Moldable and Castable Polygons

Arnold Rosenbloom and David Rappaport*

June 18, 1992

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 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 [6]. 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.

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 2 pieces of the mold can be removed by translation then the resulting statue is 2-Moldable. Alternatively, we can form the statue by taking 2 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 an optimal $O(n)$ 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[6] for a polygon. In both cases we exhibit an $O(n)$ optimal algorithm. Finally we use the results of [2] to obtain an $O(n \log n)$ algorithm for determining 2-Castability.

The concept of monotonicity of chains of a polygon will play an important role in the development of algorithms for recognizing moldable sets. Recently, this concept has received increasing attention. In [5] an $\theta(n)$ time and space algorithm is given for determining if a polygon is monotone. In [4] a simple $\theta(n)$ algorithm is given which triangulates a monotone polygon. In [6] an $\theta(n)$ time and space algorithm is given (a consequence of [3]) which determines the minimum number of monotone chains into which a given polygon can be decomposed. We extend this result by supplying a straightforward algorithm which solves the same problem in the same time and space bounds.

2 Notation

By a direction in R^2 we mean a non-origin point which will be used to determine changes in position. By ${}^{d}\overrightarrow{z}$ 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 of the half line based on p and including q. For any $X \subseteq R^2$ $\delta X, X^o, X^c, \overline{X}$ 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. Given natural numbers i, j and n by $[i \ldots j]_n$ we will mean the set of equivalence classes from i to j modulo n. When the n is obvious it will be omitted. 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 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.

*Computing and Information Science, Queens University, Kingston Ontario K7L 3N6 email: rosenblo@qucis.queensu.ca daver@qucis.queensu.ca

Given 2 points $p, q \in \delta P$ 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 $\forall z \in [p \ldots q] \overset{d}{\overrightarrow{z}} \cap P^o = \emptyset$. 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}.

3 Polygons and Monotonicity

A polygonal chain $[p \ldots q]$ is **monotone with respect to** L if there is some orientation of L so that the ordering of the projection of the vertices in $[p \ldots q]$ on L agrees with the ordering of the vertices themselves. We will say that $[p \ldots q]$ is monotone if there is some line L so that $[p \ldots q]$ is monotone with respect to L. A chain $[p \ldots q]$ is **forward maximally monotone** if it is monotone but any extension of the chain (at the q end) produces a non-monotone chain. Similarly, the chain is **backward maximally monotone** if it is monotone but any extension at the p end produces a non-monotone chain. 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. A monotone decomposition is **minimal** if the number of chains in the decomposition is minimal.

4 Towards a Computable Characterization of 2-Moldability

We will demonstrate that determining 2-Moldability is equivalent to determining 2-Monotonicity.

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

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

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 $[v_i \ldots v_j]$ and $[v_j \ldots v_i]$ are monotone. Unfortunately, this naive algorithm has complexity at least $O(n^2)$.

5 Computing Forward Maximal Monotone Chains

This section details an optimal $O(n)$ time and space algorithm for determining all forward maximal monotone chains for a given polygon. This algorithm will be used to develop linear time algorithms for determining 2-Monotonicity, 2-Moldability and a minimum monotone decomposition of a simple polygon.

5.1 Computing $C[]$

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 $[v_i \ldots v_j]$ is a forward maximal monotone chain.

Lemma: A chain $[p \ldots q]$ of P is 2-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 $\{(\cos(\theta), \sin(\theta)) | \theta \in R\}$. For a, b in the unit circle, by the **angle from** a **to** b; written $\angle ab$, we will mean the set of points 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 on the unit circle and solving the equivalent problem.

Problem: Given a sequence 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

For each $i \in 1, \ldots, n$ 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.

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

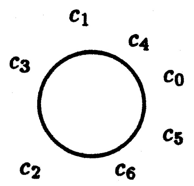

Figure 1: Here we have P[0] = 1, P[1] = 3, P[2] = 3, P[3] = 4, P[4] = 1, P[5] = 1, P[6] = 1.

For distinct natural numbers $a, b \in 0, \ldots, n-1$ let U_a^b denote the set $\{c_a, a_{a+1}, \ldots, c_b\}$. Let M_a^b denote the set of points of U_a^b which are in some angle formed by a consecutive pair of elements of U_a^b which appear farther on in the sequence. Define $L_a^b = U_a^b - M_a^b$

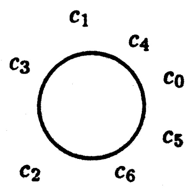

Figure 2: Here we have $U_0^5 = \{c_0, c_1, c_2, c_3, c_4, c_5\}$,$M_0^5 = \{c_0, c_1\}$ and $L_0^5 = \{c_2, c_3, c_4, c_5\}$

Claim: Assume $|\cup_{k=a}^{b-1} \angle\angle c_k c_{k+1}| \leq \pi$ then there are c, c' in L_a^b such that $\angle cc' = \cup_{k=a}^{b-1} \angle\angle c_k c_{k+1}$

This last claim leads to an O(n) time and space algorithm which computes $P[]$. The algorithm maintains a linked list $\mathcal{L}$, three pointers to $\mathcal{L}$, *first*, *last*, *current* and two indexes i, j. i represents the next element of $P[]$ to be computed and j represents the index of the next element of $< c_0, \ldots, c_{n-1} >$ to be added to $\mathcal{L}$. $\mathcal{L}$ will represent L_i^{j-1} in as much as $z \in \mathcal{L} \Leftrightarrow z \in L_i^{j-1}$. Elements of $\mathcal{L}$ have four attributes, so if $z \in \mathcal{L}$ represents c_k then z contains the integer k (referred to as index(z) the real number c_k (referred to as point(z)) and the two linking pointers f(z) and b(z) connecting $\mathcal{L}$.

Notation: $\mathcal{L}_{c[i]}$ is the unique member of $\mathcal{L}$ representing $c[i]$.

Using the ideas presented so far, we arrive at the following algorithm.

Algorithm: Compute $P[]$

Input: An integer n, an array of angles $c[]$ of size n. The array $c[]$ is assumed to satisfy all of the requirements for $< c_0, \ldots, c_{n-1} >$ outlined above.

Output: The array $P[]$.

```
Algorithm:
{Initialize $\mathcal{L}$}
If $\angle\angle c[0]c[1] = \angle c[0]c[1]$ then
    $f(\mathcal{L}_{c[0]}) \leftarrow \mathcal{L}_{c[1]}$
    $b(\mathcal{L}_{c[1]}) \leftarrow \mathcal{L}_{c[0]}$
    $f(\mathcal{L}_{c[1]}) \leftarrow NULL$
    $b(\mathcal{L}_{c[0]}) \leftarrow NULL$
    $first \leftarrow \mathcal{L}_{c[0]}$
    $last \leftarrow \mathcal{L}_{c[1]}$
else
    $b(\mathcal{L}_{c[0]}) \leftarrow \mathcal{L}_{c[1]}$
    $f(\mathcal{L}_{c[1]}) \leftarrow \mathcal{L}_{c[0]}$
    $b(\mathcal{L}_{c[1]}) \leftarrow NULL$
    $f(\mathcal{L}_{c[0]}) \leftarrow NULL$
    $last \leftarrow \mathcal{L}_{c[0]}$
    $first \leftarrow \mathcal{L}_{c[1]}$
end if
$current \leftarrow \mathcal{L}_{c[1]}$
$i \leftarrow 0$
$j \leftarrow 2$
while($i < n$)
  {A}
  while($|\angle$point($first$) point($last$)$| \leq \pi$)
    { Add $c[j]$ into $\mathcal{L}$ }
    if $\angle\angle c[j-1]c[j] = \angle c[j-1]c[j]$ then
      {B}
      remove all points which are forward
        from $current$ but contained in
        $\angle c[j-1]c[j]$ maintaining $last$
      and $f(\mathcal{L}_{c[j-1]})$
    $f(\mathcal{L}_{c[j]}) \leftarrow f(\mathcal{L}_{c[j-1]})$
    $b(\mathcal{L}_{c[j]}) \leftarrow \mathcal{L}_{c[j-1]}$
    $f(\mathcal{L}_{c[j-1]}) \leftarrow \mathcal{L}_{c[j]}$
    if $current = last$ then $last \leftarrow \mathcal{L}_{c[j]}$
  else
    {C}
    $\{\angle\angle c[j-1]c[j] = \angle c[j]c[j-1]\}$
    remove all points which are back
      from $current$ but contained in
      $\angle c[j]c[j-1]$ maintaining $first$
      and $b(\mathcal{L}_{c[j-1]})$
    $b(\mathcal{L}_{c[j]}) \leftarrow b(\mathcal{L}_{c[j-1]})$
    $f(\mathcal{L}_{c[j]}) \leftarrow \mathcal{L}_{c[j-1]}$
    $b(\mathcal{L}_{c[j-1]}) \leftarrow \mathcal{L}_{c[j]}$
    if $current = first$ then $first \leftarrow \mathcal{L}_{c[j]}$
  end if
  $current \leftarrow \mathcal{L}_{c[j]}$
  $j \leftarrow j + 1 \bmod n$.
end while
{Remove an element from $\mathcal{L}$}.
{D}
If $current = first$ then
  {E}
  while($|\angle$ point($first$) point($last$)$| > \pi$)
    $k \leftarrow$index($last$)
    {F}
    while($i \neq k + 1 \bmod n$ and $i < n$)
      $P[i] \leftarrow j - 2$
      $i \leftarrow i + 1$
    end while
    Remove the last element from $\mathcal{L}$(update last)
  end while
else
  {$current = last$}
  {G}
  while($|\angle$point($first$) point($last$)$| > \pi$)
    $k \leftarrow$index($first$)
    {H}
    while($i \neq k + 1 \bmod n$ and $i < n$)
      $P[i] \leftarrow j - 2$
      $i \leftarrow i + 1$
    end while
    Remove the first element from $\mathcal{L}$(update $first$)
  end while
end if
end while
```

5.1.1 Analysis and Complexity

Correctness of the algorithm follows from the discussion preceding the algorithm and the following observations...

- $\mathcal{L}$ represents L_i^{j-1}
- $\angle$ point($first$) point($last$)$= \cup_{k=i}^{j-2} \angle\angle c_k c_{k+1}$
- if z_1 and z_2 appear in this order in L then point(z_2) $\in \angle$ point(z_1) point($last$).

O(n) time complexity follows from the following amortized analysis. The amount of time spent initializing $\mathcal{L}$ is constant. During the addition of $c[j]$ into $\mathcal{L}$ (step B,C) the algorithm may delete neighbors of $current$ with each deletion taking constant time. We obtain an O(n) amortized cost for these deletions (step A) by noting that each element may appear in $\mathcal{L}$ at most two times throughout the execution of the algorithm. The sequence of values j takes on is some prefix of the sequence $< 2, 3, \ldots, n-1, 0, 1, 2, \ldots, n-1 >$ in which each $m \in \{0, \ldots, n-1\}$ appears twice. Now $c[m]$ is inserted into $\mathcal{L}$ exactly when $j = m$, consequently it must be the case that $c[m]$ is inserted to and deleted from $\mathcal{L}$ at most twice. The code inside while loops F and H is executed at most n times throughout the lifetime of the algorithm. The execution cost of the code inside while loops E and G sums up to the number of deletions which take place and so together they cost O(n). It should be noted that every time A is executed at least one insertion takes place. Similarly, every time D is executed, at least one deletion takes place and i is incremented by at least one. Consequently the main while loop costs the algorithm linear time. The space bound comes from the fact that each $c[m]$ appears at most once in $\mathcal{L}$ so the size of $\mathcal{L}$ is O(n).

Noting that $C[i] = P[i] + 1$ we obtain the following theorem.

Theorem: There is a linear time algorithm to compute all forward maximal monotone chains of a given polygon.

5.2 Application 1: Determining 2-Moldability of a Polygon

We can now use $C[]$ 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: 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 [6] and solve the minimum monotone decomposition problem. That is, given a polygon as a sequence of vertices, determine its minimum monotone decomposition. 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 [3] 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 [6]. It should be noted that some variant of the algorithm of [6] might be used to decide 2-Moldability. However, our algorithm is more straightforward.

6 2-Castability

To relate 2-Castability to 2-Moldability we note that a polygon is 2-Castable when it is 2-Moldable with mold pieces $[p \ldots q]$ and $[q \ldots p]$ and the segment $\overline{pq}$ stays strictly inside the polygon.

6.1 Deciding 2-Castability

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. We do this by checking all pairs of points $p, q \in \delta P$ which determine a 2-Mold for P and then checking if $\overline{pq} \subset P$. 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.

6.1.1 Background Algorithms

[2] 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.

[1] 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.

6.1.2 Towards an Algorithm

To decide 2-Castability, we will walk two chains around the polygon, all the while determining if the chains determine a 2-Cast.

For a given polygon P, the array $B[]$ satisfies $B[i] = j$ if and only if $[v_{B[i]} \ldots v_i]$ is a backwards maximal monotone chain. We will refer to $[B[j] \ldots j]$ and $[i \ldots C[i]]$ as **the back chain at** j and **the forward chain at** 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.

The characterization of 2-Castability leading to our algorithm comes from the following claims.

Claim: P is 2-Moldable if and only if there is some $i \in 0, \ldots n-1$ and $j \in [i \ldots C[i])$ such that the forward chain at i and the back chain at j completely cover the boundary of P.

Claim: Given i and j as in the last claim, if $p \in [v_i \ldots v_j]$ and $q \in [v_{B[j]} \ldots v_{C[i]}]$ then p and q determine a 2-Mold for P.

This claim is the key to the correct scheduling of update and test operations generated by our algorithm. We will maintain two indexes (i, j) into the sequence of vertices such that $j \in [i \dots C[i]]$. We also maintain two dynamic convex hull structures, one corresponding to the convex hull of $[v_{C[i]} \dots v_i]$ and another corresponding to the convex hull of $[v_j \dots v_{B[j]}]$ (referred to as CH_i and CH_j respectively). At each main step of the algorithm we will determine if the pair of chains $[v_i \dots v_{C[i]}]$ and $[v_{B[j]} \dots v_j]$ admit a 2-Mold. That is, we will test if $B[j] \in (j \dots C[i]]$. If this is the case then we will determine the kind of intersection (if any) of CH_i and CH_j. If the chains do not admit a 2-Mold then we will advance (increment modulo n) either i or j or both. With each advance, we do some convex hull maintenance.

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

If the chain is a single point then we want to determine if the pair of points $v_i, v_{C[i]}$ determines 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 and linear space.

If the chain is not a single point then we want to determine whether there is any $q \in [v_{B[j]} \dots v_{C[i]}]$ 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 $[v_{B[j]} \dots v_{C[i]}]$.

In all cases outlined above we advance by incrementing j. 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 $add_{v_{B[j]+1}}^{CH_j} \dots add_{v_{B[j+1]}}^{CH_j}$

Throughout the execution of our algorithm, if $i \neq j$ then $j = i + 1$ (taking indices modulo n) so the only case left to consider is when $j = i + 1$. Again we break this 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 way as the chain overlap subcase in the $i = j$ 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 $del_{v_{C[i-1]}}^{CH_i} \dots del_{v_{C[i]-1}}^{CH_i}$. We are now back in the $i = j$ case and we continue.

It should be clear that after advancing $2n$ times (each at logarithmic cost) we have returned to our starting position and so have checked the entire polygon. 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.

7 Future Work

Obvious extensions include determination of a polygons 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] 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.

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

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

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

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

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