Reachable grasps on a polygon: the convex rope algorithm
M. A. Peshkin and A. C. Sanderson
IEEE Transactions on Robotics and Automation 2:1 (1986)

Reachable Grasps on a Polygon: The Convex Rope Algorithm

MICHAEL A. PESHKIN AND ARTHUR C. SANDERSON, MEMBER, IEEE

Abstract—**An algorithm that finds the externally visible vertices of a polygon is described. This algorithm generates a new geometric construction, termed the *convex ropes* of each visible vertex. The convex ropes give the range of angles from which each vertex is visible, and they give all the pairs of vertices which are reachable by a straight robot finger. All of the convex ropes can be found in expected time order n, where n is the number of vertices of the polygon. We discuss the application of this geometric construction to automated grasp planning. The algorithm may also be useful in image interpretation and graphics where efficient computation of visible points is important. The direct application of the algorithm is restricted to two dimensions since sequential ordering of vertices is required. Extention to three dimensions would rely on well chosen intersecting or projective planes.**

Manuscript received August 6, 1985; revised December 30, 1985.

The authors are with the Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA 15213, USA.

IEEE Log Number 8607695.

I. Background and Motivation

An algorithm that efficiently finds the externally visible vertices of a polygon and the range of angles from which each is visible is described in this paper. The associated geometric constructions, called *convex ropes,* provide much useful information for robot grasp selection and planning involving polyhedral objects. They may also be useful for image interpretation and graphics, where efficient computation of visible points is essential for analysis in terms of image projections.

Currently, when a robot grasps an object, the details of the motion have generally been specified in advance by a human programmer with detailed knowledge of the object geometry. If robots are to become less dependent on task-specific programming, grasp selection based on stored geometric models is one of the functions that must be automated.

The grasping problem is to find surfaces on an object, and a configuration of a robot's fingers, satisfying the following three conditions [5].

1) The fingers must be in contact with the object.
2) The configuration must be reachable; i.e., there must be a collision-free path for the robot to get to the grasping configuration.
3) The object must be stable once grasped; i.e., it must not slip during subsequent motion.

Evidently the grasping problem is intimately related to the collision-avoidance problem. However, in collision-avoidance, one effective strategy is to find "safe" paths. Conservative approximations may be made to guarantee that collisions will not occur. For instance, a stationary obstacle may be represented by its convex hull. If a moving part does not intersect the obstacle's convex hull, surely it will not intersect the obstacle. But there may be motions that are deemed unavailable by this method, even though in fact they are collision-free.

In grasping, such conservative approximations may be useful in planning approach, but the contact requirement makes them inappropriate for generating the grasps. The grasping configuration itself is a marginal collision: the robot fingers touch the object.

There is also the requirement of stability of the object within the grasp. What may first come to mind here are quantitative physical characteristics of the grasp such as is resistance to slippage when a certain force is applied. However, even more basic stability requirements must be met first, many of which are qualitative geometric characteristics.

In this paper we will restrict ourselves to two-fingered robots, where the fingers are straight and parallel: the classic two-jaw gripper. In this context three of the geometric requirements on a grasp are as follows.

1) The object must touch the fingers either on the inner surface of both fingers, or on the outer surface of both fingers, but not one of each (the *matter distribution requirement* in [3]).
2) Surfaces that the fingers are expected to touch must be parallel.
3) Such pairs of parallel surfaces must be reachable from the same side.

Others may be found in [3] and [9].

Previous approaches to automated grasp generation have followed this basic outline [6].

1) Generate a set of candidate grasp configurations, typically by forming all pairs of edges or surfaces.

2) Reject those configurations which do not satisfy the geometric requirements.
3) Reject those configurations which are not reachable by the robot fingers.
4) Choose among those that remain on the basis of stability or other goals.

For an object with n edges or surfaces, the number of candidate grasps generated in step 1 is of order n^2. In addition, there is a class of grasps that is not found at all by this strategy. Often a single finger may touch an object at two vertices that are not connected by an object surface. Fig. 1 shows a grasp made up of two such *virtual surface* contacts (neither of which is a segment of the convex hull of the object.) The existence of this type of grasp was noted (though not treated) in [4]. To the best of our knowledge, all previous grasp generation procedures would fail to produce the grasp shown in Fig. 1. Consideration of such grasps further increases the number of candidate grasps.

Steps 2 and 3 require geometric computations comparing each grasping surface with many other surfaces of the object. For intricate objects, these calculations can become unwieldy. Our approach is to integrate some of the grasp selection criteria into the grasp generation procedure, eliminating subsequent testing and rejection. Our approach is to

1) generate only the set of edges, pairs of edges, or surfaces which are reachable by a single infinitely thin robot finger;
2) form pairs which satisfy the geometric requirements; and
3) choose among those on the basis of stability or other goals.

In the following sections we introduce a new geometric construction, called the *convex rope,* and an algorithm that computes it. The algorithm efficiently performs step 1. It also provides most of the information needed for step 2. Step 3 will not be addressed in this paper.

The algorithm, and the rope construction as well, apply directly only to two-dimensional polygons. Some three-dimensional objects, such as extrusions and punched parts, have one uninteresting axis. Their cross-section contains all the information about the object. In these cases the rope construction can be applied to the cross section.

More complicated three-dimensional objects may be treated by consideration of the intersection of the object with well-chosen planes, yielding two-dimensional figures. This projective approach has been applied successfully in other collision-avoidance and grasp selection problems [7], [1], [9], [5], [2].

II. Introduction

The rope algorithm is based on an order n convex hull algorithm [8]. It capitalizes on the orderly traversal made by the convex hull algorithm of all the vertices of a polygon. A similar traversal of the vertices allows the rope algorithm to extract geometric information about the interior pockets and folds of a complicated figure.

A vertex of a polygon is externally visible if a line can be drawn from it to infinity without intersecting the polygon. Such a vertex can be seen from some angle, with the observer in the plane of the polygon. Conversely, if a vertex is not externally visible, it can not be seen from any angle, and, for instance, no attention need be given to it in a graphics display. An entire polygon may be externally visible, but we will be concerned here with finding the externally visible vertices of a general polygon.

For each externally visible vertex x there is a continuous range of angles from which it is visible (as in Fig. 2). At either end of the range the line-of-sight to the vertex x is obstructed by some other vertex of the polygon. In Fig. 2 these two vertices are labeled a and b. One of the functions of the rope algorithm is to identify the two obstructing vertices.

At either end of the range described above, the line-of-sight grazes the obstructing vertex. As in the example shown (Fig. 2), in many cases the line-of-sight intersects some other segment of the polygon, called the *shadowed segment*, after grazing the target vertex x. The algorithm also identifies the shadowed segment.

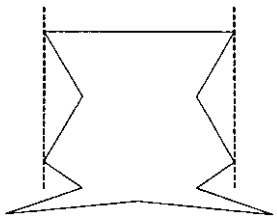

Fig. 1. Parallel virtual surfaces (dashed) which form a grasp of the polygon (solid).

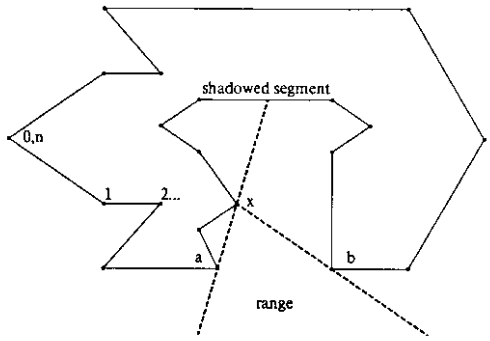

Fig. 2. Range of angles from which point x is externally visible, and the construction defining the shadowed segment.

Only points of the polygon that are externally visible are reachable by straight fingers in the plane of the polygon. In addition to individual points, some entire edges of the polygon are reachable. If the two vertices that delimit an edge of the polygon are both visible from the angle of that edge, then both can be touched simultaneously by a straight finger at that angle. In this case the finger makes contact with the entire edge.

It is also possible for a pair of vertices that do not delimit an edge to be simultaneously reachable from the angle of the line formed by the two vertices. In this case a straight finger may make contact with both vertices simultaneously, forming a two-point contact with the finger, or virtual surface contact. In Fig. 2 vertices a and x form a virtual surface. The algorithm provides the information needed to determine simultaneous reachability of two vertices. It also identifies, for each vertex, the other vertices which are candidates for the formation of a virtual surface contact.

Finally, if a finger is in contact with an edge or a virtual surface, its end can extend beyond the end of the edge or the more distant vertex of the virtual surface. It is obstructed only when it extends far enough to intersect the shadowed segment described above, which is also identified by the rope algorithm.

III. Rope Construction

Consider a simple polygon P described by a sequence of points in the plane (x_i, y_i) which are consecutive vertices of the polygon. We will refer to the vertices by their sequence number i, i.e., vertex 3 means the point (x_3, y_3). The only constraints on the points are that the polygon described by them must be closed, and that one edge of the polygon must not intersect another. These requirements give the polygon a well-defined interior.

The sequence of vertices may be a clockwise (CW) or a counterclockwise (CCW) description of the polygon. In a CW description, the interior of the polygon is to the right of the vector from one vertex to the next; in a CCW description, the interior of the polygon is to the left of the vector.

If P has n vertices, we define vertex $\mathbf{0}$ to be equivalent to vertex n. Also, we choose our starting vertex $\mathbf{0}$ so that it is an extremal point in the negative x direction. This insures that $\mathbf{0}$ is on the convex hull. The convex hull of P is a simple polygon whose vertices are some of the vertices of P. A description of the convex hull is the sequence of

vertices $\{C_k, k = 0, 1, 2 \cdots m\}$ which is a subsequence of the integers. Let vertices $\{0, 1 \cdots n\}$ be a CCW description of P. Sufficient conditions for $\{C\}$ to be a description of the convex hull of P are as follows.

1) $C_0 = 0$, $C_m = n$.
2) $\{C\}$ is convex: every vertex C_i is left of **line**(C_{i-2}, C_{i-1}), i.e., makes a left turn.
3) $\{C\}$ is external, meaning that if a vertex j satisfies $C_{i-1} < j < C_i$ then j is left of **line**(C_{i-1}, C_i).

The CCW convex rope to an externally visible vertex j is, like the convex hull, described by a subsequence $\{R_k, k = 0, 1, 2 \cdots p\}$ of the integers. We require $R_0 = 0$, and $R_p = j$. $\{R\}$ must be convex and external. Unlike the convex hull, the convex rope is not a closed polygon. An example of a CCW convex rope is shown in Fig. 3, where R, the convex rope to $j = 9$, is $\{0, 3, 4, 6, 9\}$.

The CCW convex rope is the shortest path from vertex **0** to vertex j, external to the polygon, subject to the constraint that it be a CCW path; that is, it must be described by an increasing subsequence of the integers.

The CW convex rope to a vertex j is the obvious analogous construction described by a decreasing subsequence of the integers starting with $R_0 = n$ (which is identical to vertex **0**), and ending at $R_p = j$.

The CW and CCW convex ropes to vertex j together constitute a *cardioid hull* of the polygon (Fig. 4). If j happens to be on the convex hull, then the cardioid hull and the convex hull are the same. If j is not on the convex hull, the cardioid hull has a single point (vertex j) at which it is not convex.

The cardioid hull may be thought of as the shortest path around the polygon, external to it, that passes through vertex j. (Removing the requirement that the path pass through vertex j leaves a definition of the convex hull.) There is a different cardioid hull for each vertex j, so long as j is externally visible and is not on the convex hull.

The convex ropes have the important property that if $\{R_k, k = 0, 1, 2 \cdots p\}$ is the CCW convex rope to R_p, then the truncated sequence of vertices $\{R_k, k = 0, 1, 2 \cdots q\}$ (where $q < p$) is the CCW convex rope to R_q. It is therefore possible to arrange the externally visible vertices of the polygon in a tree structure (Fig. 5). Each vertex j has a pointer called **source**$_j$ to its parent vertex in the tree. **Source**$_j$ is the vertex which immediately precedes j on the CCW convex rope to j. That is, if $j = R_p$, **source**$_j = R_{p-1}$.

The entire CCW convex rope to any vertex j can be read off (backwards) as $\{j$, **source**(j), **source(source**(j)**)**, **source(source-(source**(j)**))**, $\cdots$ **0**$\}$. Information sufficient to generate the CCW convex rope to all of the externally visible vertices of the polygon is stored by having a single pointer from each vertex to its source. Obviously the same applies to the CW convex ropes. In general we need two pointers from each vertex: a CW source and a CCW source.

Recall that the last vertex (n) of the polygon is equivalent to the first (**0**), and that it was chosen to be on the convex hull. The CCW convex rope to n is the convex hull of the polygon, and **source**$_n$ is a vertex on the convex hull. Since the CCW convex rope to vertex **0** is a sequence containing only one element, namely **0**, **source**$_0$ is undefined. It is convenient to define **source**$_0$ = **source**$_n$. This removes the uniqueness of vertex **0** as the starting point of all of the ropes.

With this definition, we have introduced a loop (Fig. 6) into the tree described earlier (Fig. 5). The root node of the tree is identified with one of the terminal nodes. In reading off a convex rope (backwards) as $\{j$, **source**(j), **source(source**(j)**)**, **source(source-(source**(j)**))**, $\cdots\}$ there is no longer a final term. When the term 0 is encountered, the sequence goes into a repeating cycle consisting of those vertices on the convex hull. Note that there is nothing unique about vertex **0** once **source**$_0$ has been defined. There is no way to distinguish vertex **0** from any other vertex on the convex hull, by looking only at the looped tree.

As can be immediately seen in Fig. 4, the rays from vertex j to its CCW source and CW source define a range of angles. Any ray from j intermediate between these two rays will not intersect the polygon; any ray outside of their range will intersect the polygon. If j is lacking a CW source or a CCW source (or both), as for example vertex $j + 2$ in Fig. 4, then j is not externally visible: all rays from j intersect the polygon.

Each pair (j, **source**$_j$) is an reachable grasping surface. Some of these are also real surfaces of the polygon, while others, such as the dotted line in Fig. 7, are virtual surfaces defined by two points. A single, straight, and infinitely thin finger in the plane of the polygon can make contact with the polygon on just these reachable grasping surfaces. One of the geometric requirements imposed on the selection of a pair of grasping surfaces is that the object must touch the fingers

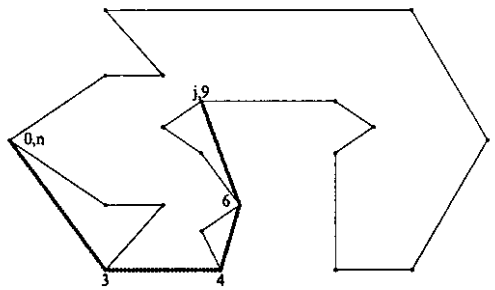

Fig. 3. Counter-clockwise convex rope to j.

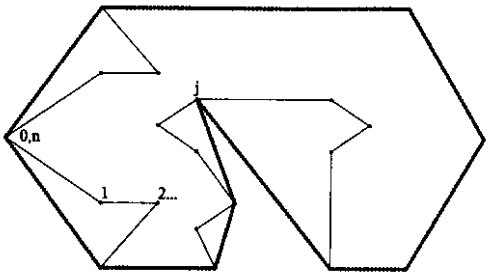

Fig. 4. Cardioid hull of the polygon, constructed for vertex j.

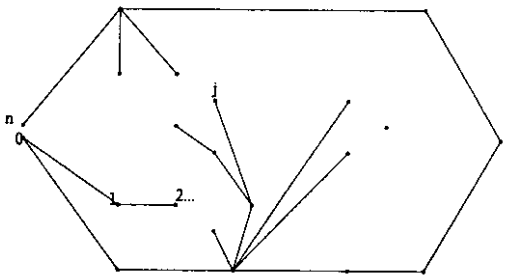

Fig. 5. Tree structure of the CCW convex ropes to all the externally visible vertices.

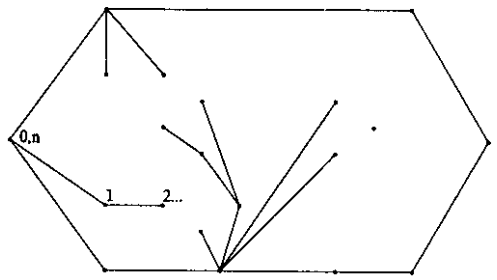

Fig. 6. Looped tree structure of the CCW convex ropes.

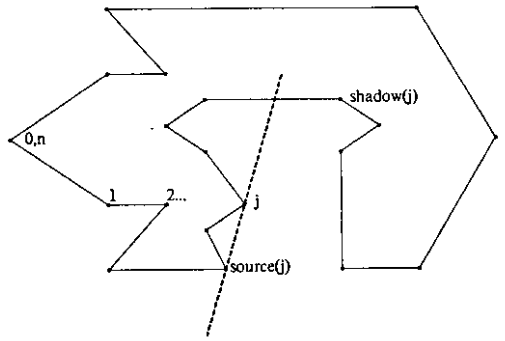

Fig. 7. Construction showing the relationship of j, **source**(j), and **shadow**(j), as used in the convex rope algorithm.

either on the inner surface of both fingers or on the outer surface of both fingers, but not one of each. This "matter distribution" requirement can be fulfilled by choosing the surface one finger is to contact from among the CW class of ropes, and the other surface from among the CCW class.

A surface (or virtual surface) (j, **source**$_j$) is matched with a parallel surface from the opposite class. To avoid choosing pairs that are reachable only from opposing sides, one need only note the distinction between parallel and anti-parallel surfaces. Surfaces are parallel if their rays (j, **source**$_j$) are parallel.

An additional useful piece of geometric information is available. For each externally visible vertex j, the ray originating at j and passing through **source**$_j$ clearly does not intersect the polygon. The ray originating at **source**$_j$ and passing through j does intersect the polygon, unless j happens to be on the convex hull (Fig. 7). The segment of the polygon that the ray first intersects is designated (**shadow**$_j$ − 1, **shadow**$_j$), and is found by the algorithm.

By finding the intersection of the **ray(source**$_j$, j) with **segment-(shadow**$_j$ − 1, **shadow**$_j$), we can also find the amount of clearance behind the interior point of contact (j) with the finger. This is just the distance from j to the point of intersection. This information is needed when attempting to pair the grasping surface with another one.

Convex Hull Algorithm

Input

1) (x_i, y_i), for $0 \le i \le n$, are the Cartesian coordinates of the vertices of the polygon.
2) Vertex **0** is extremal in the negative x direction.
3) Vertices **0** and n are equivalent.
4) The polygon is traversed in a CCW sense with increasing subscript.

Output

stack$_i$ are the vertex numbers of the vertices on the convex hull.

Algorithm

1) Definition of function CCW(k, j, i), ranging from $-\pi$ to π:

   if ($k = -1$) CCW(k, j, i) = the CCW angle from the positive x-axis to **ray**$((x_j, y_j)(x_i, y_i))$.
   if ($k \ge 0$) CCW(k, j, i) = the CCW angle from **ray**$((x_k, y_k)(x_j, y_j))$ to **ray**$((x_j, y_j)(x_i, y_i))$.

2) Initialize

   $\text{stack}_0 \leftarrow -1$
   $\text{stack}_1 \leftarrow 0$ $\text{stack}_2 \leftarrow 1$
   $\text{pathangl}_1 \leftarrow 0$ *pathangl*$_2 \leftarrow$ CCW(−1, 0, 1)
   $\text{cumangl} \leftarrow \text{pathangl}_2$
   $\text{sp} \leftarrow 2$

3) Compute

   for $i = 2$ to n

   Update cumangl and pathangl:
   $\text{cumangl} \leftarrow \text{cumangl} + \text{CCW}(i-2, i-1, i)$
   $\text{pathangl}_{sp+1} \leftarrow \text{pathangl}_{sp} + \text{CCW}(\text{stack}_{sp-1}, \text{stack}_{sp}, i)$
   Test for disagreement:
   if $\text{pathangl}_{sp+1} - \text{cumangl} > 0.1$
   Reject $i - 1$:
   $\text{sp} \leftarrow \text{sp} - 1$
   $\text{pathangl}_{sp+1} \leftarrow \text{pathangl}_{sp} + \text{CCW}(\text{stack}_{sp-1}, \text{stack}_{sp}, i)$
   Test for right turn:
   while $\text{sp} > 1$ and $\text{pathangl}_{sp+1} \le \text{pathangl}_{sp}$
   Reject stack:
   $\text{sp} \leftarrow \text{sp} - 1$
   $\text{pathangl}_{sp+1} \leftarrow \text{pathangl}_{sp} + \text{CCW}(\text{stack}_{sp-1}, \text{stack}_{sp}, i)$
   $sp \leftarrow \text{sp} + 1$
   $\text{stack}_{sp} \leftarrow i$

   end

V. CCW Convex Rope Algorithm

Input

1) (x_i, y_i), for $0 \le i \le n$ are the Cartesian coordinates of the vertices of the polygon.
2) Vertex **0** is extremal in the negative x direction.
3) Vertices **0** and n are equivalent.
4) The polygon is traversed in a CCW sense with increasing subscript.

Output

1) **stack**$_i$ are the vertex numbers of the vertices on the convex hull.
2) **source**$_i$ is the vertex preceding i on the CCW convex rope to i.
3) (**shadow**$_i - 1$, **shadow**$_i$) is the segment that first intersects **ray(source**$_i$, i)

Algorithm

1) Definition of function CCW(k, j, i), ranging from $-\pi$ to π:

   if ($k = -1$) CCW(k, j, i) = the CCW angle from the positive x-axis to **ray**$((x_j, y_j)(x_i, y_i))$
   if ($k \ge 0$) CCW(k, j, i) = the CCW angle from **ray**$((x_k, y_k)(x_j, y_j))$ to **ray**$((x_j, y_j)(x_i, y_i))$

2) Initialize

   $\text{stack}_0 \leftarrow -1$
   $\text{stack}_1 \leftarrow 0$
   $\text{stack}_2 \leftarrow 1$
   $\text{pathangl}_1 \leftarrow 0$
   $\text{pathangl}_2 \leftarrow \text{CCW}(-1, 0, 1)$
   $\text{cumangl}_1 \leftarrow \text{pathangl}_2$
   $\text{sp} \leftarrow 2$
   $\text{source}_1 \leftarrow 0$
   $\text{mostcw}(0) \leftarrow \{1\}$

3) Compute
   for i = 2 to n
   Update cumangl and pathangl:
   $\text{cumangl}_i \leftarrow \text{cumangl}_{i-1} + \text{CCW}(i-2, i-1, i)$
   $\text{pathangl}_{sp+1} \leftarrow \text{pathangl}_{sp} + \text{CCW}(\text{stack}_{sp-1}, \text{stack}_{sp}, i)$
   Test for disagreement:
   if $\text{pathangl}_{sp+1} - \text{cumangl}_i > 0.1$
   Reject $i - 1$:
   $\text{sp} \leftarrow \text{sp} - 1$
   $\text{pathangl}_{sp+1} \leftarrow \text{pathangl}_{sp} + \text{CCW}(\text{stack}_{sp-1}, \text{stack}_{sp}, i)$
   Test for right turn:
   while $\text{sp} > 1$ and $\text{pathangl}_{sp+1} \le \text{pathangl}_{sp}$

```
        Reject stack_sp:
          shadow(stack_sp) ← i
          sp ← sp − 1
          pathangl_{sp+1} ← pathangl_sp + CCW(stack_{sp−1}, stack_sp, i)
      let v at all times represent the last element of mostcw(stack_sp)
      while CCW(stack_sp, v, i) > 0
            if segment(stack_sp, v) intersects segment(i − 1, i)
            or if pathangl_sp + CCW(stack_{sp−1}, stack_sp, v)
                              + CCW(stack_sp, v, v + 1)
                                − cumangl_{v+1} > 0.1
          Old most-CW is invalid:
          source_x ← −1, v ≤ x < i
          unlink last element of mostcw(stack_sp)
      else
          Stack_sp is wrong—Extend the path:
          pathangl_{sp+1} ← pathangl_sp + CCW(stack_{sp−1}, stack_sp, i)
          sp ← sp + 1
          stack_sp ← v
    source_i ← stack_sp
    link i to mostcw(stack_sp)
    sp ← sp + 1
    stack_sp ← i
  source_0 ← source_n
end
```

VI. Description of CCW Convex Rope Algorithm

The CCW convex rope algorithm is built upon an order n algorithm for finding the convex hull, described fully in [8].

Both algorithms make use of a cumulative angle calculated for each side of the polygon. A vector from vertex $i-1$ to vertex i will be called **vector**$(i-1, i)$. We associate the cumulative angle of this vector with vertex i, calling it **cumangl**(i).

Cumangl(1) is defined to be the angle of **vector(0, 1)** with respect to the positive x-axis. Each subsequent cumulative angle, **cumangl**(i), is obtained from the previous cumulative angle **cumangl**$(i-1)$, by adding to it the CCW angle from **vector**$(i-2, i-1)$ to **vector**$(i-1, i)$. If **vector**$(i-1, i)$ is CCW (i.e., to the left) of **vector**$(i-2, i-1)$, we assign it a higher cumulative angle. **Cumangl**(i) is just the angle of **vector**$(i-1, i)$ with respect to the x-axis, but with an integer multiple of 2π added on. The cumulative angle is not treated modulo 2π.

Pathangl(j) is the angle of **vector(stack**$(j-1)$, **stack**(j)**)** with **cumangl**, which is computed by following the edges of the polygon, **pathangl** is computed by following segments of the convex path defined by the vertices on the stack.

The algorithm as presented here finds the CCW ropes for a CCW description of a polygon. By reversing all the comparisons, the CW ropes for a CW description can be found. Of course a CCW description can be transformed into a CW description by reversing the sequence of the vertices.

In addition to the simple stack used in the convex hull algorithm, the convex rope algorithm uses several other data structures.

The array **source**$_i$ contains, at the completion of the algorithm, the source vertex j for each externally visible vertex i. For most of the vertices which are not externally visible, **source**$_i$ will be undefined (as indicated by a negative value). Under certain conditions a CCW source will be assigned to a vertex i, but no CW source will be assigned when the algorithm is applied to the reversed object, (or vice versa). Only externally visible vertices have both a CCW and a CW source.

The array **shadow**$_j$ contains the value of i at which vertex j was removed from the stack. Since i was the first vertex which could remove j, $(k, j, i-1)$ must be convex (Fig. 8), while (k, j, i) is concave. Therefore, **line**(k, j) must intersect **segment**$(i-1, i)$. Since k is the source vertex for j, vertex i is **shadow**$_j$.

Finally, **mostcw**(j) contains data used internally by the algorithm. It can be thought of as a set of stacks, one for each vertex j, although for economy of space it is best implemented as linked lists. The most recent element linked to list **mostcw**(j) is the most CW vertex h (measured from j), for which **source**$_h = j$. When a new vertex i is encountered for which **source**$_i = j$, vertices j and h are compared to see which is most CW from j.

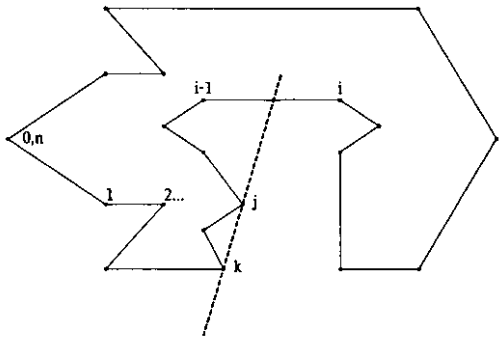

Fig. 8. Construction showing the relationship of j, k, and i, as used in the convex rope algorithm.

If i is more CW than h, i is linked to list mostcw(j). If, however, i is not more than h, this indicates that j is not truly the source for both i and h. There are three distinct cases where i is not more than CW than h, as illustrated in Fig. 9. Vertex h could be A, B, or C, and in each case the new vertex i is CCW of h.

If $h = A$, the path to i intersects the polygon. This results from the removal of vertices from the convex path, which must now be reattached. The proper response is to reattach $h = A$ to the path between j and i.

If $h = B$ or C, then h in fact properly has no source; it is not externally visible. The record of source vertices for h through i-1 is destroyed. Vertex h is unliked from list **mostcw**(j), leaving some other vertex as the most CW (or none).

Case B can be recognized by the concavity of $h = B$ with respect to j and i. Case C can be recognized by the fact that the segment of the polygon from i-1 to i intersects the segment of the convex path from j to $h = C$. Case A is recognized as the absence of case B or C.

When the cause is found to be case A, vertices are added to the stack. Unlike the convex hull algorithm, it is not assured that each vertex is added to the stack once and removed at most once. Therefore the algorithm may require more than order n steps. In Fig. 10 we have an example of a pathological object, which requires the CCW rope to wind and unwind itself from the faceted curved surface once for each tooth on the comb structure. With almost $n/2$ facets and $n/4$ teeth, this requires order n^2 steps.

We can loosely define *complexity* as the number of zigs and zags of an object, or the number of turns needed to get to a point deep inside it. *Density* is the number of vertices used in the description of a feature of an object. Then the CCW rope will have to wind and unwind a number of times similar to the complexity, and each winding or unwinding will involve a number of vertices similar to the density. The expected number of steps is roughly the product of complexity and density. Since the number of features is also similar to the complexity, the total number of vertices will be roughly the product of the complexity and the density. The number of vertices and the number of steps required are expected to be of the same order. In the absence of a theory of random polygons, we have not been able to find the expected number of steps in a more rigorous manner.

To test the algorithm, we have run it on 25 000 randomly generated polygons, each having 53 or 58 sides. No errors were encountered.

The rope algorithm has been timed for a wide range of randomly generated polygons, as well as for several demonstration polygons. The number of calls to the CCW function is a good measure of the number of steps the algorithm requires. On polygons from size $n = 10$ to $n = 1000$, the number of calls to CCW was consistently close to $8n$. For comparison, the maximum number of calls to CCW in the convex hull algorithm is $5n$.

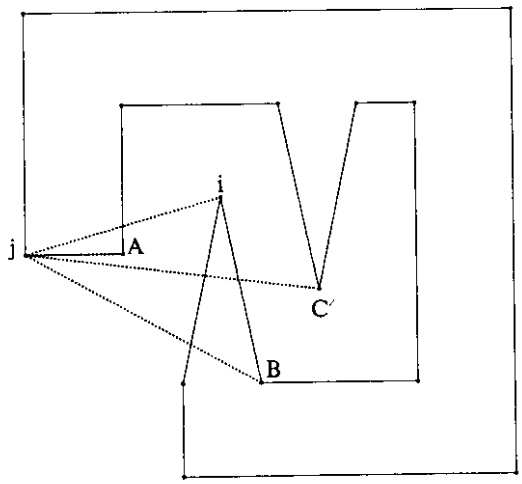

Fig. 9. Three possible situations in which path-patching is needed.

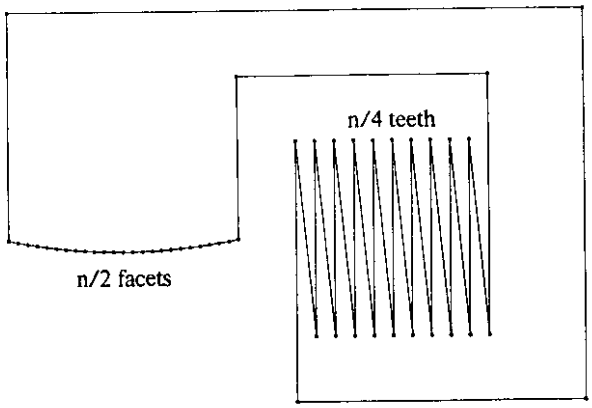

Fig. 10. Pathological object which causes the convex rope algorithm to use its worst case number of steps.

VII. Conclusion

The algorithm discussed above takes a polygon with n sides, and in expected time of order n, identifies all of the externally visible vertices and all of the reachable pairs of vertices. It determines the available clearance of each surface or virtual surface found. It divides the surfaces into two classes, so that by choosing one surface from each to form a grasp, the matter distribution requirement [3] can be fulfilled with no computation.

Previous strategies for grasp selection have been based on grasp generators which produce mostly unreachable grasps, followed by filters to remove the unreachable grasps. In the algorithm described here appropriate geometric constructions are used to avoid generating the unreachable grasps. For objects of substantial complexity, this strategy results in reduced computational complexity for the grasp selection process.

Acknowledgment

The authors acknowledge the helpful suggestions of Randy Brost, Mark Cutkosky, and Marc Raibert.

References

[1] N. Ahuja, R. T. Chien, R. Yen, and N. Bridwell, "Interference detection and collision avoidance among three dimensional objects," in *1st Ann. Nat. Conf. Artificial Intelligence,* 1980, pp. 44–48.

[2] R. A. Brooks, "Planning collision-free motions for pick-and-place operations," *Int. J. Robotics Res.*, vol. 2, no. 4, pp. 19–44, Winter, 1983.

[3] C. Laugier, "A program for automatic grasping of objects with a robot arm," in *Proc. 11th Int. Symp. Ind. Robots,* 1981, pp. 287–294.

[4] T. Lozano-Perez, "The design of a mechanical assembly system," M.S. thesis, Massachusetts Institute of Technology, Cambridge, MA, Dec. 1976.

[5] ——, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, no. 10, pp. 681–698, Oct. 1981.

[6] ——, "Task planning," *Robot Motion Planning and Control.* Cambridge, MA: M.I.T. Press, 1982, pp. 490–493.

[7] ——, "Spatial planning: a configuration space approach," *IEEE Trans. Computers,* vol. 2, no. 2, pp. 108–120, 1983.

[8] M. A. Peshkin and A. C. Sanderson, "A modeless convex hull algorithm for simple polygons," Tech. Rep., CMU-RI-TR-85-8, Robotics Institute, Carnegie-Mellon University, Pittsburgh, PA, 1985.

[9] J. D. Wolter, R. A. Volz, and A. C. Woo, "Automatic generation of gripping positions," Tech. Rep. RSD-TR-2-84, University of Michigan, Ann Arbor, Feb. 1984.

Reachable grasps on a polygon: the convex rope algorithm (pdf file)

M. A. Peshkin and A. C. Sanderson
IEEE Transactions on Robotics and Automation 2:1 (1986)

Abstract:

We describe an algorithm which finds the externally visible vertices of a polygon, and which generates a new geometric construction we call the *convex ropes* of each visible vertex. The convex ropes give the range of angles from which each vertex is visible, and give all the pairs of vertices which are reachable by a straight robot finger. All of the convex ropes can be found in expected time order *n*, where *n* is the number of vertices of the polygon. We discuss the application of this geometric construction to automated grasp planning. The algorithm may also be useful in image interpretation and graphics where efficient computation of visible points is important. The direct application of the algorithm is restricted to two dimensions since sequential ordering of vertices is required. Extention to three dimensions would rely on well chosen intersecting or projective planes.

Keywords:

grasping, reachable, planning, external visibility, polygon, convex rope, convex hull, algorithm, C-space, computational geometry

```
::::::::::::::
ccw.c
::::::::::::::
/* ccw.c   Wed Feb 27 13:09:01 1985    m.peshkin(CMU) */
/* return angle from (k,j) to (j,i)  in  range -pi to +pi */

double ccw(k, j, i)
int k, j, i;
{
    extern int sense, n, x[], y[];
    double sine, cosine, arctan;
    double x0, y0, x1, y1;
    double atan2();

    if (k < 0) {        /* if k is fake (-1) use angle to x axis instead */
        x0 = 1;
        y0 = 0;
    }
    else {
        x0 = x[j] - x[k];
        y0 = y[j] - y[k];
    }

    x1 = x[i] - x[j];
    y1 = y[i] - y[j];

    sine   = x0 * y1 - x1 * y0;
    cosine = x0 * x1 + y0 * y1;
    arctan = atan2(sine, cosine);

    return((float) sense * arctan);
}
::::::::::::::
heart.c
::::::::::::::
/* heart.c   Wed Feb 27 13:09:01 1985   m.peshkin */
/* run rope.c CCW and CW */
```

```
double heart(source, shadow)
    int source[2][2000];               /* rope-origin [CCW/CW][point i] */
    int shadow[2][2000];               /* rope extention stoppers */
{

    extern int n;               /* number of points in object (0 to n-1) */
    extern int sense;           /* +1 for CCW ropes, -1 for CW ropes */
    extern int x[], y[];        /* x, y coords of points */
    int itemp, i;
    double rope();

/* find CW ropes */

    for (i=0; i<=n/2; i++) {            /* renumber object*/

        itemp = x[i];
        x[i] = x[n-i];
        x[n-i] = itemp;

        itemp = y[i];
        y[i] = y[n-i];
        y[n-i] = itemp;
    }

    sense = -1;
    rope(source[0],shadow[0]);          /* CCW ropes algorithm */

    for (i=0; i<=n/2; i++) {            /* renumber object back */

        itemp = x[i];
        x[i] = x[n-i];
        x[n-i] = itemp;

        itemp = y[i];
        y[i] = y[n-i];
        y[n-i] = itemp;
    }

    for (i=0; i<=n; i++) {              /* renumber answers, too */
        if (source[0][i] < 0) source[1][n-i] = -1;
        else source[1][n-i] = n - source[0][i];
        if (shadow[0][i] < 0) shadow[1][n-i] = -1;
        else shadow[1][n-i] = n - shadow[0][i] + 1;
    }

/* find CCW ropes */

    sense = 1;
    rope(source[0],shadow[0]);

    return;
}
::::::::::::::
readob.c
::::::::::::::
/* readob.c   Wed Feb 27 13:09:01 1985    m.peshkin(CMU) */
/* read in points and rearrange; return n */

readob()
{
    extern int n, x[], y[];
    int i, min;

/* read in */

    for (n=0; scanf("%d %d", &x[n], &y[n]) == 2; n++);
```

```
/* find a point on the hull (leftmost) */

    min = 0;
    for (i=0; i x[min] ) {
            min = i;
        }
    }

/* rearrange the list to put that point at 0 and n */

    for (i=0; i<=min; i++) {
        x[n+i] = x[i];
        y[n+i] = y[i];
    }
    for (i=0; i<=n; i++) {
        x[i] = x[i+min];
        y[i] = y[i+min];
    }
    for (i=0; i<=n; i++) {
        x[i+n] = x[i];
        y[i+n] = y[i];
    }
    return;
}
::::::::::::::
rope.c
::::::::::::::
/* rope.c   Wed Feb 27 13:09:01 1985     m.peshkin(CMU) */
/* find convex ropes of vertices of a CCW polygon */
/* based on sklansky scan with bykat detection */

rope(source, shadow)
int source[], shadow[];
{
    extern int n, x[], y[];
    int mcw[2000], vtx[2000], nxt[2000];
    int sp, hul[2000];  /* stack for vertices */
    double cum[2000];   /* cumulative angle of polygon edge (i-1, i) */
    double ang[2000];   /* angle of virtual edge hul[i-1], hul[i] measured */
    double angvpo;      /*   by following the convex path from hul[0] */
    double ccw();       /* ccw(i,j,k) is CCW angle from (i,j) to (j,k) */
    int i, j, m, v, rm, xs, p, q;
    int kill, dead, link, space;

/* initialize */

    space = 0;
    for (j=0; j<=n; j++) mcw[j] = source[j] = shadow[j] = -1;

    hul[ sp = 0 ] = -1;
    ang[ sp = 0 ] = 0.;                 /* (undefined) */
    hul[ sp = 1 ] = 0;
    ang[ sp = 1 ] = 0.;                 /* by definition */
    hul[ sp = 2 ] = 1;
    ang[ sp = 2 ] = ccw(-1, 0, 1);      /* angle (wrt x axis) of (0,1) */

    source[1] = 0;              /* the rope to 1 came from 0 */
    mcw[0] = space++;           /* pointer to the most-CW rope from 0 */
    vtx[mcw[0]] = 1;            /* the most-CW rope from 0 goes to 1 */
    nxt[mcw[0]] = -1;           /* there is no next-most-CW rope from 0 */

    i = hul[sp];                /* to be advanced immediately */
    cum[i] = ang[sp];

/* convex hull algorithm */

    while ( i++ < n ) {
```

```
        cum[i] = cum[i-1] + ccw(i-2, i-1, i);   /* cum angle of (i-1, i) */
        ang[sp+1] = ang[sp] + ccw(hul[sp-1], hul[sp], i);

        if ( ang[sp+1] - cum[i] > .1 ) {        /* bykat? */

            hul[sp] = i;
            sp--;
            ang[sp+1] = ang[sp] + ccw(hul[sp-1], hul[sp], i);
        }

        while ( sp > 1 && ang[sp+1] <= ang[sp] ) {      /* turned right? */

            shadow[hul[sp]] = i;
            sp--;
            ang[sp+1] = ang[sp] + ccw(hul[sp-1], hul[sp], i);
        }

/* patch up convex path */

        while (( m=mcw[ j=hul[sp] ] ) >= 0 && ccw(j, v=vtx[m], i) > 0) {

            /* v is more CW than i. find out if v is removable: */
            /* it is if seg(i-1, i) intersects rope(j, v) */

            p = ((x[i]-x[v])*(y[i-1]-y[i])+(y[v]-y[i])*(x[i-1]-x[i]));
            q = ((x[j]-x[v])*(y[i-1]-y[i])-(y[j]-y[v])*(x[i-1]-x[i]));
            xs = (p == 0) || ((q != 0) &&
                ((float) p/q >= 0.) &&
                ((float) p/q <= 1.));

            /* and it is if v is concave */

            angvpo = ang[sp] + ccw(hul[sp-1], j, v) + ccw(j, v, v+1);
            rm = ( angvpo - cum[v+1] > .1 );

            if ( rm || xs ) {           /* is v removable? */

                /* yes: cut every rope destination back to v */

                mcw[j] = nxt[mcw[j]];
                for (kill=i-1 ; kill >= v; kill--) {
                    dead = source[kill];
                    source[kill] = - v;
                    if (dead < 0) kill = - dead;        /* accelerated */
                }
            }
            else {                      /* no: run down the rope */
                ang[sp+1] = ang[sp] + ccw(hul[sp-1], j, v);
                hul[++sp] = v;
            }
        }

        source[i] = j;                  /* rope to i came from j */
        link = mcw[j];                  /* (linked list maintainance) */
        mcw[j] = space++;
        nxt[mcw[j]] = link;
        vtx[mcw[j]] = i;                /* mcw rope from j goes to i */
        hul[ ++sp ] = i;                /* accept i */
    }

    source[0] = source[n];
    return;
}
```

Text of paper:

IEEE Transactions on Robotics and Automation 2:1 (1986) REACHABLE GRASPS ON A POLYGON: THE CONVEX ROPE ALGORITHM M. A. Peshkin A. C. Sanderson Robotics Institute Carnegie-Mellon University ABSTRACT We describe an algorithm which finds the externally visible vertices of a polygon, and which generates a new geometric construction we call the @i(convex ropes) of each visible vertex. The convex ropes give the range of angles from which each vertex is visible, and give all the pairs of vertices which are reachable by a straight robot finger. All of the convex ropes can be found in expected time order @i(n), where @i(n) is the number of vertices of the polygon. We discuss the application of this geometric construction to automated grasp planning. The algorithm may also be useful in image interpretation and graphics where efficient computation of visible points is important. The direct application of the algorithm is restricted to two dimensions since sequential ordering of vertices is required. Extention to three dimensions would rely on well chosen intersecting or projective planes. KEYWORDS grasping, reachable, planning, external visibility, polygon, convex rope, convex hull, algorithm, C-space, computational geometry Background and Motivation @blankspace(.25 inch) This paper describes an algorithm which efficiently finds the externally visible vertices of a polygon and the range of angles from which each is visible. The associated geometric constructions, called @i(convex ropes), provide much useful information for robot grasp selection and planning involving polyhedral objects. They may also be useful for image interpretation and graphics, where efficient computation of visible points is essential for analysis in terms of image projections. Currently, when a robot grasps an object, the details of the motion have generally been specified in advance by a human programmer with detailed knowledge of the object geometry. If robots are to become less dependent on task-specific programming, grasp selection based on stored geometric models is one of the functions that must be automated. The grasping problem is to find surfaces on an object, and a configuration of a robot's fingers, satisfying three conditions:@cite[Perez81] @begin(itemize) The fingers must be in contact with the object. The configuration must be reachable, i.e., there must be a collision-free path for the robot to get to the grasping configuration. The object must be stable once grasped, i.e., it must not slip during subsequent motion. @end(itemize) Evidently the grasping problem is intimately related to the collision-avoidance problem. However, in collision-avoidance one effective strategy is to find paths which are "safe." Conservative approximations may be made to guarantee that collisions will not occur. For instance, a stationary obstacle may be represented by its convex hull. If a moving part does not intersect the obstacle's convex hull, surely it will not intersect the obstacle. But there may be motions which are deemed unavailable by this method, even though in fact they are collision-free. In grasping, such conservative approximations may be useful in planning approach, but the contact requirement makes them inappropriate for generating the grasps. The grasping configuration itself @i(is) a marginal collision: the robot fingers touch the object. There is also the requirement of stability of the object within the grasp. What may first come to mind here are quantitative physical characteristics of the grasp such as its resistance to slippage when a certain force is applied. However, even more basic stability requirements must be met first, many of which are qualitative geometric characteristics. In this paper we will restrict ourselves to two fingered robots, where the fingers are straight and parallel: the classic two-jaw gripper. In this context, three of the geometric requirements on a grasp are: @begin(itemize) The object must touch the fingers either on the inner surface of both fingers, or o n the outer surface of both fingers, but not one of each. (The "matter distribution requirement"@cite(laugier81).) Surfaces that the fingers are expected to touch must be parallel. Such pairs of parallel surfaces must be reachable from the same side. @end(itemize) Others may be found in@cite[laugier81] and@cite[wolter84]. Previous approaches to automated grasp generation have followed this basic outline@cite[perez82]: @begin(enumerate) Generate a set of candidate grasp configurations, typically by forming all pairs of edges or surfaces. Reject those configurations which do not satisfy the geometric requirements. Reject those configurations which are not reachable by the robot fingers. Choose among those that remain on the basis of stability or other goals. @end(enumerate) For an object with @i(n) edges or surfaces, the number of candidate grasps generated in step 1 is of order @i(n)@+(2). In addition, there is a class of grasps which is not found at all by this strategy. Often a single finger may touch an object at two vertices which are not connected by an object surface. Figure 1 shows a grasp made up of two such @i(virtual surface) contacts (neither of which is a segment of the convex hull of the object.) The existence of this type of grasp was noted (though not treated) in@cite(perez76). To the best of our knowledge, all previous grasp generation procedures would fail to produce the grasp shown in Figure 1. Consideration of such grasps further increases the number of candidate grasps. Steps 2 and 3 require geometric computations comparing each grasping surface with many other surfaces of the object. For intricate objects, these calculations can become unwieldy. Our approach is to integrate some of the grasp selection criteria into the grasp generation procedure, eliminating subsequent testing and rejection. Our approach is to @begin(enumerate) generate only the set of edges, pairs of edges, or surfaces which are reachable by a single infinitely thin robot finger. form pairs which satisfy the geometric requirements, and choose among those on the basis of stability or other goals. @end(enumerate) In the following sections we introduce a new geometric construction, called the @i(convex rope), and an algorithm which computes it. The algorithm efficiently performs step 1. It also provides most of the information needed for step 2. Step 3 will not be addressed in this paper. The algorithm, and the rope construction as well, apply directly only to two-dimensional polygons. Some three-dimensional objects, such as extrusions and punched parts, have one uninteresting axis. Their cross-section contains all the information about the object. In these cases the rope construction can be applied to the cross-section. More complicated three-dimensional objects may be treated by consideration of the intersection of the object with well-chosen planes, yielding two-dimensional figures. This "projective" approach has been applied successfully in other collision-avoidance and grasp selection problems @cite[perez83]@cite[ahuja80]@cite[wolter84]@cite[perez81]@cite[brooks83]. @newpage @section(Introduction) @blankspace(.25 inch) The rope algorithm is based on an order @i(n) convex hull algorithm@cite[Peshkin-hull]. It capitalizes on the orderly traversal made by the convex hull algorithm of all the vertices of a polygon. A similar traversal of the vertices allows the rope algorithm to extract geometric information about the interior pockets and folds of a complicated figure. A vertex of a polygon is @i(externally visible) if a line can be drawn from it to infinity, without intersecting the polygon. Such a vertex can be "seen" from some angle, with the observer in the plane of the polygon. Conversely, if a vertex is not externally visible, it can not be seen from any angle, and, for instance, no attention need be given to it in a graphics display. An entire polygon may be externally visible, but we will be concerned here with finding the externally visible vertices of a general pol ygon. For each externally visible vertex @b(x) there is a continuous range of angles from which it is visible (as in Figure 2). At either end of the range the line-of-sight to the vertex @b(x) is obstructed by some other vertex of the polygon. In Figure 2 these two vertices are labeled @b(a) and @b(b). One of the functions of the rope algorithm is to identify the two obstructing vertices. At either end of the range described above, the line-of-sight grazes the obstructing vertex. As in the example shown (Figure 2), in many cases the line-of-sight intersects some other segment of the polygon, called the @i(shadowed) segment, after grazing the target vertex @b(x). The algorithm also identifies the shadowed segment. Only points of the polygon which are externally visible are reachable by straight fingers in the plane of the polygon. In addition to individual points, some entire edges of the polygon are reachable. If the two vertices which delimit an edge of the polygon are both visible from the angle of that edge, then both can be touched simultaneously by a straight finger at that angle. In this case the finger makes contact with the entire edge. It is also possible

for a pair of vertices which do not delimit an edge to be simultaneously reachable from the angle of the line formed by the two vertices. In this case a straight finger may make contact with both vertices simultaneously, forming a two-point contact with the finger, or @i(virtual surface) contact. In Figure 2 vertices @b(a) and @b(x) form a virtual surface. The algorithm provides the information needed to determine simultaneous reachability of two vertices. It also identifies, for each vertex, the other vertices which are candidates for the formation of a virtual surface contact. Finally, if a finger is in contact with an edge or a virtual surface, its end can extend beyond the end of the edge or the more distant vertex of the virtual surface. It is obstructed only when it extends far enough to intersect the shadowed segment described above, which is also identified by the rope algorithm. @newpage @section(The Rope Construction) @blankspace(.25 inch) Consider a @i(simple polygon) @b(P) described by a sequence of points in the plane @b[(x@-[i], y@-[i])] which are consecutive vertices of the polygon. We will refer to the vertices by their sequence number @b[i], i.e., @i[vertex 3] means the point @b[(x@-[3], y@-[3])]. The only constraints on the points are that the polygon described by them must be closed, and that one edge of the polygon must not intersect another. These requirements give the polygon a well-defined interior. The sequence of vertices may be a @i(clockwise) (CW) or a @i(counterclockwise) (CCW) description of the polygon. In a CW description, the interior of the polygon is to the right of the vector from one vertex to the next; in a CCW description, the interior of the polygon is to the left of the vector. If @b(P) has @b(n) vertices, we define vertex @b[0] to be equivalent to vertex @b[n]. Also, we choose our starting vertex @b[0] so that it is an extremal point in the negative x direction. This insures that @b(0) is on the convex hull. The convex hull of @b(P) is a simple polygon whose vertices are some of the vertices of @b[P]. A description of the convex hull is the sequence of vertices @b[{C@-[k], k=0,1,2 ... m}] which is a subsequence of the integers. Let vertices @b[{0, 1 ... n}] be a CCW description of @b(P). Sufficient conditions for @b({C}) to be a description of the convex hull of @b(P) are: @begin(enumerate) @b[C@-[0]] = @b[0], @b[C@-[m]] = @b[n] @B[{C}] is @i(convex): every vertex @b[C@-[i]] is left of @b[line(C@-[i-2], C@-[i-1])], i.e., makes a left turn. @B[{C}] is @i(external), meaning that if a vertex @b[j] satisfies @b[C@-[i-1] < j < C@-[i]] then @b[j] is left of @b[line(C@-[i-1], C@-[i])]. @end(enumerate) The @i(CCW convex rope) to an externally visible vertex @b[j] is, like the convex hull, described by a subsequence @b[{R@-[k], k=0,1,2 ... p}] of the integers. We require @b[R@-[0]] = @b[0], and @b[R@-[p]] = @b[j]. @b({R}) must be convex and external. Unlike the convex hull, the convex rope is not a closed polygon. An example of a CCW convex rope is shown in Figure 3, where @b(R), the convex rope to @b(j) = @b(9), is @b[{0, 3, 4, 6, 9}] The CCW convex rope is the shortest path from vertex @b[0] to vertex @b[j], external to the polygon, subject to the constraint that it be a CCW path: that is, it must be described by an @i[increasing] subsequence of the integers. The @i[CW convex rope] to a vertex @b[j] is the obvious analogous construction described by a decreasing subsequence of the integers starting with @b[R@-[0]] = @b[n] (which is identical to vertex @b[0]), and ending at @b[R@-[p]] = @b[j]. The CW and CCW convex ropes to vertex @b(j) together constitute a @i(cardioid hull) of the polygon (Figure 4). If @b(j) happens to be on the convex hull, then the cardioid hull and the convex hull are the same. If @b(j) is not on the convex hull, the cardioid hull has a single point (vertex @b(j)) at which it is not convex. The cardioid hull may be thought of as the shortest path around the polygon, external to it, that passes through vertex @b(j). (Removing the requirement that the path pass through vertex @b(j) leaves a definition of the convex hull.) There is a different cardioid hull for each externally visible vertex @b(j), so long as @b(j) is externally visible and is not on the convex hull. The convex ropes have the important property that if @b[{R@-(k), k = 0,1,2 ... p}] is the CCW convex rope to @b(R@-(p)), then the truncated sequence of vertices @b[{R@-(k), k = 0,1,2 ... q}] (where @b(q) < @b(p)) is the CCW convex rope to @b(R@-(q)). It is therefore possible to arrange the externally visible vertices of the polygon in a tree structure (Figure 5). Each vertex @b(j) has a pointer called @b(source@-(j)) to its parent vertex in the tree. @b(Source@-(j)) is the vertex which immediately precedes @b(j) on the CCW convex rope to @b(j). That is, if @b(j) = @b(R@-(p)), @b(source@-(j)) = @b(R@-(p-1)). The entire CCW convex rope to any vertex @b(j) can be read off (backwards) as: @b[{j, source(j), source(source(j)), source(source(source(j))), ... 0}]. Information sufficient to generate the CCW convex rope to @i(all) of the externally visible vertices of the polygon is stored by having a @i(single) pointer from each vertex to its source. Obviously the same applies to the CW convex ropes. In general we need two pointers from each vertex: a @i(CW source) and a @i(CCW source). Recall that the last vertex (@b(n)) of the polygon is equivalent to the first (@b(0)), and that it was chosen to be on the convex hull. The CCW convex rope to @b(n) is the convex hull of the polygon, and @b(source@-(n)) is a vertex on the convex hull. Since the CCW convex rope to vertex @b(0) is a sequence containing only one element, namely @b(0), @b(source@-(0)) is undefined. It is convenient to define @b(source@-(0)) = @b(source@-(n)). This removes the uniqueness of vertex @b(0) as the starting point of all of the ropes. With this definition, we have introduced a loop (Figure 6). into the tree described earlier (Figure 5). The root node of the tree is identified with one of the terminal nodes. In reading off a convex rope (backwards) as: @b[{j, source(j), source(source(j)), source(source(source(j))), ...}] there is no longer a final term. When the term @b(0) is encountered, the sequence goes into a repeating cycle consisting of those vertices on the convex hull. Note that there is nothing unique about vertex @b(0) once @b(source@-(0)) has been defined. There is no way to distinguish vertex @b(0) from any other vertex on the convex hull, by looking only at the looped tree. As can be immediately seen in Figure 4, the rays from vertex @b(j) to its CCW source and CW source define a range of angles. Any ray from @b(j) intermediate between these two rays will not intersect the polygon; any ray outside of their range will intersect the polygon. If @b(j) is lacking a CW source or a CCW source (or both), as for example vertex @b(j+2) in Figure 4, then @b(j) is not externally visible: all rays from @b(j) intersect the polygon. Each pair @b[(j, source@-(j))] is an reachable grasping surface. Some of these are also real surfaces of the polygon, while others, such as the dotted line in Figure 7, are virtual surfaces defined by two points. A single, straight, infinitely thin finger in the plane of the polygon can make contact with the polygon on just these reachable grasping surfaces. One of the geometric requirements imposed on the selection of a @i(pair) of grasping surfaces is that the object must touch the fingers either on the inner surface of both fingers, or on the outer surface of both fingers, but not one of each. This "matter distribution" requirement can be fulfilled by choosing the surface one finger is to contact from among the CW class of ropes, and the other surface from among the CCW class. A surface (or virtual surface) @b[(j, source@-(j))] is matched with a parallel surface from the opposite class. To avoid choosing pairs which are reachable only from opposing sides, one need only note the distinction between parallel and anti-parallel surfaces. Surfaces are parallel if their @i(rays) @b[(j, source@-(j))] are parallel. An additional useful piece of geometric information is available. For each externally visible vertex @b(j), the ray originating at @b(j) and passing through @b(source@-(j)) clearly does not intersect the polygon. The ray originating at @b(source@-(j)) and passing through @b(j) @i(does) intersect the polygon, unless @b(j) happens to be on the convex hull (Figure 7). The segment of the polygon which the ray first intersects is designated @b[(shadow@-(j)-1, shadow@-(j))], and is found by the algorithm. By finding the intersection of the @b[ray(source@-(j), j)] with @b[segment(shadow@-(j)-1, shadow@-(j))], we can also find the amount of @i(clearance) behind the interior point of contact (@b(j)) with the finger. This is just the distance from @b(j) to the point of intersection. This information is needed when attempting to pair the grasping surface with another one. @newpage @section(Convex Hull Algorithm) @blankspace(.25 inch) @begin(flushleft) INPUT: @b[(x@-(i), y@-(i))], for @b[0 @math(@lte) i @math(@lte) n], are the cartesian coordinates of the vertices of the polygon. Vertex @b(0) is extremal in the negative x direction. Vertices @b(0) and @b(n) are equivalent. The polygon is traversed in a CCW sense with increasing subscript. OUTPUT: @b(stack@-(i)), for @b[0 @math(@lte) i @math(@lte) n], are the vertex numbers of the vertices on the convex hull. THE ALGORITHM: @i(definition of function) @b{CCW(k, j, i)}, ranging from -@g(p) to @g(p): if (@b(k) = @b{-1})

@b{CCW(k, j, i)} = the CCW angle from the positive x axis to @b{ray((x@-(j), y@-(j)) (x@-(i), y@-(i)))} if (@b(k) @math(@gte) @b{0}) @b{CCW(k, j, i)} = the CCW angle from @b{ray((x@-(k), y@-(k)) (x@-(j), y@-(j)))} to @b{ray((x@-(j), y@-(j)) (x@-(i), y@-(i)))} @i(initialize) @\ @\stack@-(0) @math(@leftarrow) -1 @\stack@-(1) @math(@leftarrow) 0 @\stack@-(2) @math(@leftarrow) 1 @\pathangl@-(1) @math(@leftarrow) 0 @\pathangl@-(2) @math(@leftarrow) CCW(-1, 0, 1) @\cumangl @math(@leftarrow) pathangl@-(2) @\sp @math(@leftarrow) 2 @i(compute) @\for i = 2 to n @\@\ @\@\cumangl @math(@leftarrow) cumangl + CCW(i-2, i-1, i) @\@\@i(update cumangl and pathangl) @\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp-1), stack@-(sp), i) @\@\ @\@\if pathangl@-(sp+1) - cumangl > .1 @\@\@\@i(test for disagreement) @\ @\@\@\sp @math(@leftarrow) sp - 1 @\@\@\@\@i(reject i-1) @\@\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp-1), stack@-(sp), i) @\@\@\ @\@\while sp > 1 and pathangl@-(sp+1) @math(@lte) pathangl@-(sp) @\@i(test for right turn) @\@\ @\@\@\sp @math(@leftarrow) sp - 1 @\@\@\@\@i(reject stack@-(sp)) @\@\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp-1), stack@-(sp), i) @\@\@\ @\@\sp @math(@leftarrow) sp + 1 @\@\stack@-(sp) @math(@leftarrow) i @\end @newpage @section(CCW Convex Rope Algorithm) @blankspace(.25 inch) INPUT: @b[(x@-(i), y@-(i))], for @b[0 @math(@lte) i @math(@lte) n], are the cartesian coordinates of the vertices of the polygon. Vertex @b(0) is extremal in the negative x direction. Vertices @b(0) and @b(n) are equivalent. The polygon is traversed in a CCW sense with increasing subscript. OUTPUT: @b(stack@-(i)) are the vertex numbers of the vertices on the convex hull. @b(source@-(i)) is the vertex preceding @b(i) on the CCW convex rope to @b(i) @b[(shadow@-(i)@math(-)1, shadow@-(i))] is the segment which first intersects @b(ray(source@-(i), i)) THE ALGORITHM: @i(definition of function) @b{CCW(k, j, i)}, ranging from -@g(p) to @g(p): if (@b(k) = @b{@math(-)1}) @b{CCW(k, j, i)} = the CCW angle from the positive x axis to @b{ray((x@-(j), y@-(j)) (x@-(i), y@-(i)))} if (@b(k) @math(@gte) @b{0}) @b{CCW(k, j, i)} = the CCW angle from @b{ray((x@-(k), y@-(k)) (x@-(j), y@-(j)))} to @b{ray((x@-(j), y@-(j)) (x@-(i), y@-(i)))} @i(initialize) @\ @\stack@-(0) @math(@leftarrow) @math(-)1 @\stack@-(1) @math(@leftarrow) 0 @\stack@-(2) @math(@leftarrow) 1 @\pathangl@-(1) @math(@leftarrow) 0 @\pathangl@-(2) @math(@leftarrow) CCW(@math(-)1, 0, 1) @\cumangl@-(1) @math(@leftarrow) pathangl@-(2) @\sp @math(@leftarrow) 2 @\source@-(1) @math(@leftarrow) 0 @\mostcw(0) @math(@leftarrow) {1} @i(compute) @\for i = 2 to n @\@\ @\@\cumangl@-(i) @math(@leftarrow) cumangl@-(i@math(-)1) + CCW(i@math(-)2, i@math(-)1, i) @\@\@i(update cumangl and pathangl) @\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp@math(-)1), stack@-(sp), i) @\@\ @\@\if pathangl@-(sp+1) @math(-) cumangl@-(i) > .1 @\@\@\@i(test for disagreement) @\ @\@\@\sp @math(@leftarrow) sp @math(-) 1 @\@\@\@\@i(reject i@math(-)1) @\@\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp@math(-)1), stack@-(sp), i) @\@\@\ @\@\while sp > 1 and pathangl@-(sp+1) @math(@lte) pathangl@-(sp) @\@i(test for right turn) @\@\ @\@\@\shadow(stack@-(sp)) @math(@leftarrow) i @\@\@\sp @math(@leftarrow) sp @math(-) 1 @\@\@\@\@i(reject stack@-(sp)) @\@\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp@math(-)1), stack@-(sp), i) @\@\@\ @\@\let v at all times represent the last element of mostcw(stack@-(sp)) @\@\while CCW(stack@-(sp), v, i) > 0 @\@\@\if segment(stack@-(sp), v) intersects segment(i@math(-)1, i) @\@\@\or if pathangl@-(sp) + CCW(stack@-(sp@math(-)1), stack@-(sp), v) @\@\@\@\@\ + CCW(stack@-(sp), v, v+1) @math(-) cumangl@-(v+1) > .1 @\@\@\@\source@-(x) @math(@leftarrow) @math(-)1, v @math(@lte) x < i@\@i[old most-CW is invalid.] @\@\@\@\unlink last element of mostcw(stack@-(sp)) @\@\@\else @\@\@\@\pathangl@-(sp+1) @math(@leftarrow) pathangl@-(sp) + CCW(stack@-(sp@math(-)1), stack@-(sp), i) @\@\@\@\sp @math(@leftarrow) sp + 1@\@\@i[stack@-(sp) is wrong. extend the path] @\@\@\@\stack@-(sp) @math(@leftarrow) v @\@\source@-(i) @math(@leftarrow) stack@-(sp) @\@\link i to mostcw(stack@-(sp)) @\@\sp @math(@leftarrow) sp + 1 @\@\stack@-(sp) @math(@leftarrow) i @\source@-(0) @math(@leftarrow) source@-(n) @\end @end(flushleft) @newpage @section(Description of CCW Convex Rope Algorithm) @blankspace(.25 inch) The CCW convex rope algorithm is built upon an order @i(n) algorithm for finding the convex hull, described fully in@cite[Peshkin-hull]. Both algorithms make use of a @i(cumulative angle) calculated for each side of the polygon. A vector from vertex @b{i-1} to vertex @b{i} will be called @b{vector}(@b{i-1}, @b{i}). We associate the cumulative angle of this vector with vertex @b{i}, calling it @b{cumangl}(@b{i}). @b{Cumangl}(@b{1}) is defined to be the angle of @b{vector}(@b{0}, @b{1}) with respect to the positive x axis. Each subsequent cumulative angle, @b{cumangl}(@b{i}), is obtained from the previous cumulative angle @b{cumangl}(@b{i-1}), by adding to it the CCW angle from @b[vector(i-2, i-1)] to @b[vector(i-1, i)]. If @b{vector}(@b{i-1}, @b{i}) is CCW (i.e., to the left) of @b{vector}(@b{i-2}, @b{i-1}), we assign it a higher cumulative angle. @b{Cumangl}(@b{i}) is just the angle of @b{vector}(@b{i-1}, @b{i}) with respect to the x-axis, but with an integer multiple of 2@g(p) added on. The cumulative angle is @i(not) treated modulo 2@g(p). @b{Pathangl}(@b{j}) is the angle of @b[vector(stack(j-1), stack(j))] with respect to the x axis, with an integer multiple of 2@g(p) added on. Unlike @b{cumangl}, which is computed by following the edges of the polygon, @b{pathangl} is computed by following segments of the convex path defined by the vertices on the stack. The algorithm as presented here finds the CCW ropes for a CCW description of a polygon. By reversing all the comparisons, the CW ropes for a CW description can be found. Of course a CCW description can be transformed into a CW description by reversing the sequence of the vertices. In addition to the simple stack used in the convex hull algorithm, the convex rope algorithm uses several other data structures. The array @b(source@-(i)) contains, at the completion of the algorithm, the @i(source vertex) @b(j) for each externally visible vertex @b(i). For most of the vertices which are not externally visible, @b(source@-(i)) will be undefined (as indicated by a negative value). Under certain conditions a CCW source will be assigned to a vertex @b(i), but no CW source will be assigned when the algorithm is applied to the reversed object, (or vice versa.) Only externally visible vertices have both a CCW and a CW source. The array @b(shadow@-(j)) contains the value of @b(i) at which vertex @b(j) was removed from the stack. Since @b(i) was the @i(first) vertex which could remove @b(j), @b[(k, j, i-1)] must be convex (Figure 8), while @b[(k, j, i)] is concave. Therefore, @b(line(k, j)) must intersect @b(segment(i-1, i)). Since @b(k) is the source vertex for @b(j), vertex @b(i) is @b(shadow@-(j)). Finally, @b(mostcw(j)) contains data used internally by the algorithm. It can be thought of as a set of stacks, one for each vertex @b(j) although for economy of space it is best implemented as linked lists. The @i(most recent) element linked to list @b(mostcw(j)) is the most CW vertex @b(h) (measured from @b(j)), for which @b(source@-(h)) = @b(j). When a new vertex @b(i) is encountered for which @b(source@-(i)) = @b(j), vertices @b(i) and @b(h) are compared to see which is most CW from @b(j). If @b(i) is more CW than @b(h), @b(i) is linked to list @b(mostcw(j)). If, however, @b(i) is not more CW than @b(h), this indicates that @b(j) is not truly the source for both @b(i) and @b(h). There are three distinct cases where @b(i) is not more CW than @b(h), as illustrated in Figure 9. Vertex @b(h) could be @b(A), @b(B), or @b(C), and in each case the new vertex @b(i) is CCW of @b(h). If @b(h) = @b(A), the path to @b(i) intersects the polygon. This results from the removal of vertices from the convex path which must now be reattached. The proper response is to reattach @b(h) = @b(A) to the path between @b(j) and @b(i). If @b(h) = @b(B) or @b(C), then @b(h) in fact properly has no source; it is not externally visible. The record of source vertices for @b(h) through @b(i-1) is destroyed. Vertex @b(h) is unlinked from list @b(mostcw(j)), leaving some other vertex as the most CW (or none). Case @b(B) can be recognized by the concavity of @b(h) = @b(B) with respect to @b(j) and @b(i). Case @b(C) can be recognized by the fact that the segment of the polygon from @b(i-1) to @b(i) intersects the segment of the convex path from @b(j) to @b(h) = @b(C). Case @b(A) is recognized as the absence of case @b(B) or @b(C). When the cause is found to be case

@b(A), vertices are added to the stack. Unlike the convex hull algorithm, it is not assured that each vertex is added to the stack once and removed at most once. Therefore the algorithm may require more than order @i(n) steps. In Figure 10 we have an example of a pathological object, which requires the CCW rope to wind and unwind itself from the faceted curved surface once for each tooth on the comb structure. With almost @i(n/2) facets, and @i(n/4) teeth, this requires order @i(n)@+(2) steps. We can loosely define @i(complexity) as the number of zigs and zags of an object, or the number of turns needed to get to a point deep inside it. @i(Density) is the number of vertices used in the description of a feature of an object. Then the CCW rope will have to wind and unwind a number of times similar to the complexity, and each winding or unwinding will involve a number of vertices similar to the density. The expected number of steps is roughly the product of complexity and density. Since the number of features is also similar to the complexity, the total number of vertices will be roughly the product of the complexity and the density. The number of vertices and the number of steps required are expected to be of the same order. In the absence of a theory of random polygons, we have not been able to find the expected number of steps in a more rigorous manner. To test the algorithm, we have run it on 25000 randomly generated polygons, each having 53 or 58 sides. No errors were encountered. The rope algorithm has been timed for a wide range of randomly generated polygons, as well as for several demonstration polygons. The number of calls to the @b(CCW) function is a good measure of the number of steps the algorithm requires. On polygons from size @b(n)=10 to @b(n)=1000, the number of calls to @b(CCW) was consistently close to 8@b(n). For comparison, the maximum number of calls to @b(CCW) in the convex hull algorithm is 5@b(n). @newpage @section(Conclusion) @blankspace(.25 inch) The algorithm discussed above takes a polygon with @i(n) sides, and in expected time of order @i(n), identifies all of the externally visible vertices and all of the reachable pairs of vertices. It determines the available clearance of each surface or virtual surface found. It divides the surfaces into two classes, so that by choosing one surface from each to form a grasp, the matter distribution requirement@cite(laugier81) can be fulfilled with no computation. Previous strategies for grasp selection have been based on grasp generators which produce mostly unreachable grasps, followed by filters to remove the unreachable grasps. In the algorithm described here appropriate geometric constructions are used to avoid generating the unreachable grasps. For objects of substantial complexity, this strategy results in reduced computational complexity for the grasp selection process. @blankspace(.5 inch) @section(Acknowledgements) @blankspace(.25 inch) The authors acknowledge the helpful suggestions of Randy Brost, Mark Cutkosky, and Marc Raibert.