proof.
$\Rightarrow$: Assume p, q determines a 2-Mold for P and $p = v_i$ is a vertex of P. Since $[p \ldots q]$ is monotone we have $[p \ldots q] \subseteq [(p = v_i) \ldots v_{C[i]}]$. Similarly $[q \ldots p] \subseteq [v_{B[i]} \ldots v_i]$ so

$$\partial P = [p \ldots q] \cup [q \ldots p] \subseteq [v_i \ldots v_{C[i]}] \cup [v_{B[i]} \ldots v_i] \subseteq \partial P$$

and $\partial P = [v_i \ldots v_{C[i]}] \cup [v_{B[i]} \ldots v_i]$. Now $\partial P - \{v_i\}$ is connected so

$$[v_{B[i]} \ldots v_i) \cap (v_i \ldots v_{C[i]}] \neq \emptyset$$

and in fact

$$[v_{B[i]} \ldots v_i) \cap (v_i \ldots v_{C[i]}] = [v_{B[i]} \ldots v_{C[i]}]$$

Note that $q \in [v_{B[i]} \ldots v_i)$ and $q \in (v_i \ldots v_{C[i]}]$ so $q \in [v_{B[i]} \ldots v_{C[i]}]$.
$\Leftarrow$: Assume

$$\partial P = [v_i \ldots v_{C[i]}] \cup [v_{B[i]} \ldots v_i], p = v_i, q \in [v_{B[i]} \ldots v_{C[i]}]$$

then, as before, $\partial P - \{v_i\}$ is connected therefore $[v_{B[i]} \ldots v_i) \cap (v_i \ldots v_{C[i]}]$ is non-empty and in fact equals $[v_{B[i]} \ldots v_{C[i]}]$. Therefore $q \in [v_{B[i]} \ldots v_i)$ and $q \in (v_i \ldots v_{C[i]}]$ from which we conclude $[q \ldots p]$ and $[p \ldots q]$ are monotone, hence P is 2-Moldable with a 2-Mold determined by p, q. $\Box$

This Lemma tells us that we can completely characterize the set of points which determine a 2-Mold for a polygon by the following linear time algorithm.

ALGORITHM : Characterize Polygon

Compute $B[]$

Compute $C[]$

for $i \leftarrow 0$ to $n-1$

  for $k \leftarrow 0$ to 1

    $j \leftarrow i + k \bmod n$

    if $\partial P = [v_i \ldots v_{C[i]}] \cup [v_{B[j]} \ldots v_j]$ then

      all pairs (p, q) with $p \in [v_i \ldots v_j], q \in [v_{B[j]} \ldots v_{C[i]}]$ determine a 2-Mold for P

    end if

  end for

end for

6.3 Determining Visibility

We can use the above algorithm for determining 2-Castability if we can determine whether any points in $[v_i \ldots v_j]$ see any points in $[v_{B[j]} \ldots v_{C[i]}]$. To accomplish this, assume we have the convex hulls of $[v_{C[i]} \ldots v_i]$ and $[v_j \ldots v_{B[j]}]$ (referred to as CH_i and CH_j respectively). With CH_i and CH_j in hand, we can determine visibility between points of $[v_i \ldots v_j]$ and points of $[v_{B[j]} \ldots v_{C[i]}]$ by determining the intersection status of CH_i and CH_j. The relationship between intersection and visibility will depend on whether $i = j$ or $j = i+1$ and whether $[v_{B[j]} \ldots v_{C[i]}]$ is a single point or not. The following lemma establishes the relationship in the case $i = j$ and $[v_{B[j]} \ldots v_{C[i]}]$ is not a single point. Of course, we are assuming $[v_{B[j]} \ldots v_{C[i]}]$ has the proper sense since we will be testing visibility only if the forward chain at i and the back chain at j cover the boundary of P.

Lemma 11 : Assume $i = j$ (ie. $[v_i \ldots v_j] = \{v_i\}$) and $(v_{B[i]} \ldots v_i) \cap (v_i \ldots v_{C[i]}) \neq \emptyset$ then CH_i and CH_j intersect precisely at v_i if and only if some point of $(v_{B[j]} \ldots v_{C[i]})$ sees v_i.

Proof: $\Rightarrow$: Consider a line L through v_i which places CH_i and CH_j in different half planes. Since

$$\partial P - \{v_i\} = (v_i \ldots v_{B[j]}] \cup (v_{B[j]} \ldots v_{C[i]}) \cup [v_{C[i]} \ldots v_i)$$

is connected we must have $(v_i \ldots v_{B[j]}] \cup (v_{B[j]} \ldots v_{C[i]}) \cup [v_{C[i]} \ldots v_i) \cap L \neq \emptyset$. Now since L separates CH_i and CH_j we have $(v_i \ldots v_{B[j]}] \cap L = [v_{C[i]} \ldots v_i) \cap L = \emptyset$ so $(v_{B[j]} \ldots v_{C[i]}) \cap L \neq \emptyset$.
$\Leftarrow$: Assume $\exists q \in (v_{B[j]} \ldots v_{C[i]})$ with $\overline{v_i q} - \{v_i, q\} \subset P^o$, we must show that CH_i and CH_j do not intersect except at v_i. Notice that it is sufficient to display a line L which separates the chains $(v_i \ldots v_{B[j]}]$ and $[v_{C[i]} \ldots v_i)$. Let $L = \overleftrightarrow{v_i q}$, $z \in L - \overline{v_i q}$, then either $q \in \overline{v_i z}$ or $v_i \in \overline{qz}$. In the first case, if $z \in \partial P$ then any non-trivial parallel projection of L onto any line L' places the image of q between the image of v_i and the image of z contradicting the monotonicity of $[v_i \ldots q]$ or $[q \ldots v_i]$ so no such

z exists. The second case is handled similarly. To obtain our result, notice that an application of the Jordan Curve Theorem places the chains $(v_i \ldots q)$ and $(q \ldots v_i)$ on opposite sides of L. $\square$

Lemma 12 : Let $j \in \mathbb{Z}_n^{i,i+1}$, then if $v_{B[j]}$ or $v_{C[i]}$ sees v_i then so does some point of $(v_{B[j]} \ldots v_{C[i]})$.

Together, these two Lemmas tell us that in order to establish if v_i sees any point of $[v_{B[j]} \ldots v_{C[i]}]$, we must simply check whether CH_i and CH_j intersect just at v_i.

We offer, without proof, the three remaining properties relating visibility to hull intersection.

Lemma 13 :

1. $j = i$ and $[v_{B[j]} \ldots v_{C[i]}]$ is a single point, then v_i sees $v_{C[i]} \Leftrightarrow CH_i$ and CH_j both have the edge $\overline{v_i v_{C[i]}}$ but with opposite orientations.
2. $j = i + 1 \bmod n$ and $[v_{B[j]} \ldots v_{C[i]}]$ is not a single point, then some point of $[v_i \ldots v_j]$ sees some point of $[v_{B[j]} \ldots v_{C[i]}] \Leftrightarrow CH_i$ and CH_j do not intersect.
3. $j = i+1 \bmod n$ and $[v_{B[j]} \ldots v_{C[i]}]$ is a single point, then some point of $[v_i \ldots v_j]$ sees $v_{C[i]} \Leftrightarrow CH_i$ and CH_j intersect precisely at $v_{C[i]}$.

6.4 Background Algorithms

One way to arrive at an efficient algorithm for determining 2-Castability (suggested by the last section) is to modify the **Characterize Polygon** algorithm to permit the CH_i, CH_j intersection tests. Of course, as i and j advance, we must continually make the hulls CH_i, CH_j available for such tests. This section describes efficient algorithms for maintaining the hulls and performing the intersection tests.

[3] provides us with an amortized $O(n \log n)$ time and $O(n)$ space algorithm for the dynamic maintenance of a convex hull. Its input consists of a sequence of $O(n)$ possibly intermixed insert, delete and test operations. At each stage of the algorithm a structure representing the current convex hull is either updated or queried. The structure supports logarithmic search queries along the current hull. That is, between each add and delete operation we can determine any feature of the convex hull which depends upon a binary search of the points which determine the hull.

[2] provides us with an algorithm which, given two convex hulls, determines in $O(\log n)$ time, whether the hulls intersect. The algorithm accomplishes this through two binary searches, one on each hull.

Notation : An add or delete operation is one of add_p^Q, del_p^Q where add_p^Q, del_p^Q means add point p to the convex hull Q and delete point p from hull Q respectively. $test\cap$ represents testing the intersection of the current hulls.

The above algorithms allow us to compute the sequence $< op_1, \ldots, op_n | op_i \in \{add, del, test\cap\} >$ in $O(n \log n)$ time and space. That is, we can compute any interleaved sequence of adds, deletes and intersection tests of two hulls in time $O(n \log n)$ and space $O(n)$ provided the add-delete-test sequence is determined before execution.

6.5 The Modifications (putting it all together)

When $j = i$, testing whether $B[j] \in \mathbb{Z}_n^{j+1,C[i]}$ determines if there is any $q \in \partial P$ such that v_i and q determine a 2-Mold. If this is the case, then **all** such q will in fact be inside $[v_{B[j]} \ldots v_{C[i]}]$. The kind of intersection test we will carry out depends on the type of chain formed by $[v_{B[j]} \ldots v_{C[i]}]$.

If the chain is a single point then we want to determine if the pair of points $v_i, v_{C[i]}$ determine a 2-Cast. The pair will determine a 2-Cast if and only if CH_i and CH_j intersect only on the common edge $\overline{v_i v_{C[i]}}$. This will be the case if and only if both hulls contain the edge $\overline{v_i v_{C[i]}}$ but with opposite orientations. In this case the interior of the hulls and all other edges are on opposite sides of the line determined by the common edge vertices. The algorithm of [2] admits such a test in O($\log n$) time.

If the chain is not a single point then we want to determine whether there is any $q \in [v_{B[j]} \ldots v_{C[i]}]$ so that the segment $\overline{v_i q}$ remains inside the polygon. Equivalently, we test whether v_i is a vertex of CH_i and CH_j and check the neighboring vertices to see if in fact the hulls intersect precisely at this point. To be exact, we check if the infinite wedges based on v_i determined by v_i and its 4 neighbors, the immediate successors and predecessors of v_i on CH_i and CH_j, intersect. If v_i is not a vertex of either hull or the wedges intersect then the two hulls intersect and there is no 2-Cast determined by v_i and any point in $[v_{B[j]} \ldots v_{C[i]}]$.

The corresponding hull operations (prior to changing j) amount to the single delete $del_{v_j}^{CH_j}$ and the possibly 0 or more adds $add_{v_{B[j]+1}}^{CH_j} \ldots add_{v_{B[j+1]}}^{CH_j}$

As above, we break the case $j = i + 1 \bmod n$ into two subcases. If the back and forward chains overlap at a single point then we determine the 2-Castability status of the polygon in much the same way as the chain overlap subcase in the $j = i$ case above. Finally if the back and forward chains overlap then we determine 2-Castability by determining if CH_i and CH_j intersect. They don't overlap precisely when the polygon is 2-Castable. We advance by incrementing i, executing the $add_{v_i}^{CH_i}$ and the possibly 0 or more deletes $del_{v_{C[i-1]}}^{CH_i} \ldots del_{v_{C[i]-1}}^{CH_i}$. We are now back in the $i = j$ case and we continue.

It should be clear that after advancing n times ($j = i, j = i + 1$) we have returned to our starting position and so have checked the entire polygon. An amortized analysis shows that each vertex appears at most twice in either convex hull so that there are O(n) *add*s and *del*s. There are also only O(n) *test*$\cap$s since we perform some constant number for each i. Consequently the algorithm is in O($n \log n$). The O(n) space bounds apply since the underlying structures require linear space provided we can schedule the test operations before running the algorithm. We can either perform all four intersection queries on the structure at each step of the algorithm or we can 'simulate' the algorithm once to determine the test scheduling and run the full algorithm later. In either case we have the same result.

Theorem 14 : There is a O($n \log n$) time, O(n) space algorithm which determines if an input polygon is 2-Castable.

Without too much more effort, additional O(n) time, we can determine a pair of points which determine a 2-Cast (provided the polygon is 2-Castable).

7 Future Work

Obvious extensions include determination of a polygon's k-Moldability/k-Castability in dimensions 2 and 3. Of particular interest is the determination of 2-Moldability in 3 dimensions. Alternatively, variations on the moldability/castability theme which allow rotation as well as transformation should be investigated.

References

[1] D. Rappaport A. Rosenbloom. Moldable and castable polygons. In *Proceedings of the Fourth Canadian Conference on Computational Geometry*, 1992.

[2] B. Chazelle and D.P. Dobkin. Intersection of convex objects in two and three dimensions. *Journal of the Association for Computing Machinery*, 34:1–27, 1987.

[3] J. Hershberger and S. Suri. Offline maintenance of planar configurations. In *SODA 91*, 1991.

[4] C.C. Lee and D.T. Lee. On a circle cover minimization problem. *Information Processing Letters*, 18:109–115, 1984.

[5] Franco P. Preparata and Michael Ian Shamos. *Computational Geometry an Introduction.* Springer-Verlag, 1985.

[6] Franco P. Preparata and Kenneth J. Supowit. Testing a simple polygon for monotonicity. *Information Processing Letters*, 12(4):161–164, 1981.

[7] Arnold Rosenbloom. Moldable and castable polygons. Master's thesis, Queen's University, 1992.

[8] R. Swaminathan V. Chandru, V.T. Rajan. Monotone pieces of chains. In *Proceedings of The Second Canadian Conference on Computational Geometry*, 1990.

[9] R. Swaminathan V. Chandru, V.T. Rajan. Monotone pieces of chains. *ORSA Journal on Computing*, 1992.